The Dissertation Committee for Roshan Kumar
certifies that this is the approved version of the following dissertation:

The Vehicle Routing Problem on Tree Networks: Exact and Heuristic Methods

Committee:

Randy Machemehl, Supervisor

S. Travis Waller, Co–Supervisor

Zhanmin Zhang

Anantaram Balakrishnan

Erhan Kutanoglu

Avinash Unnikrishnan

The Vehicle Routing Problem on Tree Networks: Exact and Heuristic Methods

by

Roshan Kumar, B.E., M.S.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2011

For *ammamma*

I wish you were here

Acknowledgments

Writing this dissertation is one of the most challenging tasks that I have undertaken. It would not have been possible without the help and support of the following people.

Dr. S. Travis Waller has been a great adviser and true friend. He gave me the freedom to pursue various research ideas while never letting me lose focus. He never let me get bogged down and his words of advise always had a very calming effect on me. I greatly benefited from Dr. Anant Balakrishnan's courses on network algorithms and supply chain optimization. I am also grateful that he took a keen interest in my dissertation; his contributions improved this dissertation by leaps and bounds. Dr. Erhan Kutanoglu's course on logistics optimization introduced me to the subject and this dissertation's genesis was in the term project for that course. I also want to thank Dr. Randy Machemehl and Dr. Zhanmin Zhang for their invaluable comments and suggestions.

Avi's involvement in my personal and academic life has been immense. From suggesting the topic to providing comments to improve the dissertation, he helped me every step of the way. He somehow managed to effortlessly switch between the roles of friend, guide, confidante, and constant GTalk companion while being the best in all of them. Conversations with Nati made those long days and nights at ECJ and UTA tolerable. Interesting discussions with Lauren and David about life, universe, and everything helped me get through

some otherwise slow days. Steve and Nez, along with other TeQSON members, lent an ear to my ideas and tolerated my incessant complaining. Jen ensured that I didn't have to work on any Tech Memos whilst writing my dissertation. Rishi's attempts at improving my gender-neutral Hindi, albeit futile, were a great source of entertainment, while PD continues to remain a great friend many miles away. Rajesh, Raghu, Binny, and Shashank helped me get over many of my frustrations by joining me in pounding an innocent racquetball against a wall. Tanuj's shared love of Arrested Development, The Big Lebowski, biking, and Gujarati snacks made many a weekend enjoyable.

Libbie Toler and Lisa Macias made sure that I met all my deadlines and received all my travel refunds and fellowship stipends.

I want to take a moment to thank my parents and my sister, Rithika, for their love, support, and encouragement. They always told me to pursue my dreams and facilitated it in every which way they could. No words can express the gratitude I feel towards them. Prakash has played the crucial role of a big brother whom I can look up to perfectly. Lastly, none of this would have been possible without Rachna. She has stood by me through various ups and downs. She allayed my fears, frustrations, and insecurities, but was quick to admonish my pessimism. Hopefully, very soon, Skype will no longer be our preferred method of communication.

I would like to dedicate this dissertation to my late grandmother, Mrs. Parvathi Mani, whom I miss dearly and whose words of wisdom continue to guide me today.

The Vehicle Routing Problem on Tree Networks: Exact and Heuristic Methods

Publication No. _____

Roshan Kumar, Ph.D.
The University of Texas at Austin, 2011

Supervisor: Randy Machemehl
Co–Supervisor S. Travis Waller

The Vehicle Routing Problem (VRP) is a classical problem in logistics that has been well studied by the operations research and transportation science communities. VRPs are defined as follows. Given a transportation network with a depot, a set of pickup or delivery locations, and a set of vehicles to service these locations: find a collection of routes starting and ending at the depot, such that (i) the customer's demand at a node is satisfied by exactly one vehicle, (ii) the total demand satisfied by a vehicle does not exceed its capacity, and (iii) the total distance traveled by the vehicles is minimized. This problem is especially hard to solve because of the presence of sub–tours, which can be exponential in number.

In this dissertation, a special case of the VRP is considered – where the underlying network has a tree structure (TVRP). Such tree structures are found in rural areas, river networks, assembly lines of manufacturing systems,

and in networks where the customer service locations are all located off a main highway.

Solution techniques for TVRPs that explicitly consider their tree structure are discussed in this dissertation. For example, TVRPs do not contain any sub-tours, thereby making it possible to develop faster solution methods. The variants that are studied in this dissertation include TVRPs with Backhauls, TVRPs with Heterogeneous Fleets, TVRPs with Duration Constraints, and TVRPs with Time Windows. Various properties and observations that hold true at optimality for these problems are discussed. Integer programming formulations and solution techniques are proposed. Additionally, heuristic methods and conditions for lower bounds are also detailed. Based on the proposed methodology, extensive computational analysis are conducted on networks of different sizes and demand distributions.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The Vehicle Routing Problem (VRP) is a classical problem in logistics that has been well studied by the operations research and transportation science communities. The VRP aims to design minimum cost delivery routes from a centralized depot or depots to customers at different locations subject to some side constraints. The VRP is known to be NP–hard and is especially difficult to solve because of the presence of sub–tour elimination constraints, which are exponential in number.

In this dissertation, a special case of the VRP is considered – where the underlying network has a tree structure (TVRP). The objective of this dissertation is to develop customized algorithms and solution techniques for some unexplored variants of the TVRP. It is possible to develop faster solution techniques for these problems by explicitly taking advantage of their special network structure.

In this chapter, first, the VRP is described in detail along with a taxonomy of the VRPs that exist in the literature. Next, the TVRP is introduced and some examples of tree networks in the real world are provided. The chapter concludes with an overview of the topics that will be examined.

1.1 The Vehicle Routing Problem

The vehicle routing problem can be defined as follows. Given a transportation network with a depot, $G = (N, A)$; a set of delivery locations, defined to be the nodes, N; non–negative costs between the locations, defined to be costs c_{ij} on arcs (i, j); a set of vehicles, K, to service these locations; and other additional input parameters and characteristics: find a collection of routes starting and ending at the depot, such that

(i) the customer's demand is satisfied,

(ii) a vehicle does not visit a customer more than once, and

(iii) some side constraints are satisfied.

The main objective of the VRP is to minimize the total distance traveled or the total cost of operating the vehicles. In some cases, the total travel cost or travel time is minimized (Laporte, 1992).

The VRP was first studied by Dantzig and Ramser (1959). The problem was defined to have a clover leaf structure because each vehicle serves a subset of customers and returns to the depot. The VRP is a generalization of the Traveling Salesman Problem (TSP) and is NP–hard (Lenstra and Kan, 1981; Frederickson et al., 1976).

Given a fully connected network, the TSP aims to find a single minimum cost tour that spans all the nodes of the network. There are no capacity

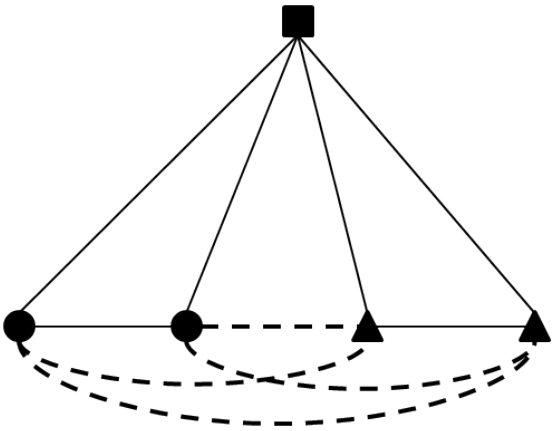


Figure 1.1: Uncapacitated VRP not the same as TSP

or other side constraints. Even in the absence of side constraints, the VRP does not always reduce to a TSP.

Consider Figure 1.1, the cost of traveling from nodes of type $\triangle$ to type $\bigcirc$ is very expensive (say, cost M, denoted by dotted edges here), while all other edges have unit cost. The depot is denoted by a $\square$. A TSP tour will yield a solution of $4+M$, where M is a very large number. On the other hand, if there are two vehicles available, the VRP will yield a solution of 6, with each vehicle serving one node type. So, only a single–vehicle VRP will reduce to a TSP.

1.1.1 VRP Taxonomy

Side constraints are the defining feature of VRPs and add to the complexity of the problem. The side constraints that need to be satisfied depend on the characteristics of the customers and the vehicles (Toth and Vigo, 2002).

The most common side constraints include:

(i) *Capacity Constraints*: Each vehicle has a fixed capacity, and the total demand serviced by the vehicle cannot exceed its capacity. The fleet of vehicles can have homogeneous or heterogeneous capacities. These problems are called Capacitated Vehicle Routing Problems (CVRPs).

(ii) *Distance or Time Constraints*: Each vehicle route cannot exceed a certain pre-specified distance or time, depending on how the arc lengths are described. These problems are abbreviated as DVRPs.

(iii) *Time Windows*: Customers can specify periods of the day (time–windows) during which they can be served. In some cases, along with prescribing time–windows, a service time at the customer location is also specified. These problems are referred to as VRPTWs.

(iv) *Backhauls*: In some instances, customer locations are divided into two subsets – linehauls and backhauls. Linehaul customers require delivery from the depot, whereas backhaul customers require pickup to the depot. A precedence relation which require linehaul customers to be served before backhaul customers is the constraining feature of these problems (VRPBs). This precedence constraint is practically motivated by the fact that vehicles are often rearloaded, and rearrangement of the loads on the trucks at the delivery points is not deemed economical or feasible (Goetschalckx and Jacobs-Blecha, 1993). Another practical reason

is that, in many applications, linehaul customers have a higher priority than backhaul customers (Toth and Vigo, 2002).

(iv) *Pickup and Delivery*: These problems are similar to (iv), except that no precedence relation needs to be satisfied. They are abbreviated as VRPPDs.

(v) *Split Delivery*: The split delivery problem (VRPSD) relaxes the constraint that only one vehicle should serve a customer. Each customer order can be split between multiple vehicles in this case.

Two or more of these side constraints can also be combined to form a single problem.

The common input parameters and characteristics that define the nature of the VRP at hand include:

(i) *Vehicle Fleet Characteristics:* Some logistics providers have vehicle fleets which are heterogeneous and consist of a variety of vehicle types having different capacities. In most instances, the vehicle fleets are homogeneous with all vehicles having the same capacity.

(ii) *Number of Vehicles:* The number of vehicles available at the modeler's disposal can be assumed to be a parameter or a decision variable. When the number of available vehicles equals the number of demand locations, the vehicles are said to be 'free'. In such cases, the total number of used vehicles should be minimized. Conversely, sometimes vehicle fleet size is

'fixed', and the number of vehicle routes is usually equal to the number of vehicles available.

(iii) *Number of Depots:* The VRP can be extended to include multiple depots. Each vehicle route starts and ends at the same depot, and each depot has a fleet of vehicles available at its disposal. In some applications, the locations that each depot serves is determined *a priori*, and a separate VRP is solved for each depot (Toth and Vigo, 2002). Multi–depot VRPs have been used to solve problems related to transporting of school meals (Cassidy and Bennett, 1972), soft drinks (Golden and Wasil, 1987), gasolines, diesel, heating oil (Brown et al., 1987), etc.

(iv) *Dynamic Variants:* In these situations, only partial knowledge of the arc traversal times is known *a priori*. More information is unraveled with the determination of the solution. Also, the arc traversal times are dependent on the number of vehicles using the arc and change with the time period. These problems have emerged as an active area of research due to recent technological advances that allow real–time information to be quickly obtained and processed (Gendreau and Potvin, 1998). A time parameter is associated with the VRP here. Such problems are harder to solve than their static counterparts.

(v) *Stochastic Information:* Stochastic Vehicle Routing problems arise when some elements of the problem are random (Gendreau et al., 1996). Only the probability distributions, but not the actual realizations, of these

elements are known. Information about the demands, arc travel times, or set of customers to be served are assumed to be stochastic here. These problems 'stochasticize' VRPs and are computationally intractable as they combine integer and stochastic programs.

(vi) *Network Structure:* Some transportation networks might have special structures which make them amenable to customized heuristics and solution methods. One example of this are tree networks. Tree networks are found in abundance in road networks and possess special spatial properties as they do not contain any directed or undirected cycles. As a result, it is possible to develop faster and more accurate algorithms for such VRP types.

Figure 1.2 is modified from Toth and Vigo (2002) and Eksioglu et al. (2009), and shows the common VRPs and how some of the side constraints can be combined. Many input elements and parameters are first necessary to define a VRP, these are shown in square boxes to the top of the figure. The most commonly studied VRP types are represented by circles, with corresponding arrows denoting the interactions between various side constraints.

Additionally, there might exist other driver–related and operational constraints. Driver–related constraints account for maximum allowable driving time for a driver, costs accrued due to overtime, mandatory breaks after a certain driving duration etc. Operational constraints include service levels to be met when time–windows are considered, accounting for subsets of arcs

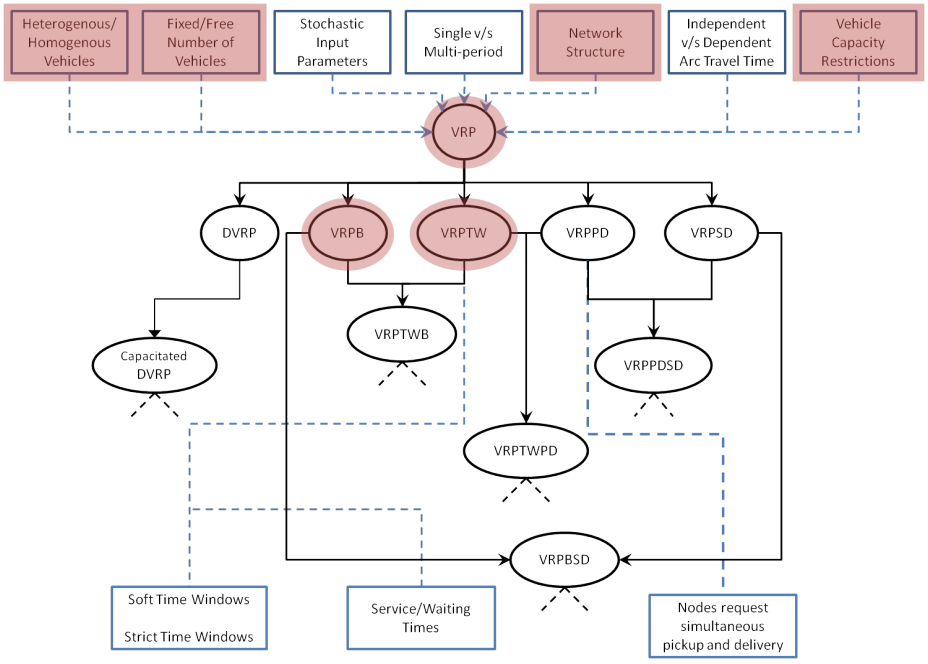


Figure 1.2: VRPs, their interrelations, and variations. The areas of focus are shaded.

(roads) on which certain vehicle types are not allowed, not allowing routes with vehicle load less than a prescribed proportion of capacity etc.

1.2 The Vehicle Routing Problem on Trees

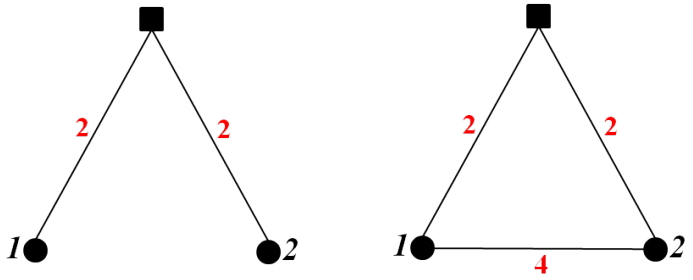
This dissertation examines some variants of the Vehicle Routing Problem on Tree Networks (TVRP). The objective of the TVRP is still to find cost minimizing vehicle routes, but with the added caveat that the network under consideration is a tree. The classical definition of a tree is implied here – networks that contain no directed or undirected cycles. In a tree network there

exists only one unique path between any two pairs of nodes. The capacitated version of the TVRP is NP–hard, while the uncapacitated version reduces to solving the TSP on trees. The TVRP differs from the VRP in that the network in a VRP contains an edge between every pair of nodes. The main complicating factors in TVRPs is that a vehicle can *visit* a node without actually *serving* it. This is not the case with VRPs, where, whenever a node is visited by a vehicle, it is also served. In spite of this, TVRPs are of special interest as the tree structure can be exploited to obtain faster solution methods.

All TVRPs can be solved as VRPs by finding the all–to–all shortest paths between the nodes. In the example shown in Figure 1.3a all edges are bidirectional and the numbers on the edges denote edge costs. There exists no edge between nodes 1 and 2. Therefore, to go from 1 to 2, or from 2 to 1, one must go through the depot. However, once the shortest path from 1 to 2 is added as edge {1,2} (with cost 4), the problem can simply be solved as a VRP as there are now edges between all nodes (Figure 1.3b). This makes the problem much more difficult to solve because of the increased number of variables.

Tree structures are encountered in river networks, railway networks, and rural areas. In rural areas, the absence of infrastructure between every customer location results in a tree structure. In some regions, shortest paths between all pairs of nodes might result in a tree network.

Tree networks also exist in areas where the transportation network consists of a main highway (for example the interstate system) and the customer



(a) The tree

(b) Tree with all–to–all shortest paths

Figure 1.3: Solving TVRP as a VRP

locations are located off the highway. More simplistically, the highway forms the trunk and the roads leading to the service area form the branches of such a transportation network. Sometimes, a group of nodes can be clustered together to form a tree network (Basnet et al., 1999). Although there might arterials and smaller highways connecting these nodes to each other, travel time savings and vehicle fleet restrictions might dictate the use of the main highway system to travel between these nodes.

In some cases, the entire network might not be a tree, but some nodes can be clustered to form a single node resulting in a tree network. The total demand of these clustered nodes is less than the capacity of the vehicle serving it. The overall problem is a TVRP, but a TSP solution within the cluster will determine the order in which the locations within that cluster are served.

Figure 1.4 illustrates a practical case in New Zealand where milk tankers have to transport milk from the dairy supplier to the factory. This case was

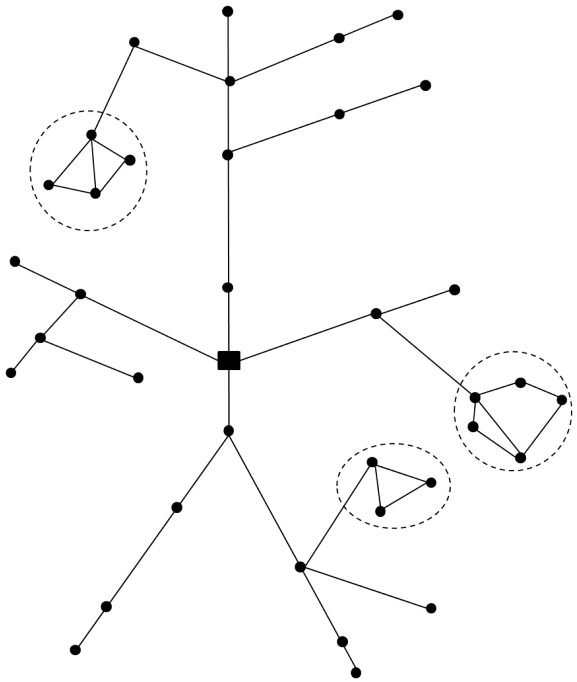


Figure 1.4: Transporting Milk Using Capacitated Tankers from Dairy Farms (◯) to the Processing Factory (□)

studied by Basnet et al. (1999). The problem can be defined briefly as follows. Milk tankers collect milk from the dairy farms and transport it to the factory for processing. The factory owns and operates these tankers and schedules tanker routes such that the total collection cost per kilogram of milk collected is minimized. Each tanker has a known fixed capacity. The amount of milk to be collected from a supplier is also known. Finally, all tanker routes begin and end at the factory. Figure 1.4 shows the network over which this milk collection is carried out. The farms are located off a major highway. Due to the rural location of these dairy farms, there is only one unique path between

two suppliers. It can also be noted that there are a few clusters of farms that are connected to each other and are all located at the end of a public access road leading in from the main highway. The practical case considered was such that the total milk produced by the farm clusters were always lesser than the tanker capacity and as a result could be reduced to a single node. The overall structure, after clustering, results in a tree network with the dairy suppliers forming customer locations or nodes and the factory forming the depot. This practical case illustrates three situations where tree networks might exist – rural areas, clustering of nodes, and customers located off a main highway.

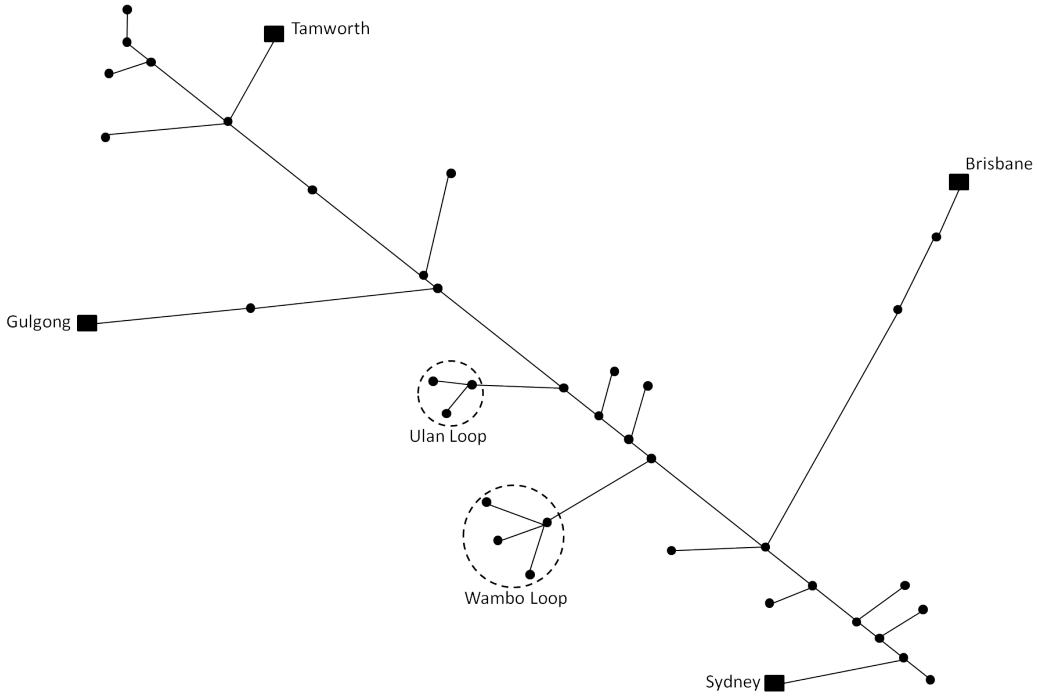


Figure 1.5: Transporting Coal from Mines (◯) to Ports (□) through a Rail Network

Another practical case where tree networks are encountered is in the mine railway system near Sydney, Australia. A co-operative mining logistics agency is responsible for the collection of coals from mines and its transportation to ports for export. The mines are connected by a rail network. The network consists of 35 coal mines owned by 14 individual producers, more than 80 different export blends of coal, and approximately 28 trains making an average of two trips per day. The trains transport the coal from the mines to the ports. Each train has a fixed capacity. The agency plans the train routes so that the total collection cost is minimized. Figure 1.5 shows this railway network. It can be seen that the railway network forms a tree with the port(s) as the depot(s) and the coal mines as customer locations.

Most river networks have a tree structure. For example, the Yangtze river in China has a wide inland river network system that connects various cities within the country. This tree–shaped river network carries about 1 billion tons of freight annually and is the cleanest, cheapest and, widely used mode of transportation (Wan, 2009). The movement of container loads within this network can be modeled as a TVRP.

Finally, outside of transportation and logistics, applications of the TVRP are also encountered in multi–product assembly line balancing problems (Jean-Marie et al., 1992). This problem is briefly defined as follows. Given multiple products that require a set of sequential operations on them (some products may require common operations) , and a set of workstations that can perform any these operations; allot operations to workstations such that (a) the

number of workstations used are minimized, and (b) workstations complete all assigned operations within a given time. Other common objective is to minimize total operating time when number of workstations is fixed (Liu et al., 2008). Furthermore, every operation can be assigned to only workstation.

The TVRP can be used to solve this problem as follows. The precedence relation between the operations can be represented as a collection of trees. Each operation is represented as a customer location, with the demand at the location equal to the total time required to complete operations on all products that require that operation. The workstations are vehicles which are assigned to 'serve' the customer locations subject to precedence constraints.

1.3 Notation

The basic notation that will be used throughout the dissertation is introduced here. Additional notation will be described as and when required.

Let $T = \{N_D, E\}$ denote a tree rooted at the depot. N_D is the set of nodes including the depot. Denote the set of nodes excluding the depot node as N, i.e. $N = N_D \setminus \{depot\}$. The set of edges in the network is represented by E, all edges facilitate movement in both directions. T contains no directed or undirected cycles. Let $|N| = n$ and $|E| = m$, then $m = n - 1$.

The demand at a node $i \in N$ is denoted by d_i. The cost of traversing edge $\{i, j\}$ is given by $c_{\{i,j\}}$. Replace the set of undirected **edges** by the set of directed **arcs** A, such that every edge is replaced by two arcs – one in each

direction – with cost c_{ij} on directed arc (i, j) and $|A| = 2m$. Let **ADJ** be the node–node adjacency matrix for $T = N_D, A)$. Let K be the set of vehicles, Cap denote each vehicle's capacity. Let $k_{LB} \leq |K| \leq k_{UB}$. If $k_{LB} = 1$ and $k_{UB} = n - 1$ then the vehicles are free and if $k_{LB} = k_{UB}$ the vehicles are fixed. It is assumed that $d_i < Cap \forall i \in N$ and that customer demands cannot be split.

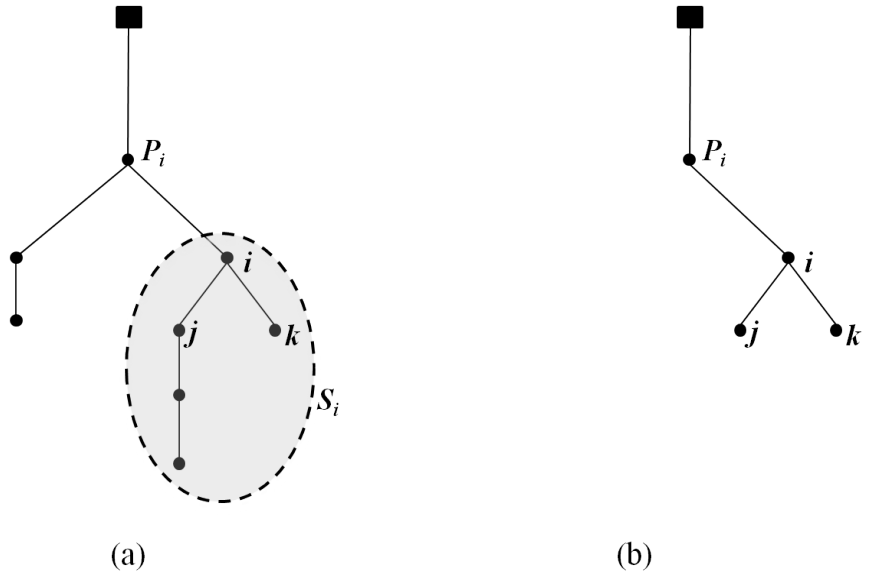


Figure 1.6: An Example Describing the Notation: (a) $c(i) = \{j, k\}$; S_i is the subtree rooted at i; P_i is the parent of i; (b) Minimal Covering Sub–tree with $R = \{j, k\}$

In compliance with other literature on trees, a leaf node of T is defined to be a node with degree 1. For notational convenience, and without loss of generality, assume that the tree grows downward from the root node. That is, all nodes are topologically below the root node. Therefore, the ancestors of a

node are above it, and its descendants are below it. The immediate ancestor of a node is called its parent. For a given node i, its parent is denoted by P_i. Observe that every node has a single unique parent node. The immediate descendant of a node is called its child. The set of children of a node i is given by $c(i)$. A node can have many children. For further notational convenience, assume that the nodes of the tree are numbered in the depth first search (DFS) order. This implies that if $\{i, j\} \in E$ and if $i < j$, then i is the parent of node j. Similarly, if $\{i, j\} \in E$ and if $i > j$, then i is the child of node j. A sub–tree S_i of the tree T is defined as a tree rooted at a node i, such that the nodes i and all its descendants are part of the sub–tree. It is a connected sub–graph of the tree, and all its nodes and edges are also part of the tree T. The example shown in Figure 1.6 shows all the notation explained above. The nodes j and k are the children of node i, and the sub–graph containing the nodes within the darker dashed region is the sub–tree S_i.

The minimal covering sub–tree of a tree is defined as follows. Given a subset of nodes $R \subset N$, the minimal covering sub–tree CS_R is the set of all the nodes in the unique paths from each node in R to the depot. The minimal covering sub–tree will always include the depot and will be rooted at the depot. The set of nodes in CS_R is given by N_R. For example, the minimal covering sub–tree CS_R for the tree in Figure 1.6(a), with $R = \{j, k\}$ is as shown in Figure 1.6(b).

The set of nodes in the unique Path from the Depot to a node i is denoted by PfD_i. That is, $PfD_i = N_R : R = \{i\}$. The set PfD_i contains the

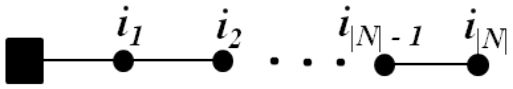


Figure 1.7: A Line Network

nodes in increasing order of index. The set of arcs in the unique path from a node i to j is denoted by $i \to j$. The cost of the path from the depot to i is given by $L_i = \sum_{j \in PfD_i} \sum_{q \in PfD_i} c_{j,q} : (j,q) \in A$.

1.4 Some Observations

In this section some results for simple cases of TVRPs are presented, namely VRPs on lines and VRPs on stars. Consider a line network $T = (N_D, A)$. In this line network, without loss of generality, it can be assumed that the depot is the left most leaf node and the customer nodes are indexed in increasing order of their distance from the depot. A network is said to be a line if its possible to move between any two nodes in the network without having to pass through the depot. A line network is shown in Figure 1.7. Node i_1 is the node with the lowest index and node $i_{|N|}$ is the node with the greatest index. It can be seen that minimizing the total distance traveled by all the used vehicles also minimizes the total number of vehicles.

Now, consider the case where the customer demand can be split between vehicles. That is, a vehicle can serve only a part of a node's demand and some other vehicle(s) can serve the remaining unserved demand. For this case, a

lower bound on the number of vehicles required to serve all the customers can be given as follows.

$$|K| = \left\lceil \frac{\sum_{j \in S_{i_1}} d_j}{Cap} \right\rceil \tag{1.1}$$

That is, the total number of vehicles required to serve all the nodes is equal to sum of the demand of all the nodes in the sub–tree S_{i_1} divided by the vehicle capacity rounded up. Now, in order for obtain a lower bound on the minimum distance traveled by the $|K|$ vehicles, the nodes have to be served in the following order. Starting with the node farthest from the depot (highest indexed node), serve the nodes in decreasing order of their distance from the depot (or decreasing order of node index). When a vehicle is filled to capacity, a new vehicle starts service at the last partially serviced node. This strategy is a lower bound on the distance traveled because the only time an edge is traversed more than twice is when a node demand is partially satisfied, secondly an edge is never traversed more than four times because a node will not be served by more than two vehicles.

The traversal method described above gives a lower bound on the total distance traveled. However, the method also gives a feasible solution to the problem. Therefore, the method described above gives the optimal number of vehicles and the optimal total distance traversed by all the used vehicles.

Consider the case where split deliveries are not allowed. However, assume that all node demands are equal, that is $d_i = d \forall i \in N$. Also assume that

the vehicles are free and have equal capacity, Cap. It is possible to replace vehicle capacity by $\bar{Cap} = \left\lfloor \frac{Cap}{d} \right\rfloor$ and assume unit demand at all the nodes. One can also observe that at optimality, at most one vehicle will have some capacity remaining while all other used vehicles will have no capacity available. This can be examined by assuming that the converse is true and then proving that if the converse were to be true, then the total distance traveled by the vehicles will be higher. Again, the optimal solution will involve serving the nodes in decreasing order of node index till the first node is served.

Now, consider the case where split deliveries are not allowed and node demands are not equal. Assume that vehicle capacities are equal. In this case, the VRP reduces to a bin–packing problem. The proof for the transformation is shown in Labbé et al. (1991); Busch (1990) and is omitted here. Therefore, the VRP on a line where the demands are not equal, but the vehicle capacities are, is NP–hard. As a result, TVRPs are NP–hard too as trees are a generalization of a line network.

The case when the network is a star is now examined. A star network is represented as shown in Figure 1.8. Every edge in a star network contains a node and the depot.

If the demands are not equal (assume equal vehicle capacities), the cost minimization objective in such a star network is redundant as, in order to serve every node, ever edge will be traversed exactly twice. However, the vehicle minimization objective is NP–hard and a simple polynomial transformation reduces the VRP on star networks to a bin–packing problem. The proof is

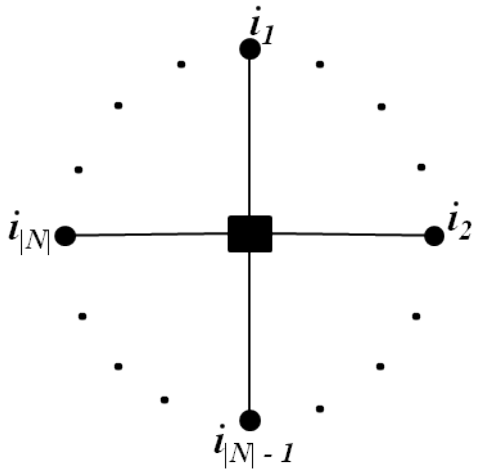


Figure 1.8: A Star Network

detailed in Labbé et al. (1991).

1.5 Dissertation Contributions

This dissertation examines some unexplored variants of the TVRP. The variants shaded in the Figure 1.2 are the focus of this dissertation. More specifically, properties, lower bounds, heuristics, and exact solution methods for the following four variants of the TVRP are discussed: (a) TVRPs with Backhauls (b) TVRPs with Heterogeneous Fixed Fleets, (c) TVRPs with Duration/Time or Distance constraints, and (d) TVRPs with Time Windows. The adaptibility of the solution techniques for different versions of the same variant – capacitated, uncapacitated, multiple vehicle types, cost minimization objective, and vehicle minimization objective – are also demonstrated.

For the different problems considered here, the main setup is similar. Given capacity and other specific side constraints, find a set of cost minimizing vehicle routes. More specifically, given a tree network, $T = (N_D, A)$; non–negative arc costs, $c_{ij} \forall (i,j) \in A$; demand at each node, $d_i \forall i \in N$; and a fleet of vehicles, K, with capacity Cap located at a depot: find a collection of cost minimizing vehicle routes that begin and end at the depot, such that the capacity and service constraints are not violated.

TVRP with Backhauls

In TVRPs with Backhauls (TVRPB), the customers are partitioned into two subsets. The first subset consists of customers who have placed an order for a given quantity of product to be delivered from the depot – the *linehaul* customers. The second subset consists of customers who require a given quantity of product to be picked up from their location and delivered to the depot – the *backhaul* customers. In short, linehaul customers are demand nodes and backhaul customers are supply nodes; and demand is requested from the depot and supply is destined for the depot. Furthermore, in a vehicle tour, the linehaul customers have to be served before backhaul customers. This precedence criterion is practically motivated by the fact that vehicles are often rearloaded, and rearrangement of the loads on the trucks at the delivery points is not deemed economical or feasible (Goetschalckx and Jacobs-Blecha, 1993). Another practical reason is that, in many applications, linehaul customers have a higher priority than backhaul customers (Toth and Vigo, 2002).

TVRP with Heterogeneous Fleets

Usually, large logistics service providers have at their disposal vehicles of different types. Such fleets are called heterogeneous fleets as the vehicles differ in their capacities and operating costs. The TVRP with Heterogeneous Fleets is briefly defined as follows. Given a vehicle fleet mix and customers located on a tree network, find a set of vehicle routes that service the customer demands.

TVRP with Time–related Constraints

In TVRPs which have additional time constraints, the arc costs are assumed to be arc traversal times. Two such TVRP variants are discussed in this dissertation. First is the TVRP with Duration and Capacity constraints. This problem is concerned with finding optimal time minimizing vehicle routes that serve customers that are located on a tree network. Each vehicle is specified a tour time limit which cannot be exceeded. The second is the Capacitated TVRP with Time Windows. In this variant, each customer specifies a deliver/pickup time window. The objective is to find cost/time minimizing vehicle routes which serve the customers during these time–windows only.

1.5.1 Dissertation Organization

The dissertation is organized as follows:

Chapter 2 contains a brief literature survey of VRPs and TVRPs. Many papers and books have been written on the subject (for example, Toth

and Vigo (2002), Laporte (2009), Christofides (1985), Bodin et al. (1983), Bodin (1990), Fisher (1995), Laporte (1992), Golden et al. (2008) etc.). The literature that is most pertinent to the topics in the dissertation is reviewed here. First a literature review of classical VRP heuristic and exact solution techniques is presented, followed by an extensive literature survey on TVRPs.

Chapters 3 and 4 deal with the TVRP with Backhaul Customers. Chapter 3 describes exact and heuristic methods to solve this problem. A few properties and observations and conditions for a lower bound are first discussed. Using these, an exact solution method is proposed. Another solution method, based on a network transformation technique, requiring fewer variables than the first one is also proposed. An optimal algorithm for solving the Traveling Salesman Problem with Backhauls on Trees is presented next. This algorithm is used in the cluster–first route–second heuristic. Computational results are presented in Chapter 4. Based on these results, a few improvements, including another heuristic, are discussed.

Chapter 5 details the exact and heuristic technique developed for the heterogeneous TVRP. In this chapter, some exact solution methods and their relation to the Generalized Assignment Problem and Bin Packing Problem are discussed. No heuristics or approximation algorithms for TVRPs exist in the open literature that deal with either heterogeneous

fleets or fixed fleets. The heuristic is capable of dealing with non-fixed and homogeneous fleets. The heuristic proposed has four steps, one of which is solving the Generalized Assignment Problem. The computational results obtained by implementing the heuristic and exact methods on networks of varying sizes and demand distributions are also presented.

Chapter 6 describes new heuristics for two variants of the TVRP – vehicle time or duration constraints and customer time–window constraints. Exact and heuristic methods and other details of the Duration Constrained Capacitated Vehicle Routing Problem on Trees (DTCVRP) are described. The heuristic is a modification of the Savings Algorithm adapted to tree networks. The heuristic for the time–windows variant is presented next. Computational results obtained by implementing these heuristics are also described.

Chapter 2

Literature Review

The VRP was first studied by Dantzig and Ramser (1959). The problem was defined to have a clover leaf structure because each vehicle serves a subset of customers and returns to the depot. Since then, many papers and books have been written on the subject (for example, Toth and Vigo (2002), Laporte (2009), Christofides (1985), Bodin et al. (1983), Bodin (1990), Fisher (1995), Laporte (1992), Golden et al. (2008) etc.). This chapter attempts to overview the literature that is most pertinent to the topics in the dissertation. First a literature review of classical VRP heuristic and exact solution techniques is presented followed by an extensive literature survey on TVRPs.

2.1 Literature Review – VRP

2.1.1 Exact Solution Methods

Some exact solution methods to solve VRPs include integer programming formulations, branch–and–bound schemes, dynamic programming approaches, and set partitioning based formulations.

Integer Programs

The most common integer programs used to solve VRPs are the vehicle flow based formulations. They include the three–index formulation proposed by Fisher and Jaikumar (1981a) and the two–index formulation proposed by Laporte et al. (1985).

Given a network $G = (N, A)$, the three–index vehicle flow formulation for CVRPs, when the vehicles are fixed and homogeneous, is given below. The depot node here is represented as 0. The formulation is as follows:

Variables:

$$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ traverses arc } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{if vehicle } k \text{ serves node } i \\ 0 & \text{otherwise} \end{cases}$$

Minimize:

$$\sum_i \sum_j \sum_k c_{ij} x_{ijk} \tag{2.1}$$

Subject to:

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N \setminus \{0\} \tag{2.2}$$

$$\sum_{k \in K} y_{0k} = |K| \tag{2.3}$$

$$\sum_{j \in N} x_{ijk} = \sum_{j \in N} x_{jik} = y_{ik} \qquad \forall i \in N, k \in K \tag{2.4}$$

$$\sum_{i \in N} y_{ik} d_i \leq Cap \qquad \forall k \in K \tag{2.5}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \qquad \forall S \subset N \setminus \{0\}, |S| \geq 2, \forall k \in K \tag{2.6}$$

$$x_{ijk},\ y_{ik} \in \{0, 1\} \qquad \forall (i,j) \in A, k \in K \tag{2.7}$$

Equations 2.2 – 2.5 impose that a node is served by only one vehicle; all $|K|$ vehicles leave the depot; if vehicle k serves node i, then that same vehicle k will also leave node i; and the sum of demands of all the customers served by a vehicle is less than its capacity. Equation 2.6 is the sub–tour elimination constraint. It states that given a subset of nodes S, if a vehicle k enters that subset than it will traverse on only $|S| - 1$ arcs in that subset. This prevents subtours because if a vehicle was to travel on $|S|$ or more arc within that subset, then it will definitely form a cycle in that subset.

Miller–Tucker–Zemlin (Gilbert et al., 1991; Miller et al., 1960; Kara et al., 2004) proposed a subtour elimination constraint that was not as tight as the one reported above. The M–T–Z constraint has been used in formulations for VRP with time windows.

Fisher and Jaikumar (1981a) noted that the three–index formulation has embedded within it the generalized assignment problem. A Bender's decomposition based solution technique was used to solve this problem. The master problem solves the assignment problem that assigns vehicles to customers and a sub–problem determines the vehicle route for every assignment. The algorithm guarantees an optimal solution when run to completion.

Laporte et al. (1985) stated the sub–tour elimination constraints using some lower bounds on the bin–packing problem. They also introduced various valid inequalities to the formulation to strengthen it and to reduce convergence time. A constraint relaxation method is used to solve the problem using this formulation. The method first solves a sub–problem that contains only a subset

of the constraints and relaxes the harder constraints. The method extracts a solution from the sub–problem at every step and checks its feasibility with a previously known best solution value. The algorithm also checks for subtour violations at every iteration.

Branch–and–Bound Schemes

Christofides et al. (1981a) developed a lower bound for symmetric VRPs called the k–degree center tree, which was based on the m–TSP relaxation of a VRP. They divided the edges that form a VRP solution into four subsets. Edges forming a k–degree center tree, edges incident to the depot node, edges not incident to the depot but part of the solution, and edges not in the solution. Then, using these subsets and their properties they developed an integer programming formulation of the problem. Lower bounds are computed by computing the k–degree center while the other constraints are relaxed within a lagrangean framework. A branch–and–bound algorithm is used to compute the optimal solution.

Fisher (1994b) similarly used minimum k–trees to obtain optimal solutions to the VRP. First, a minimum k–tree is computed for the given network. If the network has n nodes, then the tree spanning the graph with $n-1+K$ edges is called the minimum k–tree. The degree of the depot is $2K$. The side constraints – vehicle capacity, and each customer being served by only one vehicle– are dualized to obtain a lagrangean relaxation problem. The lower bound thus obtained in embedded in a branch–and–bound framework

to obtain the optimal solution. The k–tree is computed using the algorithm provided by Fisher (1994a).

Other branch–and–bound techniques have also been proposed by (Laporte and Nobert (1983), Houck (1978), Laporte et al. (1986), and Hadjiconstantinou et al. (1995)).

Set Partitioning Formulations

Set partitioning is a popular way of formulating problems which have to be evaluated over an excessive number of states. Column generation is usually used to solve such problems. Column generation tries to evaluate only the most 'profitable' states iteratively and thereby reduces the computation effort required to solve the problem. At every step a sub–problem is solved which introduces the most viable column into a master problem. These techniques are usually embedded within a branch–and–bound framework. Usually within the VRP context, the sub–problem reduces to finding a shortest paths with or without additional side constraints depending on the problem structure. Polynomial and pseudo–polynomial algorithms are used to solve these shortest paths. Balinski and Quandt (1964), Rao and Zionts (1968), Foster and Ryan (1976), and Orloff (1976) have all studied set partitioning approaches to solve VRPs.

Recently, set partitioning approaches have be studied by Letchford et al. (2002), Lysgaard et al. (2004), and Baldacci et al. (2008). Some of these new techniques solve the k–degree center tree (Christofides et al., 1981a) as a sub–

problem. Letchford et al. (2002) and Lysgaard et al. (2004) also suggest various valid inequalities that can be added to the master problem, whose relaxation is solved and then embedded within a branch–and –bound or branch–and–cut framework.

Dynamic Programming Approaches

Dynamic Programming formulations and recursions to solve CVRPs were suggested by Eilon et al. (2007) and Christofides et al. (1981b). The main idea of these dynamic programming formulations is that they find minimum cost routes for a single vehicle over a subset of routes. All the possible subsets are evaluated and the function is minimized over these subsets. Since all subsets are evaluated, the number of states of the dynamic program are exponential in nature. Christofides et al. (1981b) suggest some state relaxation procedures to reduce the total number of states over which the DP recursion is examined.

2.1.2 Heuristic Methods

Many heuristic methods exist to solve CVRPs. The classical and most popular ones will be discussed here. Additionally, the state of the art in VRP solves CVRPs using various metaheuristics. This dissertation does not focus on metaheuristic approaches, and hence, they are not discussed.

Clarke and Wright Savings Algorithm

As the name suggests, this algorithm was developed by Clarke and Wright (1964a), and is one of the most commonly used CVRP algorithms. Laporte (1992) report that this algorithm provides solutions of reasonably good quality. It is a route–first cluster–second type heuristic. First, a route is constructed to every node in the network. Merging of these routes is then considered if there is a savings in doing so. While merging routes, capacity and duration constraints are considered, and only feasible merges are executed. The algorithm terminates when no more merges are possible. The details of this algorithm, implemented within the context of TVRPs is presented in the next section. Many researchers (for example, Laporte and Semet (2002), Golden et al. (1977)) have tested, and suggested enhancements and improvements to the original heuristic.

Cluster–First Route–Second

This heuristic works conversely to the heuristic by Clarke and Wright (1964a). First, a cluster of nodes that every vehicle route is determined, next a TSP is solved for each of these clusters to determine the exact vehicle route. Many approaches have been used to cluster the vehicles – with bin–packing being one commonly used technique. Fisher and Jaikumar (1981a) suggest an approach based on solving the generalized assignment algorithm to determine the vehicle clusters.

Sweep Heuristic

A very simple sweep heuristic was suggested by Wren and Holliday (1972). The nodes of the network are represented as polar co–ordinates. At each iteration a vehicle is chosen, then a sweep is initiated to find and serve an unrouted node at the least angle with respect to the depot. The sweep continues till capacity constraints are not violated. After the vehicle clusters are determined in this manner, routes are constructed by solving a TSP for each vehicle route.

In addition to these many other heuristics exist for solving VRPs. Some heuristics optimize one route at a time, others consider all routes together. Some edge swapping heuristics are also described in Laporte (2009). Additionally, set partitioning heuristics that develop feasible routes called 'petals' have also been studied by Laporte (1992).

2.1.3 VRPs with Backhauls

In VRPs with Backhauls (VRPB), the customer nodes are divided into two subsets – L, linehaul nodes and B, backhaul nodes. The capacitated version of the problem aims to find cost minimizing vehicle routes such that:

(i) each vehicle routes begins and ends at the depot

(ii) each customer node is visited only once

(iii) sum of linehaul nodes serviced by a vehicle does not exceed capacity, and similarly for backhaul nodes

(iv) linehaul nodes are visited before backhaul nodes

(v) each route should service at least one linehaul node

Exact and heuristic methods for this problem have been proposed by Mingozzi et al. (1999), Goetschalckx and Jacobs-Blecha (1993), Anily (1996), and Toth and Vigo (1997).

Toth and Vigo (1997) provided an integer program to solve this problem. They divide the set of edges (arcs) into three subsets – arcs arcs between linehaul nodes, arcs between backhaul nodes, and arcs between linehaul and backhaul nodes. A backhaul node can never be served before a linehaul node, so these arcs are not considered while solving the problem. In their formulation, they propose capacity cut constraints (CCCs) that impose subtour elimination and capacity restrictions. These CCCs are then relaxed and the resulting transportation or assignment problem is solved to obtain a lower bound on the problem. The cuts can also be dualized and the resulting lagrangean lower bound can be embedded into a branch–and–bound framework to obtain an optimal solution. Toth and Vigo (2002) report that the lower bounds obtained from the assignment problem are of inferior quality and can be improved by combining them with the lagrangean lower bounds.

Mingozzi et al. (1999) provided a set partition formulation to solve the VRPB. They define linehaul and backhaul paths that don't exceed capacity, and formulate an IP using these paths. They propose to solve the problem

using solvers, but by reducing the number of variables by solving master and sub problems to determine the variables that need to be considered.

The savings algorithm by Clarke and Wright (1964a) was modified by Deif and Bodin (1984a) for solving heuristically solving VRPBs. Goetschalckx and Jacobs-Blecha (1993) proposed a heuristic that clustered linehaul and backhaul customers separately, and then merged these clusters together to form a single vehicle route. Goetschalckx and Jacobs-Blecha (1993) also proposed a cluster–first route–second heuristic similar to that of Fisher and Jaikumar (1981a). Finally, a cluster–first route–second type heuristic was presented by Toth and Vigo (1996). Their heuristic included some route re–optimization procedures that improve the heuristic solution quality.

2.1.4 VRPs with Time Windows

VRPs with Time Windows (VRPTW) are a class of VRPs that have been studied extensively. Many exact and heuristic techniques for solving this problem exist in the literature. Here, exact solution methods that use lagrangean relaxation and column generation procedures and heuristics are discussed. The problem involves routing a set of vehicles to service demand at customer locations such that the vehicles adhere to the pre–specified pickup or delivery time windows imposed by the customer. In other words, a vehicle can serve a customer node only within the time window that has been specified. The arc costs denote travel time and total travel time over all vehicles must be minimized. Customers may impose hard or soft time windows. Soft time

windows can be violated at a penalty cost. In the case of hard time windows, if a vehicle arrives at a node before the time window, then the vehicle is forced to wait till the service time window. Additionally, a fixed service time can be associated with each node.

The integer program for the VRPTW (Toth and Vigo, 2002) is formulated as follows. The graph is modified to include two copies of the depot node. One at the start of the route, and one at the end. Also, infeasible arcs are excluded from the graph. Infeasibility can arise when the sum of the demands of two nodes exceeds vehicle capacity, or when the time windows are such that traveling on that arc will result in time window violations. Apart from the usual capacity and service constraints, time window and service time constraints are also added to the formulation. The sub–tours are avoided implicitly by requiring vehicles to travel from one depot copy to the other, and by enforcing that a vehicle visit a particular node only once.

Heuristics for VRPTWs mainly consist of two steps – route construction and route improvement. An overview of these algorithms can be found in Braysy and Gendreau (2005) and Tan et al. (2001). Route construction consists of building a feasible route by adding one vehicle at a time to the vehicle route. These operations are performed sequentially or in parallel. Maximum savings and other time window related constraints are considered in such insertion heuristics. In the route improvement step, the incumbent solution is improved by performing local searches using swapping heuristics.

Exact algorithms based on state space relaxation techniques using dy-

namic programming recursions was developed by Christofides et al. (1981b). Lagrangean relaxation techniques for solving VRPTWs was suggested by Fisher et al. (1997). The lower bound obtained by the lagrangean problem was then embedded in a branch–and–bound algorithm. Fisher et al. (1997) also developed a relaxation based on the minimum k–trees with a time window side constraint for the VRPTW. Fisher and Jaikumar (1981a) relaxed the time window related constraints and solved a resulting generalized assignment problem to obtain lower bounds.

Desrochers et al. (1992) and Kallehauge et al. (2005) developed column generation techniques based on the set partitioning formulation for solving VRPTWs. The decomposition results in a master and sub–problem, with the sub–problem resulting in a constrained shortest path problem with time windows. Embedding this within a branch–and–bound framework results in integer solutions. Other algorithms and a unified framework for time constrained problems have been suggested in Desaulniers et al. (1994).

2.1.5 VRPs with Heterogeneous Fleets

The Heterogeneous VRP (HVRP) can be solved to optimality by trivially modifying the network flow–based integer formulation for the capacitated VRP on general networks first presented by Laporte et al. (1985) and Fisher and Jaikumar (1981b). Fisher and Jaikumar (1981b) also provided a cluster–first route–second heuristic based on the GAP and TSP for solving capacitated VRPs on general networks. With some modifications, their algorithm can be

applied to the heteregeneous VRP. An exact solution method based on solving the LP–relaxation using a column generation method was proposed by Choi and Tcha (2007). Heuristics for HVRPs based on existing VRP techniques have been studied by Golden et al. (1984); Salhi and Rand (1993); Taillard (1999); Desrochers and Verhoog (1991); Renaud and Boctor (2002).

2.2 Literature Review – TVRP

The Vehicle Routing Problem on Trees was introduced by Labbé et al. (1991). It is a special case of the VRP when the network is constrained to a tree. The capacitated TVRP (TCVRP) and all its variants were shown to be NP–hard by Labbé et al. (1991) and Hamaguchi and Katoh (1998) by transformation of the bin–packing problem (shown to be NP–complete by Garey and Johnson (1979)) to a special case of the TVRP.

2.2.1 Lower Bounds

As the TCVRP is closely related to the bin–packing problem, most of the lower bounds presented in the literature use this relation to find lower bounds. The bin–packing problem is defined as follows. Given a set of p items with weights $w_1, w_2, \ldots, w_p$; and bins of capacity Q – pack the p items into bins such that the total number of bins are minimized. A simple lower bound on the number of bins required to *pack* all p items is given by:

$$\left\lceil \frac{\sum_{i=1}^{p} w_i}{Q} \right\rceil \tag{2.8}$$

Now, consider a sub–tree S_i rooted at node i, the minimum number of vehicles required to serve the nodes, say, $bin(V_i)$ is given by:

$$bin(V_i) = \left\lceil \frac{\sum_{j \in S_i} d_j}{Cap} \right\rceil \tag{2.9}$$

Proposition 2.1. *A lower bound, $\underline{z}$, on the optimal objective value, z^*, to the TCVRP is given by:*

$$\underline{z} = 2 \sum_{i \in N} bin(V_i) c_{\{P_i, i\}} \tag{2.10}$$

Proof. As described above, the minimum number of vehicles requires to serve S_i is $bin(V_i)$. Therefore, at least $bin(V_i)$ vehicles will enter the sub–tree S_i to serve its nodes. This implies that at least $bin(V_i)$ vehicles will traverse the edge $\{P_i, i\}$ once on their way from the depot to the sub–tree, and once on their way back (after service) from the sub–tree to the depot. □

These lower bound results are used in Labbé et al. (1991), Basnet et al. (1999), Mbaraga et al. (1999), and Hamaguchi and Katoh (1998). The lower bounds on bin–packing problems can be refined by the techniques suggested by Martello and Toth (1990). They suggest improved lower bounds by considering the fact that all items with weights between $\frac{Q}{2}$ and Q must belong to separate bins. They reported a worst case performance of 2/3 on their bound. These lower bound refinements were used by Busch (1990). Labbé et al. (1991) improved the lower bound by observing that no more than two items with weights between $\frac{Q}{3}$ and $\frac{Q}{2}$ can fit in the same bin. Fekete and Schepers (2001) used dual feasible functions to obtain lower bounds with a

worst case performance guarantee of 3/4. They showed that the bound by Martello and Toth (1990) is a special case of their procedure.

2.2.2 Heuristics

This section details the various heuristics that have been employed by researchers to solve various variants of the TVRP. All these heuristics work towards obtaining a feasible upper bound on the optimal solution value to the problem.

Labbé et al. (1991) proposed a linear time (in the number of nodes) heuristic for the TCVRP. Their procedure is as follows. At each step a leaf node is chosen. If the demand at the leaf node and the parent node combined is less than the vehicle capacity, the leaf node is merged with the parent node, and both the nodes are assigned to the same vehicle. The leaf node is deleted from the tree. If the sum of the demands of the leaf node and its parent node exceed the vehicle capacity, then two cases exist. Either the leaf node demand is greater than the parent node demand, or vice versa. In the former case, the leaf node is assigned to a vehicle and is deleted from the tree. In the latter case, the parent and leaf node positions are swapped, a vehicle is assigned to the new leaf node and the leaf node is deleted. A vector of vehicle assignments is maintained to record the subtrees in each vehicle. A distance variable keeps a record of the total distance traveled by all the vehicles at each stage. This distance variable, on heuristic termination, returns the heuristic solution value (upper bound). The procedure terminates because a leaf node

is deleted at every step from the tree. It's complexity is $O(n)$ because every node is examined only once.

Rennie (1995) modified this linear time algorithm as follows. Instead of stopping the merging of nodes when capacity is violated, Rennie (1995) suggested that the corresponding vehicle *bin* be moved up the tree till a node closer to the root is encountered which can be feasibly merged.

Basnet et al. (1999) developed two heuristics for the TCVRP. The first heuristic, *H1*, is based on the savings heuristic developed by Clarke and Wright (1964a). It starts by first constructing as many vehicle routes as there are nodes, and then combines these routes by calculating the maximum savings that can be obtained by doing so. This method proceeds from an expensive feasible solution to a cheaper feasible solution. The second heuristic, *H2*, first assigns all nodes to a single vehicle and then subdivides these nodes into different vehicles such that capacity constraints are satisfied. *H2* moves from a cheaper infeasible solution to a more expensive feasible solution.

Consider three nodes i, j, and k. The root node is denoted by 1. Let vehicles Veh_i and Veh_j serve nodes i and j respectively. Let k be the last common node on vehicle routes Veh_i and Veh_j. Finally, let s_{ij} be the shortest distance between nodes i and j. Then, a maximum savings of $2s_{1k}$ can be obtained by merging vehicle routes Veh_i and Veh_j, such that capacity constraints are not violated. *H1* proceeds as follows. First a every node is assigned its own vehicle. An associated node of a vehicle is the last common node on the vehicle which has been considered as a merging point for that

vehicle route. So, in the first iteration, the associated node of each vehicle is the node it has been assigned to. A list of vehicle routes, in non–increasing order of their associated node's distance from the depot is created. Two vehicles with the same associated node are considered for merging such that the total capacity constraints are satisfied. If a vehicle route cannot be merged with any other route, its associated node is changed to the next node on the vehicle route that is farthest from the depot. At every iteration vehicle routes are merged and associated nodes are modified. At the end of every iteration, the table is resorted. The algorithm ends when every vehicle's associated node is the root node.

In heuristic *H2*, a vehicle route with all the nodes is first created. The route is now split into two routes at the node that is closest to the depot. This node is called the branching node. The branching node is not part of either route. At every iteration, the largest unassigned route is split. Then, all the routes are inspected for capacity violations. If a vehicle route satisfies capacity constraints, it is kept aside. The other routes move to the next iteration. After all nodes are assigned to permissible routes, the branching nodes are considered in non–increasing order of their distance from the root node. They are assigned to existing routes that pass through them, subject to capacity restrictions. If a branching node cannot be assigned to any route, a new route is created.

Basnet et al. (1999) compared their algorithms *H1* and *H2* with the algorithms of Rennie (1995) and Labbé et al. (1991). Although they did not

provide any worst case bounds on their algorithm, they observed that *H1* performed better than the other three algorithms. However, the computation times reported for *H1* were significantly higher than the other algorithms. *H1* considers all routes together, and attempts to merge them based on the total savings achieved. This explains the better solution quality.

Hamaguchi and Katoh (1998) developed a 1.5–approximation algorithm for the TCVRP. The algorithm assumes that the demand at each node, $d_i < 1$. A node, i, is $D-feasible$ if $d(S_i) = \sum_{j \in S_i} d_j \geq 1$, and is $D-minimal$ if i is $D-feasible$ but none of other nodes in S_i are. Moreover, it is assumed that each sub–tree can be further divided into sub–trees that satisfy $\sum_{i=1}^{k-1} d(S_i) < 1$ and $\sum_{i=1}^{k} d(S_i) \geq 1$. Furthermore, it is shown that $k = 2$. When a $D - minimal$ node is encountered , one of two strategies to serve the vehicles is used. The first strategy utilizes two vehicles to serve two sub–trees of the $D - minimal$ sub–tree and computes the cost for doing so. The second strategy further divides one sub–tree such that the total demand of one part of the divided sub–tree and the undivided one exactly sums to 1. It then uses two vehicles to serve these two sub–trees. Relations are given to decide which strategy to use. Given the way the sub–trees are defined and divided, the customer demands can be split in this algorithm (TCVRPSD). The average ratio (compared to the lower bound) for 60 instances was 1.013, and the worst–case ratio was 1.072.

A 1.35078-approximation algorithm for the TCVRP was developed by Asano et al. (2001). The same concepts of $D-feasibility$ and $D-minimality$

are used in this algorithm. The first step of the algorithm involves performing a set of seven reforming operations to reshape the tree. These reformations are done such that the lower bound of the problem remains the same. Some reforming operations include contracting sub–trees with total demand < 1 into a single node, removing demand from all internal nodes, ensuring all non–grandparent nodes have only one child (leaf) node etc. $D - feasibility$, for a node i, is redefined here as $d(S_i) = \sum_{j \in S_i} d_j \geq 2$. Further, the nodes that are $D - minimal$ are called $q - nodes$. Now, the strategies described by Hamaguchi and Katoh (1998) are used and appropriate serving strategies are determined. When there are no more $q - nodes$ a final strategy is applied whose worst–case ration does not exceed $(\sqrt{41} - 1)/4 = 1.35078$.

Katoh and Yano (2006) presented a 2–approximation algorithm for the TCVRP with pickup and delivery. They assumed splittable demand, and developed an algorithm which finds a set of feasible vehicle tours such that at any time during the tour, the sum of the goods to be delivered and picked up does not exceed the vehicle capacity. That is, they do not consider that all delivery takes place before pick up. Their algorithm consists of two main steps. In the first step, they perform a set of seven reforming operations similar to those performed by Asano et al. (2001) . In the second step, an appropriate subgraph and serving strategy is chosen. Four subgraph types are defined and for each case a different serving strategy is developed.

Mbaraga et al. (1999) developed a heuristic for the distance constrained TCVRP (CDTVRP). It is based on the heuristic developed by Labbé et al.

(1991). The heuristic is a simple modification, and at every step, in addition to checking for capacity constraints, the heuristic also checks for distance constraint violations. It is also a linear time heuristic.

2.2.3 Exact Solution Methods

Exact solution methods for TVRPs include branch–and–bound methods, column generation schemes, and Integer Programming formulations. Currently, exact methods exist only for TCVRPs, DTVRPs and CDTVRPs. Developing exact solution methods for other variants of TCVRPs is an open problem.

Chandran and Raghavan (2008) developed two integer programs for solving TCVRPs. The first one builds off the fact that, once the nodes that are part of a vehicle route are determined, nodes will be served in the order of their DFS index, a similar IP was also developed by Busch (1990). The second formulation uses the fact that there is only one path between a node and any other node in the depot, and that every node has a unique parent node. Some valid inequalities, to further expedite the convergence of the IP, were also proposed.

Labbé et al. (1991) developed a branch–and–bound enumeration scheme for solving the TCVRP. The scheme proceeds in a breadth first order. If the upper bound solution value $\bar{z}$ equals the lower bound value $\underline{z}$, then scheme terminates as it has found the optimal solution. Otherwise, a new vehicle route is created for every node with capacity between $\frac{Cap}{2}$ and Cap. These nodes

are defined to be I^1. Now, at an arbitrary iteration, r, an unassigned node, k, farthest from the depot is chosen. Branches are created from r for every vehicle route and a new route with only k in it, provided capacity constraints are satisfied. A lower bound is computed for every potential node p. If $\underline{z}^p \geq \bar{z}$, then p is not created. Otherwise, $I^p = I^r \cup \{k\}$. Thus, this procedure implicitly considers every node assignment to every vehicle, thereby exhausting all possibilities.

Mbaraga et al. (1999) used a similar branch–and–bound scheme for the DTVRP and the CDTVRP. The only modification is that, for the DTVRP, at every iteration, a branch is created only if the distance constraints are not violated. These are trivially checked as the total distance traveled by each vehicle is twice the length of tree with only the nodes in that vehicle considered. Similarly, for the CDTVRP, capacity and distance constraint violations are checked before branching.

Mbaraga et al. (1999) also define a set covering–based formulation for the CDTVRP, which is solved using column generation. The master problem of the column generation scheme solves the CDTVRP for a subset of variables, these variables are then parsed to the subproblem. The subproblem of the CDTVRP is a capacitated and distance constrained shortest path problem, whose arc costs are defined such that the shortest path will generate a column with the most negative reduced cost, which will in turn be parsed to the master problem. The tree arcs and costs are modified to form an acyclic graph for every vehicle. The constrained shortest path (SP) is then solved on these

acyclic graphs using the algorithm proposed by Desrosiers et al. (1995). The SP algorithm has a pseudo–polynomial running time.

For smaller demands, bin–packing bounds are sharper and have smaller search trees, and thus, branch–and–bound is more efficient than column generation. However, the column generation scheme is much more robust in handling heterogeneous vehicles.

2.2.4 Vehicle Scheduling Problem on Trees

In TSP with time windows (TSPTW), the total tour length has to be minimized subject to time window constraints. A version of the TSPTW, where each node has a handling time associated with it and the total waiting time is minimized, is called the Traveling Repairman Problem (TRP) (Tsitsiklis, 1992). TSPTW is also referred to as the Vehicle Scheduling Problem (VSP) when the objective minimizes the total completion time (Karuno et al., 1997). Many papers in the literature deal with TSPTWs, VSPs, and TRPs on trees, paths and lines. Usually, the left hand side time windows, a_i, are called release time, and the service times, s_i, are called handling times. TSPTWs, TRPs, and VSPs on trees are NP–hard (Tsitsiklis, 1992). However, for some special line graphs, polynomial algorithms do exist (Psaraftis et al., 1990).

Psaraftis et al. (1990) examined a straight line version of the VSP, where the objective was to minimize the maximum lateness. They considered the case where only release times were specified. For this case, a simple polynomial algorithm was presented. If 1 is the first node, and n is the last node on the

line, the vehicle simply travels to node n, waits there for a time that is equal to the maximum period it might have to wait at the node with the latest release time, and then begins service from node n to 1 without waiting at any node. The computational complexity of this algorithm is $O(n)$. In the path version of the problem it is not required to return back to node 1. Additionally, they defined properties of special line networks called shorelines (as these structures are usually found in ports where cargoes from ships have to be transported), and developed VSP algorithms for those shorelines. Yu and Liu (2010) solve the VSP on a line, but with both release and handling times. They provide a 3/2 approximation for the tour case, and a 5/3 approximation for the path case. When time windows are specified, it is easier to solve tours than paths, this is because in tours, there is more flexibility in deciding waiting strategies.

Karuno et al. (1997) examined the VSP on trees for nodes having both release and handling times. This problem is NP-hard. They provide a polynomial exact solution for this VSP when there is a depth first routing constraint. Also, using this depth first routing strategy, they developed a 2–approximate algorithm for the non–depth first routing VSP on a tree. Bhattacharya et al. (2008) provide a 5/3 approximation for the same problem. Finally, Karuno and Nagamochi (2003, 2001) solved the VSP with release and handling times on trees, but with multiple vehicles.

Chapter 3

TVRP with Backhaul Customers

3.1 Problem Definition

The Capacitated TVRP with Backhauls (TCVRPB) is concerned with finding optimal cost minimizing vehicle routes that serve linehaul and backhaul customers that are located on a tree network. The linehaul customers have a higher priority of service and are serviced before the backhaul customers.

The problem is defined as follows. Given a tree network $T = (N_D, A)$ with customer nodes $i \in N$ and arcs $(i, j) \in A$ between the nodes with non–negative arc costs c_{ij}; a set of linehaul nodes $i_L \in L$ and backhaul nodes $i_B \in B$ such that $|L| + |B| = |N|$ and $L \cap B = \emptyset$; demand at each node d_i; and a set of vehicles $k \in K$ with capacity Cap – find a collection of routes starting and ending at the depot, such that

(i) the customer's demand/supply at a node is satisfied by exactly one vehicle,

(ii) vehicle capacity restrictions are adhered to,

(iii) linehaul nodes are serviced before backhaul nodes,

The TCVRPB is discussed in the following two chapters. This chapter describes exact and heuristic methods to solve this problem, while their imple-

mentation is discussed in Chapter 4. In Section 3.2, some characteristics and properties of TVRPBs at optimality are noted. A procedure to calculate the lower bound is also described in this section. Section 3.3 contains an Integer Programming method to solve TCVRPBs to optimality. This IP is formulated using the properties and observations discussed in Section 3.2. An improved IP formulation, requiring fewer number of constraints and variables, based on a network reformulation procedure is the focus of Section 3.4. A cluster first–route second heuristic procedure is described in Section 3.6. The heuristic first decides the nodes that have to be served by each vehicle, then devises an optimal serving strategy for the vehicle. Optimal serving strategy involves solving the Traveling Salesman Problem on Trees with Backhauls, the algorithm for which is delineated in Section 3.5. The second IP formulation is faster than the first and is used in the computational analysis that is presented in Chapter 4, which also contains the details of the heuristic performance. Based on the numerical results, a few further improvements based on a new heuristic are implemented in Section 4.3. The Section 4.4 concludes the discussion with a summary of the main findings and scope for future work.

3.2 Preliminaries

3.2.1 Observations

Observation 3.1. *In a TCVRPB minimizing cost does not necessarily minimize the number of vehicles that are used in the optimal solution.*

Consider the example in Figure 3.1. The tree is rooted at the depot

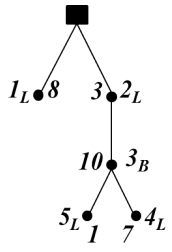


Figure 3.1: Minimizing cost does not minimize vehicles

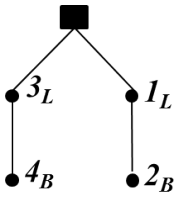


Figure 3.2: Difference between TVRPs and TVRPBs

and the capacity of each vehicle is 10. Assume that all edge costs are 1. The number next to the node is the demand/supply at that node. If only two vehicles are given, in the optimal solution vehicle 1, V_1, will serve nodes $1_L, 3_B$, and 5_L; and V_2 will serve nodes 2_L and 4_L. The optimal cost will be 14. However, if 3 vehicles are given, V_1 will serve 1_L; V_2 will serve 2_L; and V_3 will serve $3_B, 4_L$ and 5_L. The optimal cost will be 12.

This is true for the uncapacitated case too. In Figure 3.2, when only one vehicle is available, its optimal route will be $D-1_L-3_L-4_B-2_B-D$ with a cost of 10 (assuming unit arc costs). However, if two vehicles are available, each branch will be served separately, resulting in a cost of 8.

It is also interesting to note that the uncapacitated TVRPB is not

as trivial to solve as the TVRP. In the TVRP, when capacities don't exist, a single vehicle serving the nodes in the DFS order will yield an optimal solution. However, this is not the case with the TVRPB. Moreover, unlike TVRPs, a single vehicle can traverse an arc more than two times. Consider the simple tree graph shown in Figure 3.2. Assume that a single vehicle is used and all arc costs are 1. Here, in order to minimize distance, the vehicle will serve the nodes in the order $D - 1_L - 3_L - 4_B - 2_B - D$, yielding a cost of 10. Observe that arc $(D, 1_L)$ is traversed four times in the optimal solution. Also, observe that a DFS visitation of linehaul and then backhaul nodes will not yield a feasible solution.

Observation 3.2. *For a node i, assume that $j \in c(i)$. If arc $(j,i)[(i,j)]$ is traversed in the backhaul[linehaul] tour of vehicle k, then arc $(i, P_i)[(P_i, i)]$ must be traversed in the backhaul[linehaul] tour.*

If a vehicle traverses directed arc (j, i) in the backhaul tour, then every node it visits after that must be on the backhaul tour. Consequently, if it decides to go *up* the tree, it should do so on the backhaul tour. One can make the same argument for the linehaul case.

Observation 3.3. *A vehicle k will enter a subtree S_i at most twice, once on the lineaul tour and once on the backhaul tour. Moreover, the edge $\{P_i, i\}$ will be traversed at most four times by that vehicle k.*

We deal with the offline case where all demands/supply are known beforehand. Once a vehicle k enters a sub–tree S_i on a linehaul tour, it will

serve all the nodes it has chosen to serve in S_i. This is because it will never be cheaper for that vehicle to enter the sub–tree more than once on the linehaul tour. The same argument can be made about the backhaul tour. Therefore, the vehicle will traverse the edge $\{P_i, i\}$ at most four times.

3.2.2 Properties

Lemma 3.1. *If a vehicle k serves a linehaul node i_L, it will do so on its way from the depot, and never on its way back to the depot. In other words, if a vehicle k decides to serve a linehaul node i_L, it will do so immediately after visiting node P_{i_L}, and never after visiting any child node $c(i_L)$.*

Proof. Here, we assume that the demand at the nodes are known *a priori*. Therefore, the decision of which vehicle serves which nodes is made offline, and not enroute. It is known that a vehicle enters a sub–tree S_i only if it is going to serve nodes in that sub–tree. If the vehicle decides to serve a linehaul node i and some other linehaul nodes in S_i, then there is no reason for the vehicle to first serve nodes 'below' i before serving i. Thus, the property always holds. □

For example, in Figure 3.3, a vehicle k decides to serve the nodes in sub–tree S_i. The linehaul nodes are i and j. It will always be cheaper to serve node i before serving node j, because then the vehicle has to traverse every arc only twice. Whereas, in the converse case, the vehicle will have to traverse edge $\{i, j\}$ four times.

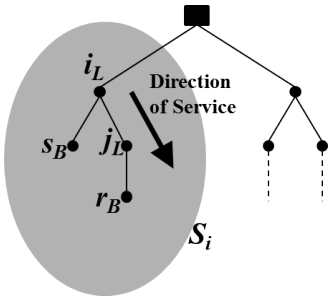


Figure 3.3: Figure describing Lemma 3.1

Lemma 3.2. *Consider a vehicle k that serves a subset of nodes R_k, such that $R_k \subseteq N$. Let the covering sub–tree of R_k be denoted by CS_{R_k}. Let the set of nodes in the unique Path from the Depot to a node $i \in N$ be denoted by PfD_i. Also, let the first backhaul node served by vehicle k be i_{1_B}. Then, the vehicle k enters (from the depot) every node in $PfD_{i_{1_B}}$ on the linehaul route and exits (towards the depot) every node in $PfD_{i_{1_B}}$ on the backhaul route.*

Proof. In the sub–tree $S_{i_{1_B}}$, i_{1_B} will be the only backhaul node, because if there was a backhaul node *below* this node, then that node will be the node that is served first in the backhaul route of the vehicle. The vehicle k enters the sub–tree $S_{i_{1_B}}$ on the linehaul route, but exits the sub–tree on the backhaul route. Once the vehicle exits this sub–tree, it never enters is again. Therefore, the edge $\{P_{i_{1_B}}, i_{1_B}\}$ will be traversed exactly twice – once on the linehaul route and once on the backhaul route. Consequently, i_{1_B} is visited exactly twice.

Now, consider the sub–tree formed by the parent on the node i_{1_B}, $S_{P_{i_{1_B}}}$. Vehicle k enters this sub–tree first on the linehaul tour. As the objective seeks

to minimize the cost, before entering sub–tree $S_{i_{1_B}}$, the vehicle will service all other linehaul nodes (in increasing order of index). After entering (on the linehaul route) and exiting (on the backhaul route) sub–tree $S_{i_{1_B}}$, the vehicle will service all the remaining backhaul nodes in increasing order of index before exiting the sub–tree $S_{P_{i_{1_B}}}$ on the backhaul route. The edge between $P_{i_{1_B}}$ and its parent is traversed exactly twice. The lemma is proved by considering the sub–tree of the parent of the last sub–tree served, and by noting that a every sub–tree of $PfD_{i_{1_B}}$ is visited exactly once. □

Two corollaries follow from Lemma 3.1 and 3.2. They are stated as follows.

Corollary 3.1. *For a vehicle k, the sub–tree S_i of a node $i \notin PfD_{i_{1_B}}$ is visited only once if it contains only linehaul or only backhaul nodes, and is visited twice otherwise.*

Corollary 3.2. *For a vehicle k, if $i_{|L|_L}$ is the last linehaul node served then the sub–tree $S_{i_{|L|_L}}$ contains no linehaul nodes and, given that the nodes are indexed in DFS order, the first backhaul node served is the highest indexed node in this sub–tree, if this sub–tree is not empty. Secondly, the sub–tree formed by the first serviced backhaul node contains no other backhaul nodes.*

3.2.3 Lower Bound

Consider a sub–tree S_i rooted at node i, the minimum number of vehicles required to serve the linehaul nodes, say, V_i^L is given by:

$$V_i^L = \left\lceil \frac{\sum_{j \in \{L \cap S_i\}} d_j}{Cap} \right\rceil$$

Similarly, the minimum number of vehicles required to serve the backhaul nodes is:

$$V_i^B = \left\lceil \frac{\sum_{j \in \{B \cap S_i\}} d_j}{Cap} \right\rceil$$

Therefore, the minimum number of vehicles required to serve sub–tree S_i is given by:

$$V_i = \max\left\{V_i^L, V_i^B\right\}$$

Proposition 3.1. *A lower bound, $\underline{z}$, on the optimal objective value, z^*, to the TCVRPB is given by:*

$$\underline{z} = 2 \sum_{i \in N} V_i c_{\{P_i, i\}} \tag{3.1}$$

Proof. As described above, the minimum number of vehicles requires to serve S_i is V_i. Therefore, at least V_i vehicles will enter the sub–tree S_i to serve its nodes. This implies that at least V_i vehicles will traverse the edge $\{P_i, i\}$. These vehicles will traverse the arc $\{P_i, i\}$ once on their way from the depot to the sub–tree, and once on their way back (after service) from the sub–tree to the depot. The result follows. □

3.3 Integer Programming Formulation

In the IP formulation, there are two major categories of constraints that need to be considered. (A) Service constraints ensure that all the demand and supplies are serviced without violating the vehicle capacity constraints. (B) Movement constraints which have to enforce movement–related constraints like, (a) all vehicles should start at the depot, (b) linehaul nodes should be serviced before backhaul nodes, (c) the vehicle moves sequentially along the tree and does not make arbitrary jumps to other nodes etc. This formulation is called BIP_1.

Formulation: **TCVRPB(1)**

$$x^l_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ uses arc } (i,j) \text{ during the linehaul tour} \\ 0 & \text{otherwise} \end{cases}$$

$$x^b_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ uses arc } (i,j) \text{ during the during backhaul tour} \\ 0 & \text{otherwise} \end{cases}$$

$$y^l_{ik} = \begin{cases} 1 & \text{if linehaul node } i \text{ is served by vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

$$y^b_{ik} = \begin{cases} 1 & \text{if backhaul node } i \text{ is served by vehicle } k \\ 0 & \text{otherwise} \end{cases}$$

Minimize:
$$z^*_{BIP_1} = \sum_i \sum_j \sum_k c_{ij}(x^l_{ijk} + x^b_{ijk}) \tag{3.2}$$

Subject to:
$$\sum_{i \in L} y^l_{ik} d_i \leq Cap \qquad \forall k \in K \tag{3.3}$$
$$\sum_{i \in B} y^b_{ik} d_i \leq Cap \qquad \forall k \in K \tag{3.4}$$
$$\sum_{k \in K} y^l_{ik} = 1 \qquad \forall i \in L \quad \sum_{k \in K} y^b_{ik} = 1 \quad \forall i \in B \tag{3.5}$$
$$y^l_{ik} \leq \sum_{j \in \Gamma(i)} x^l_{ijk} \qquad \forall i \in L \;\; \forall k \in K \tag{3.6}$$
$$y^b_{ik} \leq \sum_{j \in \Gamma(i)} x^b_{ijk} \qquad \forall i \in B \;\; \forall k \in K \tag{3.7}$$
$$y^l_{ik} \leq x^l_{P_i ik} \qquad \forall k \in K \;\; \forall i \in L \tag{3.8}$$
$$x^l_{ijk} + x^b_{ijk} = x^l_{jik} + x^b_{jik} \qquad \forall k \in K \;\; \forall \{i,j\} \in E \tag{3.9}$$
$$x^l_{P_i ik} + x^b_{iP_i k} \geq x^l_{ijk} + x^b_{ijk} \qquad \forall i \in N \;\; \forall k \in K \;\; j \in c(i) \tag{3.10}$$
$$x^l_{P_i ik} \geq x^l_{ijk} \qquad \forall i \in N \;\; \forall k \in K \;\; j \in c(i) \tag{3.11}$$
$$x^b_{iP_i k} \geq x^b_{jik} \qquad \forall i \in N \;\; \forall k \in K \;\; j \in c(i) \tag{3.12}$$
$$x^l_{iP_i k} + x^b_{iP_i k} \geq x^l_{jik} \qquad \forall i \in N \;\; \forall k \in K \;\; j \in c(i) \tag{3.13}$$
$$x^l_{P_i ik} + x^b_{P_i ik} \geq x^b_{ijk} \qquad \forall i \in N \;\; \forall k \in K \;\; j \in c(i) \tag{3.14}$$
$$x^l_{ijk} \in \{0,1\} \quad x^b_{ijk} \in \{0,1\} \qquad \forall k \in K \;\; \forall \{i,j\} \in E \tag{3.15}$$
$$y^l_{ik} \in \{0,1\} \quad y^b_{ik} \in \{0,1\} \qquad \forall k \in K \;\; \forall i \in N \tag{3.16}$$

The objective function minimizes the total cost of operating the vehicles. Constraints 3.3– 3.5 are service constraints which ensure that the total demand serviced by a vehicle does not exceed its capacity in the linehaul or backhaul tour, and that not more than one vehicle services a node.

Constraints from 3.6 to 3.14 are movement constraints. Constraint 3.6 enforces that if a vehicle k serves the linehaul node i, then at least one of the arcs emanating from i must be a linehaul arc. Constraint 3.7 enforces this for backhaul nodes. 3.6 and 3.7 are important for vehicle propagation. That is, once a vehicle services a node, it has to move on from that node to the next one. Constraint 3.8 mathematically imposes the first lemma. If a linehaul node i is served by vehicle k, then it should do so from the parent node of i; that is, a linehaul node is served by a vehicle on its way from the depot.

Constraint 3.9 which ensures that an edge is traversed at most four times, and also that if an arc (i, j) is on the linehaul tour only, then the arc (j, i) will either be on the backhaul tour, or the linehaul tour, but not both. Also, if an arc (i, j) is on the backhaul and linehaul tours, then the arc (j, i) will be on the backhaul and linehaul tours.

Constraint 3.10 enforces that for a node i, assume that $j \in c(i)$, if directed arc (i, j) is traversed in the linehaul and backhaul tour of vehicle k, then directed arc (P_i, i) will be traversed in the linehaul tour, and directed arc (i, P_i) will be traversed in the backhaul tour of vehicle k.

Here, it is imposed that a linehaul node is always *serviced* from above, and also that once a node has been served, the vehicle should either visit its parent or one of its child nodes. Constraints 3.11– 3.14 do not allow a backhaul node to be serviced before all the linehaul nodes have been serviced. Also, from Lemma 3.1, it can be said that linehaul nodes will always be served first. Therefore, all movement related constraints are satisfied in the formulation.

3.4 An Improved Integer Programming Formulation

In TCVRPBs, once a vehicle begins its backhaul tour, it cannot serve any linehaul nodes. Secondly, a vehicle cannot serve intermediate backhaul (linehaul) nodes which are adjacent to the linehaul (backhaul) nodes that are being served during the linehaul (backhaul) tour. Using these two facts, a network modification of the tree is proposed in this section. An IP formulation based on this new network is described.

3.4.1 Network Transformation

Given a tree network, $T = (N_D, A)$, the main idea of this transformation procedure involves constructing two trees – a linehaul tree and a backhaul tree – and connecting these two trees in a manner that forces a vehicle to first serve nodes in the linehaul tree before serving the nodes in the backhaul tree.

Before describing the procedure in detail, two concepts are noted and defined:

(i) *Fork Nodes*: A fork node is defined to be a node in the tree $T = (N_D, E)$ which has more than two incident edges. More formally, a node i is said to be a fork node if it has more than one child, that is $|c(i)| > 1$. If FN is the set of fork nodes in a tree T, then $FN = \{i : |c(i)| > 1 \forall i \in N\}$.

(ii) *Short–Circuiting*: The term is used loosely here and is just used to give the concept a name. Consider three nodes i, j, and l in the tree T. Let $P_j = i$ and $c(j) = \{l\}$. Note that l is the only child of j. Then, if a vehicle

is interested in serving only nodes i and l, it is possible to construct a covering sub–tree for that vehicle such that node j is not part of the sub–tree, and arcs (i, j) and (j, l) are replaced with arc (i, l) with cost c_{il} . Although j is still a part of the route from i to l, its presence in the tree is inconsequential as it does not affect the solution or serving strategy of the vehicle. If the vehicle decides to go from i to l in the covering sub–tree, then the actual route the vehicle is taking is $i - j - l$. Thus, node j has been short–circuited. Note that its only possible to short–circuit those nodes which have exactly one child. Define SC_i for every node $i \in N$ as follows:

$$SC_i = \begin{cases} 1 & \text{if } |c(i)| = 1 \\ 0 & \text{otherwise} \end{cases}$$

SC_i defines the *short–circuitability* of a node i, deeming that a node can be considered for a short–circuit if its value is 1.

Figure 3.4(a) represents the tree T with one fork node l. Now, the minimum covering sub–tree, CS_R with $R = \{i, m, n\}$ is represented in Figure 3.4(b). Node j can be short–circuited as it neither belongs to R, nor does its presence affect the route of the vehicle. Although l is not a part of R, it cannot be dismissed here, and neither can l be merged with i as that will not reflect the true cost of any route chosen by the vehicle. The covering sub–tree with $R = \{i, m\}$ is represented in Figure 3.4(c). Both j and l can be short–circuited here as they need not be explicitly considered in the routing decision for the vehicle.

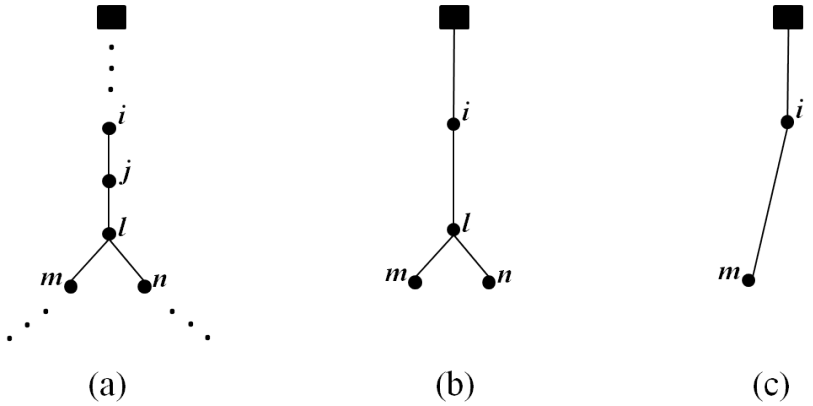


Figure 3.4: An Example Describing Fork Nodes and Short–Circuiting; In (b), $R = \{i, m, n\}$; In (c), $R = \{i, m\}$

Linehaul and Backhaul Tree

Using the concepts defined previously, a linehaul tree, $T_L = (N_L, A_L)$, and a backhaul tree, $T_B = (N_B, A_B)$, are constructed from the given tree T. For convenience, assume that each tree has a copy of the depot. Also, $N_{L_D} = N_L \cup \mathit{depot}$ and $N_{B_D} = N_B \cup \mathit{depot}$. Again, a covering sub–tree, of the tree T, with a subset of nodes $R \subseteq N$ is denoted by CS_R. T_L and T_B are constructed as follows. The trees $CS_L = (N_{CS_L}, A_{CS_L})$ and $CS_B = (N_{CS_B}, A_{CS_B})$ are constructed. For CS_L, the FN and $SC_i \forall i \in N_{CS_L}$ is calculated. If a node i's SC_i value is 1, and it is neither a fork node or a linehaul node, it is short circuited and the corresponding edge costs are modified. The resulting tree is the linehaul tree, T_L. The backhaul tree, T_B, is constructed similarly. The demands of linehaul (backhaul) nodes replicated in the backhaul (linehaul) tree are set to zero. All other demands are kept the same.

Crossover Arcs

Once the linehaul and backhaul trees have been constructed, they need to be connected such that a vehicle services backhaul nodes after servicing the linehaul nodes. The arcs which are used to make this connection are denoted as *Crossover Arcs*. Note that since linehaul nodes cannot be serviced once the backhaul route is initiated, the crossover arcs are always directed from the linehaul tree to the backhaul tree. Consider a linehaul node $i_L \in L$. If, in the original tree T, a node adjacent to this node is a backhaul node, say $j_B \in B$, then a crossover arc is constructed from the node that corresponds to the node i_L in T_L to the node that corresponds to the node j_B in T_B. The cost of this arc is cost of traversing arc (i_L, j_B) in T. A node is adjacent to another node if its either its parent or child. That is, a node j is adjacent to a node i if $j \in \{P_i \cup c(i)\}$. The set of crossover arcs are denoted as A_{CR}.

The network resulting from the transformation is denoted as $G_B = (N_{G_B}, A_{G_B})$, where $N_{G_B} = N_L \cup N_B \cup \mathit{depot}$ and $A_{G_B} = A_L \cup A_B \cup A_{CR}$.

Algorithm 3.1 formally defines the steps in the network transformation procedure. Lines 2 and 3 describe the construction of the linehaul and backhaul tree respectively using the function `TreeConstruct()`, the details of which are presented in Lines 11–27. The crossover arcs are then added to this network according to Lines 3–7. Note here that, for the ease of representation, the node numbers in T_L and T_B are the same as that in T, the numbering of nodes for the computer implementation is detailed later in the section. All nodes in T_L are identified using the subscript L, same for T_B. Also, for a node $i \in N_L$,

Algorithm 3.1 Network Transformation

Input: $T = (N_D, A) \quad c_{ij} \forall (i,j) \in A \quad d_i \forall i \in N$
Output: $G_B = (N_{G_B}, A_{G_B}) \quad c_{ij} \forall (i,j) \in A_{G_B} \quad d_i \forall i \in N_{G_B}$

1: $R \leftarrow L$; Call `TreeConstruct()`
2: $R \leftarrow B$; Call `TreeConstruct()`
3: **for** $\forall (i,j) \in A$ **do**
4: **if** $i \in L$ $j \in B$ **then**
5: $A_{CR} \leftarrow A_{CR} \cup (i,j) : i \in N_L, j \in N_B$ with cost c_{ij}
6: **end if**
7: **end for**
8: $N_{G_B} \leftarrow N_L \cup N_B \cup depot$
9: $A_{G_B} \leftarrow A_L \cup A_B \cup A_{CR}$
10: $G_B = (N_{G_B}, A_{G_B})$
11: **Function** `TreeConstruct()`
12: Build Covering Sub–tree $CS_R = (N_{CS_R}, A_{CS_R})$
13: **for** $i = 1 \rightarrow |N_{CS_R}|$ **do**
14: **if** $i \notin FN$ **then**
15: **if** $i \notin R$ $SC_i = 1$ **then**
16: delete i
17: $c(P_i) \leftarrow c(i)$
18: $P_{c(i)} \leftarrow P_i$
19: $c_{\{P_i, c(i)\}} \leftarrow c_{\{P_i, i\}} + c_{\{i, c(i)\}}$
20: **end if**
21: **else**
22: **if** $i \notin R$ **then**
23: $d_i \leftarrow 0$
24: **end if**
25: **end if**
26: **end for**
27: **end Function**

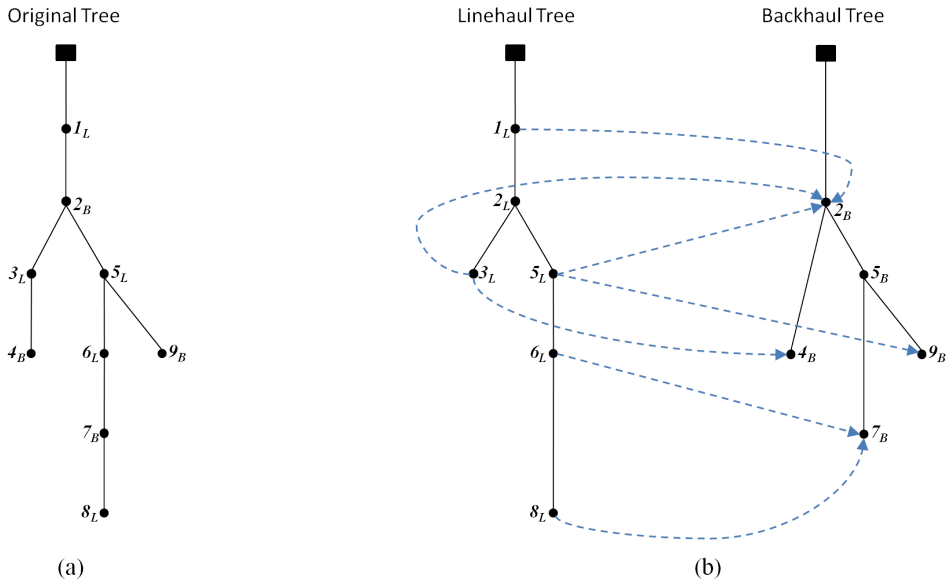


Figure 3.5: An Example Describing the Network Transformation

its immediate ancestor in N_L is defined to be its parent P_i and its immediate descendants in N_L are defined to be its children $c(i)$. The same holds for a node $j \in N_B$.

Figure 3.5 is an example of the modification procedure described above. Figure 3.5 is the original tree T. The modified network G_B is shown in Figure 3.5(b). In T, the linehaul nodes have a subscript L and the backhaul nodes have a subscript B. The linehaul and backhaul trees are constructed as shown in Figure 3.5(b). Consider backhaul node 7, this node satisfies the conditions for short–circuiting and is deleted from the linehaul tree. The arcs (6,7) and (7,8) are replaced with the arc (6,8) in the linehaul tree. Similarly, node 6 is deleted from the backhaul tree. On the other hand, node 2 is a fork node, and

needs to be replicated in both trees. The node is represented as 2_L in T_L and as 2_B in T_B. Since node 2 is a backhaul node, the demand for node 2_L is set to zero. This is just stated for completeness, as the solution method described later does not require the demand for the replicated node. Crossover arcs are constructed from a linehaul node to its adjacent backhaul node. For example, linehaul nodes 1, 3 and 5 are adjacent to backhaul node 2, hence, crossover arcs $(1_L, 2_B), (3_L, 2_B)$ and $(5_L, 2_B)$ are constructed. Lastly, in G_B the depot nodes can be combined, they are duplicated here for visual representation.

Only fork nodes whose sub–tree contains both linehaul and backhaul nodes are duplicated in G_B. Let these fork nodes be represented by FN_D. Then the total number of nodes in G_B, $|N_{G_B}| = |L| + |B| + |FN_D| + 1$. The extra node represents the depot. Thus, when $FN_D = \emptyset$, the number of nodes in G_B and T are equal.

The total number of arcs in the linehaul tree, $|A_L| = 2 \times (|L| + |FN_D|)$. Similarly, $|A_B| = 2 \times (|B| + |FN_D|)$. The number of crossover arcs, $|A_{CR}|$ is equal to the total number of adjacent backhaul nodes to every linehaul node. Recall that **ADJ** is the node–node adjacency matrix of the tree T. So, $|A_{CR}| = \sum_{i \in L} \sum_{j \in B \cap \{P_i \cup c(i)\}} \mathbf{ADJ}_{ij}$. Thus, $|A_{G_B}| = 2 \times (|L| + |FN_D|) + 2 \times (|B| + |FN_D|) + \sum_{i \in L} \sum_{j \in B \cap \{P_i \cup c(i)\}} \mathbf{ADJ}_{ij}$.

Finally, it is easy to verify that T and G_B are equivalent networks. This can be done by observing that the unique *path* between any two nodes, say i and j, in T and G_B are the same. For the case where i and j are either both linehaul or backhaul nodes, this is obviously true. For the case where i

is a linehaul node and j is a backhaul node, or vice versa, and are adjacent to each other, i and j are directly connected by a crossover arc. For the case where they are not adjacent, it can be verified that there exists only one path between the two nodes (which belong to different trees) in which nodes are not repeated. For example, in Figure 3.5, if one wishes to travel from node 3 to node 9, then on T the path taken would be $3-2-5-9$. However, on G_B, one can choose $3_L-2_B-5_B-9_B$ or $3_L-2_L-5_L-9_B$. But in both cases, the nodes visited are the same. Note that one will never choose $3_L - 4_B - 2_B - 5_B - 9_B$ as that is not a path and contains the cycle $3 - 4 - 3$ in T.

In order to avoid any confusion due to node–replication, the nodes are numbered as follows. The nodes in the linehaul tree are DFS ordered with the depot marked as node 1. That is, $N_L = \{2, \ldots, |N_L|+1\}$. Similarly, the nodes in the backhaul tree are numbered as $N_B = \{|N_L| + 2, |N_L| + 3, \ldots, |N_B| + |N_L| + 1\}$.

3.4.2 The IP Formulation

Using properties discussed in Section 3.2 and the network transformation described in the previous sub–section, an IP formulation is developed here. The IP formulation is a vehicle–indexed formulation and is defined for the network G_B. This formulation is called BIP_2. The variables are defined as follows:

Variables

$$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ travels on arc } (i,j) \in A_{G_B} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{if vehicle } k \text{ serves node } i \in N_{G_B} \\ 0 & \text{otherwise} \end{cases}$$

Note here, that unlike the earlier formulation, the varaibles are not separately defined for the linehaul and backhaul tour. This reduces the total number of variables as will be shown later in this section.

The first set of constraints are the service constraints which are similar to the Constraints 3.3 – 3.5 defined in the previous section. The first two constraints state that the total linehaul (backhaul) demand serviced by a vehicle cannot exceed its capacity. The constraints are as follows:

$$\sum_{i \in N_L : d_i > 0} y_{ik} d_i \leq Cap \qquad \forall k \in K$$

$$\sum_{i \in N_B : d_i > 0} y_{ik} d_i \leq Cap \qquad \forall k \in K$$

The next two constraints impose that every node is serviced by exactly one vehicle. The can be written as follows:

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N_L : d_i > 0$$

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N_B : d_i > 0$$

Next, movement–related constraints need to be defined. Recall Lemma 3.1. It stated that a vehicle k always serves a linehaul node on its way from the depot. This implies that a linehaul node can be served only if the arc leading up to that node from its parent is traversed. That is,

$$y_{ik} \leq x_{P_i ik} \qquad \forall i \in N_L : d_i > 0, \forall k \in K$$

In the linehaul tour, it is essential to ensure that every vehicle begins its tour at the depot. A constraint that ensures that a vehicle never moves from a node to its child unless the arc from that node's parent to that node has been traversed will impose this requirement. So, if a vehicle k serves node $i \in N_L : d_i > 0$, this constraint enforces that all the arcs on the path that leads from the depot to this node i are traversed by k.

$$x_{P_i ik} \geq x_{ijk} \qquad \forall i \in N_L : d_i > 0, \forall j \in c(i), \forall k \in K$$

Two similar constraints also need to be defined for the backhaul tour to ensure that every vehicle ends its tour at the depot. Firstly, a vehicle must proceed towards the depot after it serves a backhaul node. So, if a vehicle serves $i \in N_B : d_i > 0$, then it must travel on arc $(i, P_i) \in A_L$. This is expressed as:

$$y_{ik} \leq x_{iP_i k} \qquad \forall i \in N_B : d_i > 0, \forall k \in K$$

Secondly, every vehicle's backhaul tour must end at the depot. Whenever a vehicle travels on the arc connecting the child to a node, it must also travel on the arc connecting that node and its parent in the backhaul tree. So, if a vehicle k serves node $i \in N_B : d_i > 0$, this constraint enforces that all the arcs on the path that leads from that node i to the depot are traversed by k.

$$x_{iP_i k} \geq x_{jik} \qquad \forall i \in N_B : d_i > 0, \forall j \in c(i), \forall k \in K$$

None of the constraints defined above preserve tour continuity. That is, say a vehicle k serves node i before serving node j. Two cases exist.

(i) If i and j are nodes of the same type. Then if the nodes are linehaul (backhaul), the constraints above will just enforce that all the arcs on the path leading up to (away from) those nodes from (towards) the depot are traversed by vehicle k.

(ii) If i is linehaul and j is backhaul. Then, the constraints will enforce that all the arcs on the path from the depot to i are traversed by k and all the arcs on the path from j to the depot are traversed by k.

However, the constraints do not enforce any traversal between these two intermediate nodes. That is, once i has been served, the vehicle must travel to j before it goes back to the depot. In this sense, tour continuity is not preserved. This can be enforced by describing a flow conservation constraint for every node.

$$\sum_{j \in \{P_i \cup c(i)\}} x_{ijk} - \sum_{j \in \{P_i \cup c(i)\}} x_{jik} = 0 \qquad \forall i \in \{N_{G_B} \setminus depot\}, \forall k \in K$$

The indegree of every node i must be equal to its outdegree. The Figure 3.6 explains how this constraint helps preserve tour continuity.

Consider a vehicle k that serves the backhaul nodes $3_B, 5_B$, and 6_B, as shown in Figure 3.6(a). Let k enter the backhaul tree through the crossover arc at 3_B (shown by a dashed squiggly arc). Without the flow conservation equation, k will traverse the arcs $(3_B, 2_B)$ after serving node (3_B, $5_B, 4_B$) and $6_B, 4_B$) after serving nodes 4_B and 5_B, respectively. Finally, it will traverse the arcs from 4_B to the depot.

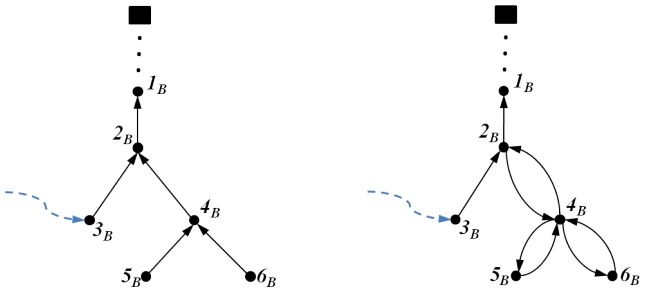


Figure 3.6: Flow Conservation Constraint Imposes Tour Continuity; (a) Without Flow Conservation and (b) With Flow Conservation

To satisfy flow conservation at node 2_B, the vehicle must traverse arc $(2_B, 4_B)$, as traversal on any other arc (to satisfy flow conservation for node 2_B) will violate other constraints. Similarly, to maintain flow conservation at nodes $4_B, 5_B$, and 6_B, k must traverse the arcs $(4_B, 5_B)$ and $(4_B, 6_B)$.

It is interesting to note that the property described in Lemma 3.2 and Lemma 3.3 are also satisfied. Finally, if a vehicle serves only linehaul nodes, then due to flow conservation, the vehicle will return back to the depot *along* the linehaul tree. Similarly for purely backhaul serving vehicles too.

For completeness, the IP formulation is given in full below:

Formulation: **TCVRPB(2)**

Minimize:
$$z^*_{BIP_2} = \sum_i \sum_j \sum_k c_{ij} x_{ijk} \tag{3.17}$$

Subject to:
$$\sum_{i \in N_L : d_i > 0} y_{ik} d_i \leq Cap \qquad \forall k \in K \tag{3.18}$$
$$\sum_{i \in N_B : d_i > 0} y_{ik} d_i \leq Cap \qquad \forall k \in K \tag{3.19}$$
$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N_L : d_i > 0 \tag{3.20}$$
$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N_B : d_i > 0 \tag{3.21}$$
$$y_{ik} \leq x_{P_i ik} \qquad \forall i \in N_L : d_i > 0, \forall k \in K \tag{3.22}$$
$$x_{P_i ik} \geq x_{ijk} \qquad \forall i \in N_L : d_i > 0, \forall j \in c(i), \forall k \in K \tag{3.23}$$
$$y_{ik} \leq x_{iP_i k} \qquad \forall i \in N_B : d_i > 0, \forall k \in K \tag{3.24}$$
$$x_{iP_i k} \geq x_{jik} \qquad \forall i \in N_B : d_i > 0, \forall j \in c(i), \forall k \in K \tag{3.25}$$
$$\sum_{j \in \{P_i \cup c(i)\}} x_{ijk} = \sum_{j \in \{P_i \cup c(i)\}} x_{jik} \qquad \forall i \in \{N_{G_B} \setminus depot\}, \forall k \in K \tag{3.26}$$
$$x_{ijk} \in \{0, 1\} \qquad \forall (i,j) \in A_{G_B}, \forall k \in K \tag{3.27}$$
$$y_{ik} \in \{0, 1\} \qquad \forall k \in K, \forall i \in \{N_{G_B} \setminus depot\} : d_i > 0 \tag{3.28}$$

Adaptability of the Formulation

The formulation BIP_2 can be adapted as follows to solve the following variants of the TCVRPB:

(i) Minimize Number of Vehicles Used: Firstly, the total number of available vehicles is kept free. That is, $|K| = |N| - 1$. Then, the objective function

is changed to:

$$z^*_{BIP_2} = \sum_{j \in c(depot)} \sum_{k \in K} x_{depotjk}$$

Due to the assumption that, in T, the depot has only one child, simply calculating the total number of vehicles traversing the arc from the depot to its child will record the total number of vehicles used by the formulation. In G_B, the depot either has two children (one for the linehaul tree and one for the backhaul tree), or there are two copies of the depot (depending on if the depots have been merges after the transformation operation). In either case, the above equation will minimize the total number of vehicles used.

(ii) Vehicle Fleet is Heterogeneous: In this case, vehicles of different capacities are employed. Modifying the parameter Cap to $Cap_k \forall k \in K$ and employing this modified parameter in the formulation will account for heterogeneous fleets.

(iii) Vehicle Dependent Arc Costs: The arc costs are no longer $c_{ij} \forall (i,j) \in A_{G_B}$ but $c_{ijk} \forall (i,j) \in A_{G_B}, \forall k \in K$. Changing the arc costs to c_{ijk} wherever they appear in BIP_2 will account for these vehicle dependent arc costs.

Number of Variables and Constraints

A node variable, y_{ik}, is defined for every node $i \in L \cup B$. The total number of y variables in BIP_2 is $|K||L \cup B|$, which is the same as the number

of y variables used in BIP_1. An arc variable, x_{ijk} is defined for every arc $(i,j) \in A_{G_B}$. The total number of x variables in BIP_2 is $|K|\big\{2(|L|+|FN_D|)+2(|B|+|FN_D|)+\sum_{i\in L}\sum_{j\in B\cap\{P_i\cup c(i)\}}\mathbf{ADJ}_{ij}\big\}$. The total number of x variables in BIP_1 is $4|K|(|L|+|B|)$. So, unless $4|FN_D|+\sum_{i\in L}\sum_{j\in B\cap\{P_i\cup c(i)\}}\mathbf{ADJ}_{ij} \geq 2(|L|+|B|)$, the second formulation always has fewer number of variables. Since that inequality is never true, one can safely assert that:

$$\#\text{of variables}_{BIP_1} > \#\text{of variables}_{BIP_2}$$

Although the total number of constraints in both the formulations are $O(|N||K|)$, it can be observed that the number of constraints in BIP_1 is $(2+|L|)|K|+7|N||K|$, and the number of constraints in BIP_2 is $2|K|+3|N||K|+3|FN_D||K|+|N|+|FN_D|$. So,

$$\#\text{of constraints}_{BIP_1} > \#\text{of constraints}_{BIP_2}$$

Strength of the Formulation

Consider the LP–relaxation of the second formulation with value $z^{LP}_{BIP_2}$. Due to constraints 3.24 and 3.25, the value of the x variables in the backhaul tour will be:

$$x_{iP_ik} = \max_{j\in S_i}\big\{y_{jk}\big\} \quad \forall i \in N_B : d_i > 0, \forall k \in K$$

Now, consider the LP–relaxation of first formulation with value $z^{LP}_{BIP_1}$. Due to constraints 3.7, 3.9, and 3.12, the value of the x variables in the backhaul tour will satisfy:

$$\frac{y_{ik}}{|c(i)\cup P_i|} \leq x^b_{iP_ik} \leq \max_{j\in S_i}\big\{y_{jk}\big\} \quad \forall i \in B, \forall k \in K$$

The x variables in the linehaul tour for both formulations will remain the same as they are both governed by the same set of constraints. From the above two relations, it can be claimed that BIP_2 is a stronger formulation than BIP_1 and that:

$$z^{LP}_{BIP_2} \geq z^{LP}_{BIP_1}$$

3.4.3 Valid Inequalities

In this section, valid inequalities that are added to further strengthen the IP formulation are described.

As noted by Barnhart et al. (1998), a compact formulation sometimes has a symmetric structure and a weak LP–relaxation value, to overcome this, some symmetry breaking inequalities should be added. The formulation BIP_2 has a symmetric structure as, in the optimal solution, a set of nodes served by any two vehicles can be interchanged. Due to this, the solution search space increases. To overcome this symmetry, two sets of inequalities are added.

Recall that the nodes in the linehaul tree are DFS ordered with the depot marked as node 1. That is, $N_L = \{2, \ldots, |N_L| + 1\}$. The first symmetry breaking inequality imposes the condition that a linehaul node can only be served by a vehicle whose index is less than the node index. Then, node 2 can be served by vehicle 1, node 3 by vehicles 1 or 2, node 4 by vehicles 1,2, or 3 and so on. This constraint is expressed as:

$$\sum_{k=1}^{i-1} y_{ik} = 1 \qquad \forall i \in \{N_L : d_i > 0 \wedge 2 \leq i \leq |K| + 1\} \tag{3.29}$$

The second symmetry breaking constraint imposes that the lowest indexed linehaul node serviced by a vehicle increases with the vehicle index. This constraint is expressed as:

$$y_{ik} \leq \sum_{j=2}^{i-1} y_{j,k-1} \quad \forall i \in \{N_L : d_i > 0 \wedge 3 \leq i \leq |K|+1\}, 2 \leq k \leq |K| \quad (3.30)$$

Next, knapsack–like constraints are added to the formulation. This constraint states that if the sum of the demands of two linehaul (backhaul) nodes exceeds the vehicle capacity, then only one of these linehaul (backhaul) nodes can be serviced by that vehicle.

$$y_{ik} + y_{jk} \leq Cap \qquad \forall i, j \in N_L : d_i + d_j > Cap, \forall k \in K \qquad (3.31)$$

$$y_{ik} + y_{jk} \leq Cap \qquad \forall i, j \in N_B : d_i + d_j > Cap, \forall k \in K \qquad (3.32)$$

In sub–section 3.2.3, V_i^L was described to be the minimum number of vehicles required to serve the sub–tree $S_i : i \in N_L$. Therefore, in the linehaul tree, the arc (P_i, i) will be traversed at least V_i^L times, or at least V_i^L vehicles will use arc (P_i, i).

$$\sum_{k \in K} x_{P_i ik} \geq V_i^L \qquad \forall i \in N_L \qquad (3.33)$$

Similarly, this constraint can be defined for the arc (i, P_i) in the backhaul tree. Note the reversal of direction as all vehicles moving on the backhaul tree will traverse the arc towards the depot, but not necessarily the arc *from* the depot.

$$\sum_{k \in K} x_{iP_i k} \geq V_i^B \qquad \forall i \in N_B \qquad (3.34)$$

3.5 The Traveling Salesman Problem on Trees with Backhauls

The heuristic for the TCVRPB that is described here first computes the subset of nodes that will be served by each vehicle and then computes the optimal route for each vehicle. Traditionally, the optimal route for each vehicle is found by solving the traveling salesman problem. In tree networks with backhauls, the traveling salesman problem can be solved to optimality in polynomial time. This section describes an algorithm for the Traveling Salesman Problem on Trees with Backhauls (TTSPB).

Definition 3.1. Given $T = (N_D, A)$ and arc costs c_{ij} with linehaul nodes $L \subseteq N$ and backhaul nodes $B \subseteq N$, and $|L| + |B| = |N|$ and $L \cap B = \emptyset$. The Traveling Salesman Problem on Trees with Backhauls is concerned with finding a cost minimizing tour that begins and ends at the depot such that all nodes are visited exactly once.

The given tree network is modified according to Algorithm 3.1. The rest of this section deals with this modified network $G_B = (N_{G_B}, A_{G_B})$. Of course, since a single tour covering all nodes is sought, there are no demands associated with the nodes.

The set of arcs in the (shortest) path from i to j is denoted by $i \to j$ and the set of nodes in the unique Path from the Depot to a node $i \in N_{G_B}$ be denoted by PfD_i. Let $i_{|L|_L}$ be the last linehaul node and i_{1_B} be the first backhaul node served in a TTSPB tour. The set of linehaul nodes that can

possibly be served last during a TTSPB tour is $|L|_L$ and the set of backhaul nodes that can possibly be served first is 1_B. It is easy to verify that the last served node in the linehaul tour must be a tail node of a crossover arc. Correspondingly, the first served node in a backhaul tour must be a head node of a crossover arc. That is, $|L|_L = \{i \in N_L : i \in L \wedge \{\exists j \in N_B : (i,j) \in A_{CR}\}\}$ and $1_B = \{j \in N_B : j \in B \wedge \{\exists i \in N_L : (i,j) \in A_{CR}\}\}$.

Lemma 3.3. *In a TTSPB tour, all arcs will be traversed except for the arcs in the sets* $\{depot \rightarrow i_{1_B}\}$, $\{i_{|L|_L} \rightarrow depot\}$, *and* $\{A_{CR} \setminus (i_{|L|_L}, i_{1_B})\}$

Proof. The above statement can be decomposed into the following three parts:

(i) The arcs in the sets $\{depot \rightarrow i_{1_B}\}$ and $\{i_{|L|_L} \rightarrow depot\}$ will not be traversed.

If one considers the edge set E_{G_B} instead of A_{G_B}, the statement can be interpreted as: The edges in the path from the depot to the node $i_{|L|_L}$ and the edges in the path from the node i_{1_B} to the depot will be traversed exactly once. This statement for the network G_B is analogous to the statement made in Lemma 3.2 for the network T and can be proved in the same way.

(ii) The only crossover arc traversed will be the arc $(i_{|L|_L}, i_{1_B})$.

Once a crossover arc is traversed, one cannot return to the linehaul tree. Therefore, only one crossover arc will be traversed and this will be the crossover arc leading out from the last linehaul node served.

(iii) All other arcs will be traversed.

Once again, this implies that in the edge set E_{G_B}, all other edges will be traversed exactly twice. This holds true because an edge will be traversed once leading up to a node for service and once away from the node after service.

□

The number of nodes in the sets $|L|_L$ and 1_B can further be reduced by noting that a linehaul node, i_L, can be the last served linehaul node only if its sub–tree, S_{i_L}, contains no linehaul nodes which are in the set $|L|_L$. Define the new set of linehaul nodes that can possibly be served last as:

$$|L|_{L_{new}} = \left\{ i \in |L|_L : \{|L|_L \cap S_i\} = \emptyset \right\}$$

Similarly, the new set of backhaul nodes that can be served first is defined as:

$$1_{B_{new}} = \left\{ j \in 1_B : \left\{ \exists i \in |L|_{L_{new}} : (i,j) \in A_{CR} \right\} \wedge \left\{ \nexists (p,q) \in A_{CR} : p \in |L|_{L_{new}}, q \in \{1_B \cap S_j\} \right\} \right\}.$$

It is known that only one crossover arc will be traversed in any TTSPB tour. The candidate crossover arcs that will be traversed in the optimal tour will have their tail nodes in the set $|L|_{L_{new}}$ and their head nodes in the set $1_{B_{new}}$. If the candidate crossover arcs were denoted by the set CR_{cand}, then

$$CR_{cand} = \{(i,j) \in A_{CR} : i \in |L|_{L_{new}}, j \in 1_{B_{new}}\} \tag{3.35}$$

Of course, in hindsight, there is no need to compute the sets $1_{B_{new}}$ and $|L|_{L_{new}}$ as CR_{cand} can be computed directly. But, these have been included to better explain the idea.

3.5.1 Optimal TTSPB Tour Cost

Unlike the regular TSP, where finding optimal cost will also return the optimal node service order, in TTSPBs calculating optimal tour cost is easier than finding than the optimal node service order.

Given the last linehaul node and the first backhaul node served in the optimal TTSPB tour, one can easily compute the tour cost using the Lemma 3.3. The candidate first linehaul and last backhaul nodes are known to be the tail and head nodes of the arcs in the set CR_{cand}. The optimal TTSPB tour cost, $TTSPB_{cost}$, is computed as follows. For every arc $(i, j) \in CR_{cand}$, first, the cost for traversing all the arcs in G_B is computed. From this cost, the path cost for traversing from i to the depot and the shortest path cost for traversing from the depot to j are subtracted. This is done because the arcs in these paths will not be a part of the TTSPB tour. Then, from this cost, all the crossover arc costs, except the one currently being used (c_{ij}) are subtracted. At the end of this procedure, $|CR_{cand}|$ possible TTSPB costs are calculated. The one with the least value is the optimal TTSPB cost, $TTSPB_{cost}$. The procedure is formalized in Algorithm 3.2.

The computational complexity of Algorithm 3.2 is $O(|CR_{cand}|)$ and $|CR_{cand}| \leq \frac{|N_{G_B}|^2}{4}$. Therefore, the algorithm is $O(|N|^2)$ as $|N_{G_B}|$ is $O(|N|)$.

3.5.2 Node Service Order

In the optimal TTSPB tour, only one crossover arc is used. The arcs that will be used in the optimal tour are also known from Lemma 3.3. In the

Algorithm 3.2 Optimal Tour Cost

Input: $G_B = (N_{G_B}, A_{G_B})$ $\quad c_{ij} \forall (i,j) \in A_{G_B}, \quad CR_{cand}$
Output: $TTSPB_{cost}$
1: $TTSPB_{cost} \leftarrow \sum_{(i,j) \in A_{G_B}} c_{ij}$
2: **for** $iter = 1 \rightarrow |CR_{cand}|$ **do**
3: $\quad X \leftarrow \sum_{(i,j) \in A_{G_B}} c_{ij} - L_i - L_j - \sum_{(i,j) \in A_{CR} \setminus (i,j)_{iter}} c_{ij}$
$\quad \{\%\ \%\ (i,j)_{iter}$ `is the` $iter^{th}$ `element of set` $CR_{cand}\ \%\ \%\}$
4: $\quad$ **if** $X < TTSPB_{cost}$ **then**
5: $\quad\quad TTSPB_{cost} \leftarrow X$, $i_{|L|_L} \leftarrow i_{iter}$, $j_{1_B} \leftarrow j_{iter}$
6: $\quad$ **end if**
7: **end for**

optimal tour, the nodes should be serviced such that no additional arcs are used.

Recall that, in the optimal tour, $i_{|L|_L}$ is the last linehaul and j_{1_B} is the first backhaul node. The set PfD_i contains the nodes in the path from the depot to i in increasing order of index. Observe that the sub–tree of a node is the union of the sub–tree of the children of that node and that node itself. Now, the nodes will be serviced in the following order.

Linehaul Nodes The tour starts at the depot. It serves the first node in $PfD_{i_{|L|_L}}$. The tour then serves all the nodes in its children's sub–trees (selecting a sub–tree arbitrarily, but traversing each sub–tree in increasing order of index), except for the sub–tree generated by the child which is a part of $PfD_{i_{|L|_L}}$. After this, the tour serves the second node in the set $PfD_{i_{|L|_L}}$, here too it visits all its children's sub–trees, except for the child that is in $PfD_{i_{|L|_L}}$. The tour continues in this fashion till it reaches the node $i_{|L|_L}$, at which point it serves the last linehaul node's

entire sub–tree. At the end of this procedure, all the linehaul nodes will be served.

Backhaul Nodes The backhaul tour starts at j_{1_B}. The first backhaul node and its entire sub–tree is served. Then, the tour serves the parent of j_{1_B} and all *unserved* sub–trees of the parent's other children. The tour continues to move up the tree in this fashion till it finally reaches the depot, at which point all backhaul nodes would have been served.

More formally, while moving down the tree along $PfD_{i_{|L|_L}}$, after serving a node $i \in PfD_{i_{|L|_L}}$, the tour will serve the following sub–set of nodes in increasing order of index:

$$T_i = \bigcup_{j \in \left\{ c(i) \setminus PfD_{i_{|L|_L}} \right\}} S_j \quad \forall i \in PfD_{i_{|L|_L}}$$

While moving up the tree along $PfD_{i_{1_B}}$, after serving a node $i \in PfD_{i_{1_B}}$, the tour will serve the following sub–set of nodes in increasing order of index:

$$T_i = \bigcup_{j \in \left\{ c(i) \setminus PfD_{i_{1_B}} \right\}} S_j \quad \forall i \in PfD_{i_{1_B}}$$

To further elucidate the procedure, consider the example in Figure 3.7. Here, 2_L is the last linehaul node served and 4_B is the first backhaul node served. The tour first serves node 1, after which it serves all the nodes in the

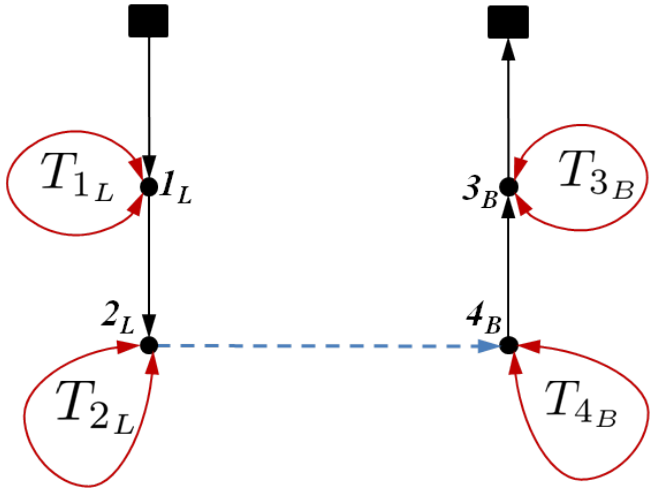


Figure 3.7: Order of Nodes Served in TTSPB

set denoted by T_{1_L}, note how this set does not contain the sub–tree generated by node 1's child (2) which is on PfD_{2_L}. The tour then serves node 2 and all the nodes in node 2's subtree. At this stage, all the linehaul nodes have been served. The tour begins its backhaul service by serving node 4 and all the nodes in node 4's sub–tree. The tour then serves node 3 and the set T_{3_B} which does not contain any nodes generated by node 3's child (4) which is on PfD_{4_B}. The tour finally ends at the depot.

3.5.3 Illustrative Example

An Example is shown in Figure 3.8. The original tree is shown in Figure 3.8(a). The nodes 2, 4, 7, and 9 are the backhaul nodes (they have been subscripted with B). The arc costs are give next to each edge. It is assumed that the cost of traversing arc (i, j) and (j, i) are equal.

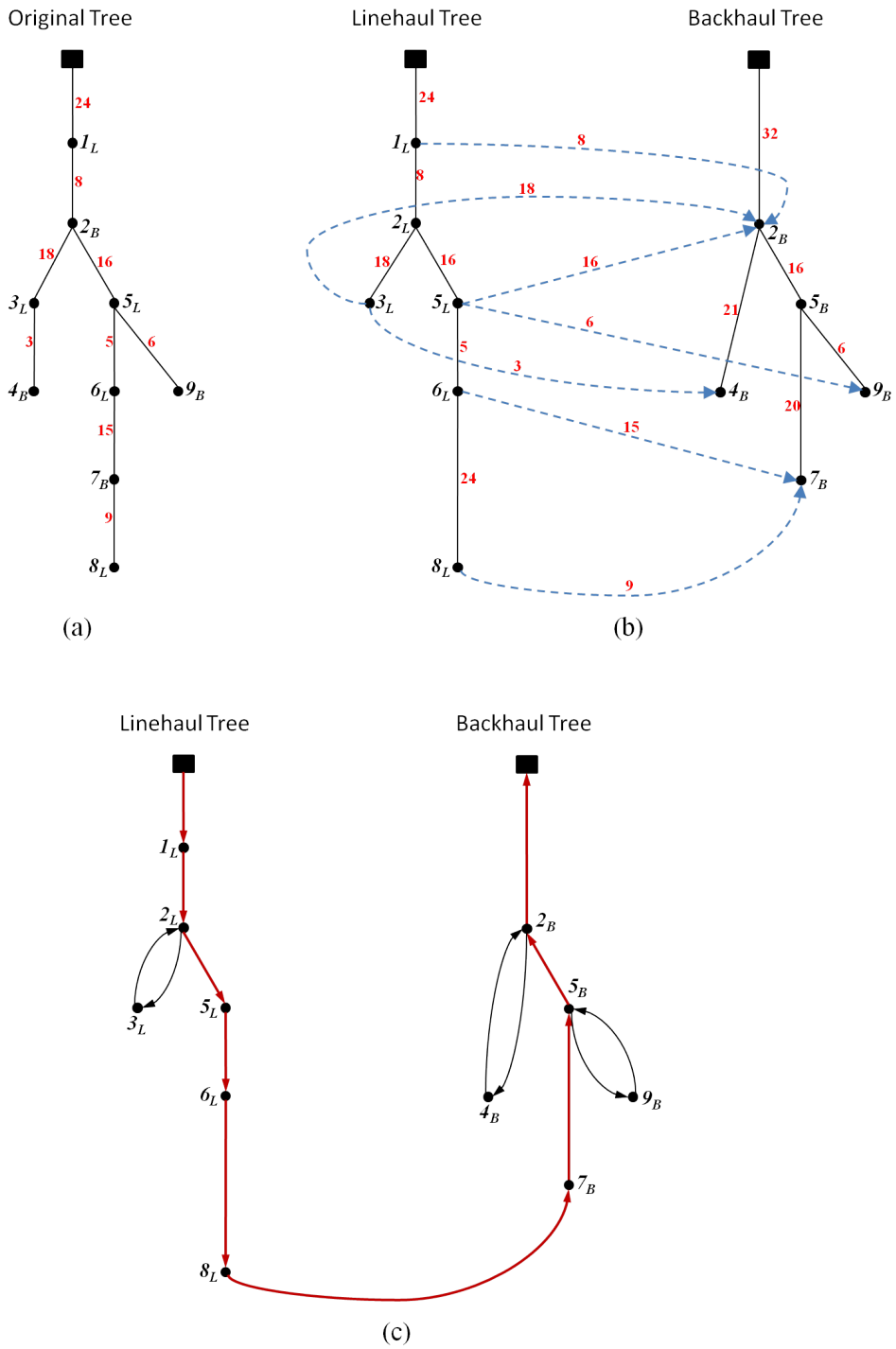


Figure 3.8: An Illustrative Example for the TTSPB; (a) Shows the Original Tree; (b) Shows the Transformed Network; (c) Shows the TTSPB Tour

The network G_B after network transformation is shown in Figure 3.8(b). In this network, each node in the linehaul tree is sub–scripted with L, similarly, a sub–script B for nodes in the backhaul tree. The modified arc costs are also shown. Now, from the discussion in the previous sub–section, the set of candidate crossover arcs, as defined in Equation 3.35 are:

$$CR_{cand} = \{(3_L, 4_B), (8_L, 7_B)\}$$

Using Algorithm 3.2, the two tour costs are 298 and 244 respectively. Therefore, the optimal TTSPB tour is the one where the last linehaul node served is node 8_L and the first backhaul node served is node 7_B. This optimal cost and the its tour is shown in Figure 3.8(c).

Now, the edges which will be used only one are shown in red in Figure 3.8(c). The two sets of nodes PfD_{8_L} and PfD_{7_B} will be traversed in the order as stated in the previous section. The tour starts at the linehaul depot (depots have been replicated here for ease of representation) and first serves node 1_L. Since T_{1_L} is empty, it then moves to node 2_L. $T_{2_L} = \{3_L\}$, and the tour serves this node. It then moves to node 5_L. Proceeding this way, the tour reaches node 5_B, where $T_{5_B} = \{9_B\}$. Therefore, the tour serves node 9_B before it moves to node 2_B. This continues till the backhaul depot is reached. At termination, the order in which the nodes are served is $depot - 1_L - 2_L - 3_L - 5_L - 6_L - 8_L - 7_B - 5_B - 9_B - 2_B - 4_B - depot$.

3.6 Heuristic

The heuristic procedure designed for the TCVRPB is a cluster–first route–second method. There are two main steps to the algorithm. In the first step, the algorithm finds the nodes that will be served by each vehicle. In the second step, the route taken by the vehicle is computed. The clustering stage of the procedure is as follows. This step deals with the tree $T = (N, A)$ and not with the network G_B. Pick a non–grandparent node i farthest from the depot. A non–grandparent node i is a node whose children $c(i)$ have no children. In other words, the children $c(i)$ are all leaf nodes. The nodes $i \cup c(i)$ are packed into a two–dimensional bin. A two-dimensional bin here implies a bin that is partitioned into two compartments, both with capacity equal to the vehicle capacity. One partition holds the linehaul nodes, while the other holds the backhaul nodes. The nodes $i \cup c(i)$ are then removed from the tree and are replaced by the bins into which these nodes have been packed. Obviously, these bins are now the children of the parent of node i, that is the bins form the set $c(P_i)$. This procedure continues till all the nodes have been accommodated into the bins. Note that, at an intermediate stage, the set $i \cup c(i)$ that has to be packed might contain the original nodes as well as bins that have been packed previously.

The first–fit decreasing bin–packing procedure is used here. At every step, the set $i \cup c(i)$ is arranged in decreasing order of demand. So, when the set that has to be packed contains a bin, the demand of that bin is the sum of the total linehaul and backhaul nodes it has served. Two such bins can

be packed into a single bin if the sum of the total demand served in linehaul compartment of both the bins is less than the vehicle capacity and the sum of the demand total demand served in the backhaul compartment of both the bins is less than the vehicle capacity. At the end of this procedure, nodes in each bin are the nodes that will be served by each vehicle. Also note that at the end of the procedure there will be no bin in which both the compartments are half empty.

The second step of the procedure just involves computing the TTSPB for the set of nodes that every vehicle served. This is done using Algorithm 3.2.

Algorithm 3.3 Heuristic for TCVRPB

Input: $T = (N, A)$ $d_i \forall i \in N$ $c_{ij} \forall (i,j) \in A$
Output: Upper Bound TCVRPB Cost, $\bar{z}$

1: $LIST \leftarrow N$
2: **while** $LIST \neq \emptyset$ **do**
3: $NGP \leftarrow \{j \in LIST : (c(j) \neq \emptyset) \wedge (c(p) = \emptyset \forall p \in c(j))\}$
4: $i \leftarrow \max_{j \in NGP}\{L_j\}$
5: $(Vehicle, Bin) \leftarrow$ `2D-BinPack`$(i \cup c(i))$
6: $LIST \leftarrow LIST \setminus \{(i \cup c(i)) \wedge (LIST)\}$
7: $N \leftarrow N \cup Bin \setminus \{i \cup c(i)\}$
8: $P_j \leftarrow P_i \forall j \in Bin$
9: **end while**
10: $\bar{z} \leftarrow 0$
11: **for** $iter = 1 \rightarrow |Vehicle|$ **do**
12: $R \leftarrow Vehicle_{iter}$; Call `TreeConstruct()` from Algorithm 3.1
13: Call Network Transformation Algorithm 3.1
14: Calculate $TTSPB_{cost}$ using Algorithm 3.2
15: $\bar{z} \leftarrow \bar{z} + TTSPB_{cost}$
16: **end for**

The steps of this procedure are described in Algorithm 3.3. NGP

is the set of non–grandparent nodes which are eligible candidates to begin packing. The function $(Vehicle, Bin) = \texttt{2D-BinPack}(X)$, takes in the set X that has to be packed, and returns the bins, Bin, they have been packed into and the set $Vehicle$ containing the set of the elements in each bin. In the first stage of the heuristic, at every step a a non–grandparent node is chosen and it is packed. A simple packing heuristic has a complexity $O(|N|^2)$ and the total number of steps in the first stage are at most $|N|$. Therefore, the first stage has a complexity of $O(|N|^3)$. However, if one were to use a faster packing heuristic, like the one proposed by Knödel (1981), the complexity will reduce to $O(|N|^2 \log |N|)$. The complexity of the second stage is same and the TTSPB, which is $O(|N|)$. Therefore, the total complexity of the heuristic is $O(|N|^3 + |N|)$ and can be potentially reduced to $O(|N|^2 \log |N| + |N|)$

The procedure is further elucidated by means of an example shown in Figure 3.9. The original tree is shown is Figure 3.9(a), the linehaul and backhaul nodes have been sub–scripted L and B respectively. The arc costs are shown on the edges (in red), and the demand at every node is shown in a green box next to the node. The vehicle capacity is assumed to be 300. The various iterations of the first stage of the heuristic are shown in Figure 3.9(b) – (g).

The non–grandparent node 7 is farthest from the tree and is chosen along with its children for packing. The nodes 7 and 8 are deleted from the tree and replaced with the partitioned bin as shown in Figure 3.9(b). The total linehaul and backhaul demand served by the bin is shown below the bin. Now,

node 6 is the non–grandparent node that is farthest from the tree. Therefore, 6 and its children are packed. The resulting bin is a child of node 5 and contains nodes 6, 7, and 8. The procedure continues in this fashion. Consider the Figure 3.9(e), in this iteration node 2 and its three children bins have to be packed. Note that, firstly, no two bins can be combined together because no two bins have both partitions half empty. Next, node 2 will be accommodated in the bin containing 3 and 4 because its the one with the highest total demand served. The heuristic uses four vehicles to serve the nodes. After computing the TTSPB cost for each vehicle, the total heuristic upper bound value is 416. The optimal solution value is 410.

The next chapter describes the computational results obtained by implementing the exact and heuristic techniques discussed in this chapter.

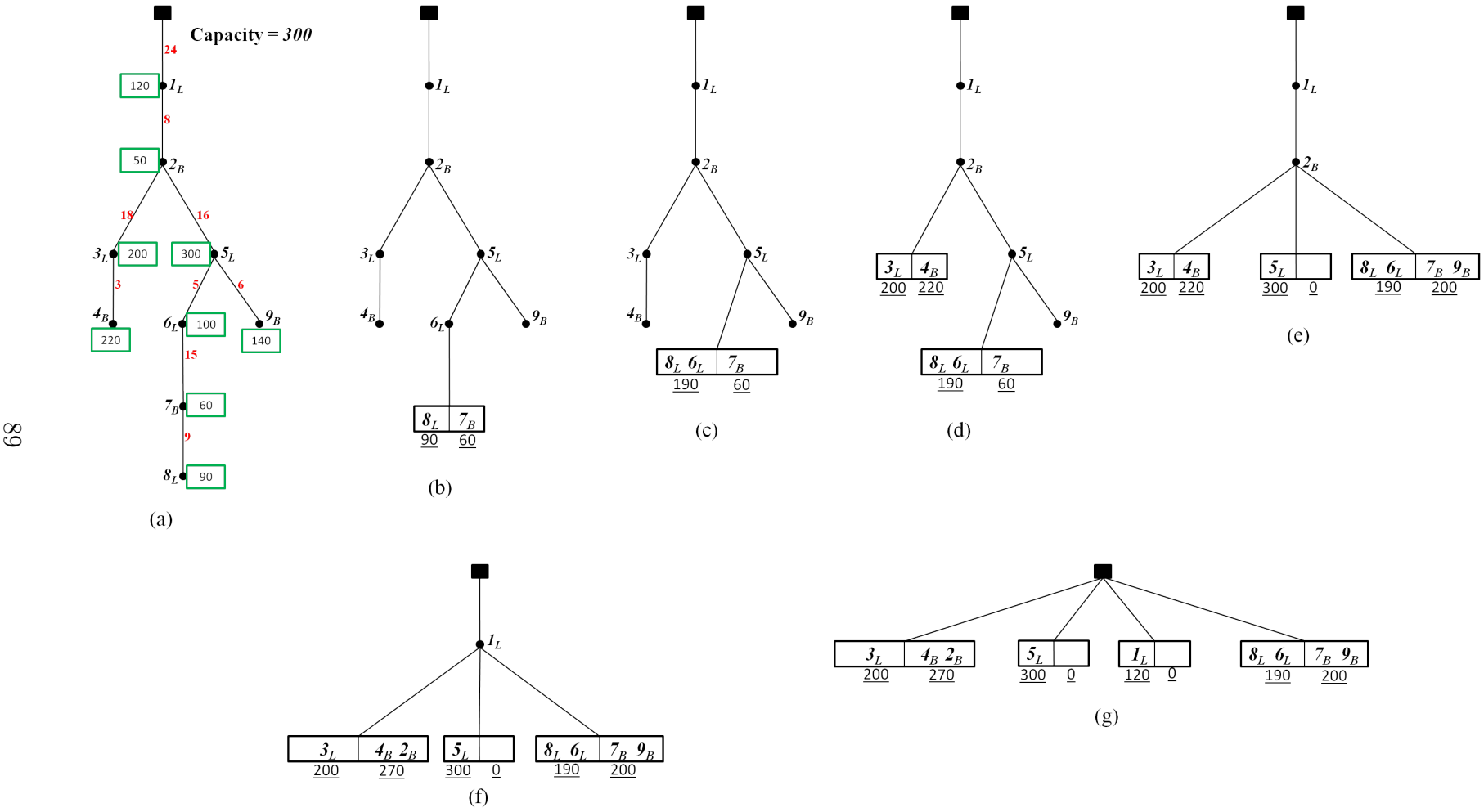


Figure 3.9: An Illustrative Example for the TCVRPB

Chapter 4

Computational Results and Further Improvements to the TCVRPB

The heuristic and the formulation were tested on tree networks of various sizes. This chapter describes the experimental setup and the computational results of this implementation. Based on the computational results, a few further improvements, along with a new heuristic, are also discussed and implemented. The computational experiments were carried out only on the second IP formulation, BIP_2. It was analytically shown that BIP_2 had fewer variables and constraints compared to BIP_1 and also produced a stronger LP–relaxation compared to BIP_1.

4.1 Test Instances

The test networks were generated using a procedure similar to the one described by Labbé et al. (1991). Every node in the tree had between 1 to 5 children. Without loss of generality, the depot was forced to have a degree of one. 20, 40, 80 and 140–node networks were generated.

4.1.1 Parameter Generation

Arc Costs The arc costs were uniformly distributed in [1, 100].

Demand Each test network was tested for five demand profiles uniformly distributed between [1 10], [1 30], [1 100], [20 20] and [20 80].

Backhaul Nodes The total number of backhaul nodes, $|B|$, were assumed to be between $\frac{1}{3}^{rd}$ to $\frac{2}{3}^{rd}$ of the total number of nodes, $|N|$, in the tree. This was done in two steps. First, a random number, x, between $[\frac{1}{3}$ $\frac{2}{3}]$ was generated and the total number of backhaul nodes was set to $|B| = \lfloor (x|N|) + 0.5 \rfloor$. Second, backhaul nodes were designated as follows. $|N|$ numbers in `Unif(0,1)` were generated. A two–dimensional array containing these numbers and the node numbers from 1 to $|N|$ was created. The nodes corresponding to the first $|B|$ smallest `Unif(0,1)` numbers generated were selected to be the backhaul nodes.

Capacity The vehicle capacity was set to 100.

Number of Vehicles The formulation BIP_2 requires the total number of vehicles $|K|$. This was set to be equal to the total number of vehicles that were used by the heuristic. The IP formulation does not impose that all the vehicles must be used. It works within the given $|K|$ and uses the optimal number of vehicles that will minimize the total cost. The number of vehicles are required because of the vehicle–index nature of the formulation. Providing the IP with the number of vehicles used

by the heuristic also helps better compare the optimal solution value to the heuristic value.

4.2 Computational Results

The second IP formulation, along with the valid inequalities, was solved using CPLEX on a PC with a 32–bit architecture, 2GB RAM, and 2.93 GHz processor. The iteration limit for branch and bound was set to 100000, the time limit was set to 3600 seconds, and the optimality criterion for the MIP (relative gap between the best found integer solution value and best found LP value) was set to 0.0001. For every demand profile and for every node size, 10 instances were solved.

The order of branching the variables was set as follows. First, a variable that represents the total number of vehicles is defined as follows:

$$v = \sum_{j \in c(depot)} \sum_{k \in K} x_{depotjk} \tag{4.1}$$

The variable v was branched first. Next, the branching rule for the variables y_{ik} was set such that these variables are branched in increasing order of the vehicle index. That is, the variable y_{ik_1} was branched before the variable y_{ik_2} for all $k_1 < k_2$.

Finally, CPLEX was warm–started using the heuristic solution. That is, the heuristic solution was fed into CPLEX as the first feasible heuristic solution. So that this initial solution does not violate the symmetry constraints, the nodes served by each vehicle in the heuristic were ordered such that the

lowest indexed node served by a vehicle increased with vehicle index and that a linehaul node is only served by a vehicle whose index is less than the node index. A summary of the computational results is presented in Table 4.1. The summary contains the following information:

Solved The number of instances (out of 10) that were solved to optimality within 3600 seconds.

$|K|$ The number of vehicles input to the IP formulation. It is the same as the number of vehicles used by the heuristic.

TimeOPT The time taken by CPLEX to solve the MIP formulation to optimality (reported only for solved instances).

TimeUB The time taken to solve the heuristic. It is the sum of times required to solve the clustering step and the TTSPB step of the heuristic.

Gap This is the gap between the last LP value and the last heuristic value found by CPLEX. It is zero for all solved instances and positive for unsolved instances. This metric helps determine how close CPLEX was to finding the optimal solution at termination.

LP/IP The ratio of the LP–relaxation value to the optimal solution value. This value helps understand the quality of the LP–relaxation of the formulation. The closer the LP–relaxation value is to the optimal value, the better it is for the branch and cut procedure.

UB/IP The ratio of the heuristic solution value and the optimal solution value helps understand the quality of the heuristic.

LB/UB The ratio of the lower bound and the heuristic solution value.

LP/LastUB The ratio of the LP–relaxation value to the last best heuristic solution value found for CPLEX. Reported only for unsolved instances, as for solved instances this value is equal to the LP/IP value.

Network	Demand	Solved[α]	$\|K\|$	TimeOPT secs	TimeUB secs	LP/IP	UB/IP	LB/UB	LP/LastUB	LP/LB	Nodes	Gap
20–node	[1 10]	10	2	0.353	0.166	0.985	1.034	0.847	–	1.130	0	0
	[1 30]	10	4	0.676	0.184	0.974	1.088	0.830	–	1.080	1.4	0
	[1 100]	10	9	2.020	0.163	0.971	1.040	0.762	–	1.027	69.1	0
	[20 20]	10	4	0.737	0.115	0.979	1.048	0.914	–	1.027	3.6	0
	[20 80]	10	8	3.622	0.143	0.942	1.056	0.745	–	1.230	156.7	0
40–node	[1 10]	10	2	3.714	3.600	0.995	1.050	0.808	–	1.175	0	0
	[1 30]	10	5	11.530	3.418	0.985	1.037	0.867	–	1.096	271.3	0
	[1 100]	10	16	96.73	1.302	0.948	1.029	0.728	–	1.273	3695	0
	[20 20]	10	8	12.780	1.269	0.985	1.056	0.874	–	1.071	232.5	0
	[20 80]	10	18	44.118	1.191	0.950	1.055	0.676	–	1.336	1285.3	0
80–node	[1 10]	10	4	24.944	7.173	0.984	1.013	0.790	–	1.22	8	0
	[1 30]	10	10	285.950	8.529	0.967	1.092	0.675	–	1.195	8249	0
	[1 100]	9	25	422.065	8.431	0.947	1.051	0.634	0.96	1.420	6031	0.012
	[20 20]	10	15	257.668	7.134	0.919	1.090	0.698	–	1.206	2294	0
	[20 80]	6	28	1140.589	14.589	0.942	1.180	0.479	0.948	1.665	10232	0.023
140–node	[1 10]	10	10	240.325	45.466	0.991	1.108	0.739	–	1.209	158	0
	[1 100]	0	49	–	54.874	–	1.142[β]	0.675	0.925	1.194	6974	0.0149
	[20 20]	4	26	1920.000	52.194	0.946	1.152	0.739	0.947	1.296	7868.6	0.030

[α] All reported values are averaged over the 10 tested instances of each problem
[β] Ratio of heuristic value to best heuristic value found by CPLEX at termination

Table 4.1: Summary of Numerical Results for the TCVRPB

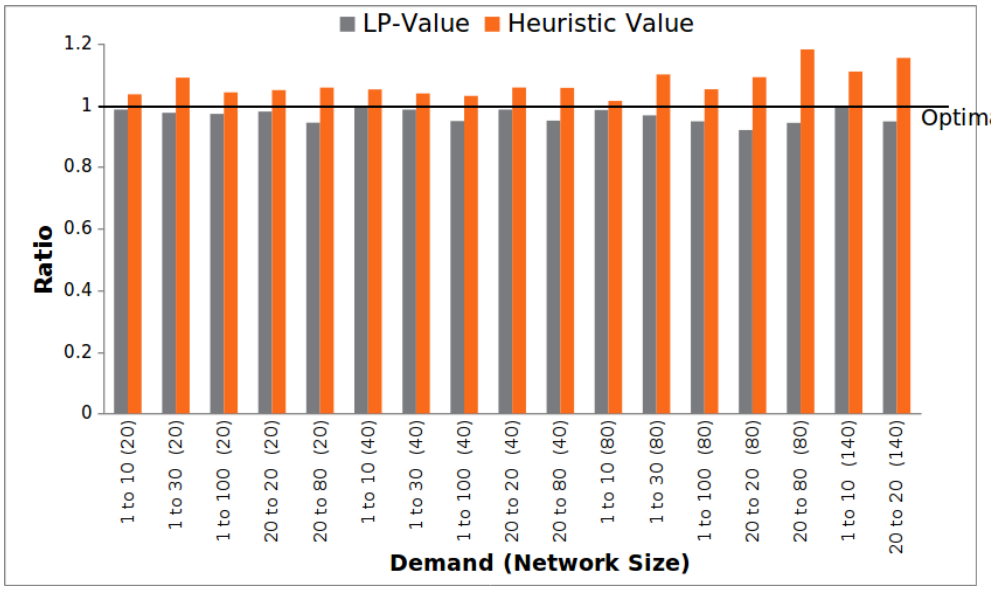


Figure 4.1: Comparison on LP, Optimal, and Heuristic Values

4.2.1 Solution Quality

All the 20–node and 40–node, and almost all of the 80–node test instances were solved to optimality within the pre–defined time limit. The solution quality can be deciphered by noting the performance of the LP–relaxation, the heuristic, and the number of branch–and–bound nodes.

LP–Relaxation The LP/IP ratio in Table 5.1 reports how close the LP–relaxation was to the optimal solution. Generally, the higher this ratio, the better the IP performance, as a high LP/IP ratio aids the branch–and–bound procedure. The LP–relaxation is particularly tight for the [1 10] and [1 30] demand distributions. For the [1 10] demand, the LP value is at least 98.1% the optimal solution value. This demand distribution is

much lesser than the vehicle capacity and, as a result, the LB was able to pack the nodes in a more efficient manner. Since the lower bound constraint was added as a valid inequality, the LP–relaxation of the problem is also very good. It can also be seen that as the mean of the demand distribution increases, the LP/IP ratio decreases. This again can be attributed to the fact that as the demand increases, the number of y variables split will also increase. For the instances where the IP was not solved to optimality, the LP/LastUB ratio demonstrates how close the LP value is to the best heuristic solution value found by CPLEX at termination. Notice that the LP–relaxation values are still very tight. The lowest reported LP/IP ratio for the demands [1 100] and [20 80] is around 92.5%. Overall, the LP/IP ratio percentage is consistently in the upper 90s implying that the IP–formulation, together with the added inequalities, are able to generate a very tight solution space.

Number of Nodes The number of branch–and–bound nodes required to solve the IP to optimality increases as both the mean of the demand and the demand distribution increase. Almost all the test instances with demand [1 10] are solved by CPLEX at the root node itself. Now, consider the instances with mean demand 50, that is the instances with demand [1 100] and [20 80]. For both these cases, the number of nodes required is much higher than the other instances. Also, as the demand [1 100] requires more nodes than [20 80]. Secondly, the number of node required is directly related to the total number of vehicles, $|K|$, in the IP–formulation.

The nodes increase with an increase in the number of vehicles. For example, for the demand distributions [1 100] and [20 80], the number of vehicles is much higher than for other demand distributions and this severely affects the total number of branch–and–bound nodes too.

Heuristic Solution The UB/IP ratio reports the performance of the heuristic solution with respect to the optimal solution. For the 20–node and 40–node instances the heuristic performs consistently well and is within 3 – 5% of the optimal solution. But, for the 80–node and 140–node cases, the heuristic solution is around 10% of the optimal solution value. Moreover, here too, to the heuristic solution seems to be directly affected by the number of vehicles. The more the number of vehicles required by the heuristic, the greater the UB/IP ratio. For example, for the 80–node and 140–node instances with demand [20 80] and [1 100], the heuristic is about 15–18% of the optimal solution value. Also, irrespective of network size, the heuristic performs very well for lower demand distributions.

Only 5 instances of the 80-node problems were unsolved and 16 instances of the 140–node problems were unsolved. For these unsolved instances, at termination, the best integer value found by CPLEX was between 1 –3 % of the optimal value. The Figure 4.1 shows the closeness of the LP and heuristic values to the optimal solution.

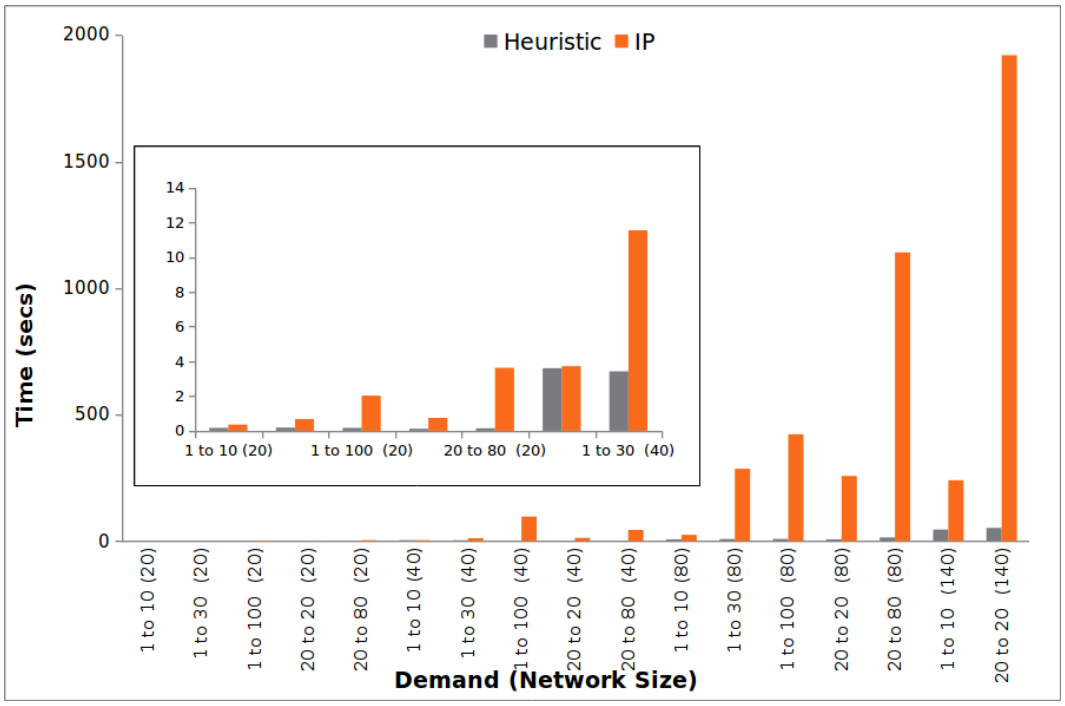


Figure 4.2: Computation Time for IP and Heuristic

4.2.2 Computational Performance

The Figure 4.2 shows the computation times required by the IP and the heuristic. As expected, as the network size increases, the optimal solution time and the heuristic computation time also increase. The optimal solution is computed fairly quickly for the 20–node and 40–node problems. Moreover, as seen previously, the computation time increases with increase in demand mean and demand distribution. For the 80–node instances, CPLEX was able to solve only 6 instances of the [20 80] demand to optimality. For the 140–node networks, CPLEX was unable to solve any of the [1 100] demand instances to optimality within an hour. The heuristic solution time, too, increased substantially for the 140–node networks. For the 80–node networks, the heuristic took a maximum of 14 seconds, while for the 140–node networks, the heuristic took 54 seconds to find a feasible solution. Also, the heuristic always outperforms the IP in terms of solution time. Here, too, there is a direct relation between the number of vehicles and the computation times. As the number of vehicles increase, the computation time increases. This is especially true for the [1 100] and [20 80] demand cases. The heuristic requires a higher number of vehicles for these instances and this, in turn, increases the computation times.

There seems to be a direct relation between higher fleet sizes ($|K|$) and solution and computational time quality. Therefore, a few further modifications that will reduce the total fleet size are suggested in the next section. Secondly, to better understand how these improvements affect the solution and computational performance, the experimental design is modified to reduce the

degrees of freedom in the parameter generation.

4.3 Further Improvements

A new heuristic, that reduces the number of vehicles, $|K|$, is described here. Recall that in the original heuristic, the clustering step was performed on the original tree $T = (N, A)$. Here, the clustering step is performed separately on the linehaul, T_L, and backhaul, T_B, trees. This will result in a set of vehicles that will each contain a subset of linehaul nodes and a set of vehicles that will each contain a subset of backhaul nodes. Then, an assignment problem is solved to determine the matching of the linehaul and backhaul vehicles.

For each tree, T_L and T_B, the clustering procedure of the heuristic proceeds as follows. Pick a non–grandparent node i farthest from the depot. A non–grandparent node i is a node whose children $c(i)$ have no children. In other words, the children $c(i)$ are all leaf nodes. The nodes $i \cup c(i)$ are packed into a bin with capacity Cap. The nodes $i \cup c(i)$ are then removed from the tree and are replaced by the bins into which these nodes have been packed. Obviously, these bins are now the children of the parent of node i, that is the bins form the set $c(P_i)$. This procedure continues till all the nodes have been accommodated into the bins.

Let $Vehicle_L$ ($Vehicle_B$) represent the set containing the set of linehaul (backhaul) nodes in each bin. The second step of the heuristic decides the pairing of linehaul and backhaul bins. Each pair will be served by the same vehicle. Note that, in this heuristic, it is possible that $|Vehicle_L| \neq |Vehicle_B|$.

There will be $||Vehicle_L| - |Vehicle_B||$ vehicles that will be purely linehaul or backhaul vehicles. Let m_{ij} be the optimal TTSPB cost of the tour that serves the subset of nodes in the i^{th} vehicle of $Vehicle_L$ and in the j^{th} vehicle of $Vehicle_B$. Further, let i take values $i = \{0, 1, \ldots, |Vehicle_L|\}$ and let j take values $j = \{0, 1, \ldots, |Vehicle_B|\}$. When i (j) is zero in m_{ij}, then it implies that j (i), is a purely backhaul (linehaul) route. In other words, assume that the 0^{th} linehaul and backhaul vehicles serve no nodes. The procedure for computing $Vehicle$ and m_{ij} is descibed in Algorithm 4.1.

The second step of the heuristic matches the linehaul and backhaul bin such that the total cost is minimized. An assignment problem is solved to obtain this matching as follows.

Minimize:

$$\bar{z} = \sum_{i=0}^{|Vehicle_L|} \sum_{j=0}^{|Vehicle_B|} m_{ij} w_{ij} \tag{4.2}$$

Subject to:

$$\sum_{j=0}^{|Vehicle_B|} w_{ij} = 1 \qquad \forall i \in \{1, 2, \ldots, |Vehicle_L|\} \tag{4.3}$$

$$\sum_{i=0}^{|Vehicle_L|} w_{ij} = 1 \qquad \forall j \in \{1, 2, \ldots, |Vehicle_B|\} \tag{4.4}$$

$$w_{ij} \geq 0 \qquad \forall i, \forall j \tag{4.5}$$

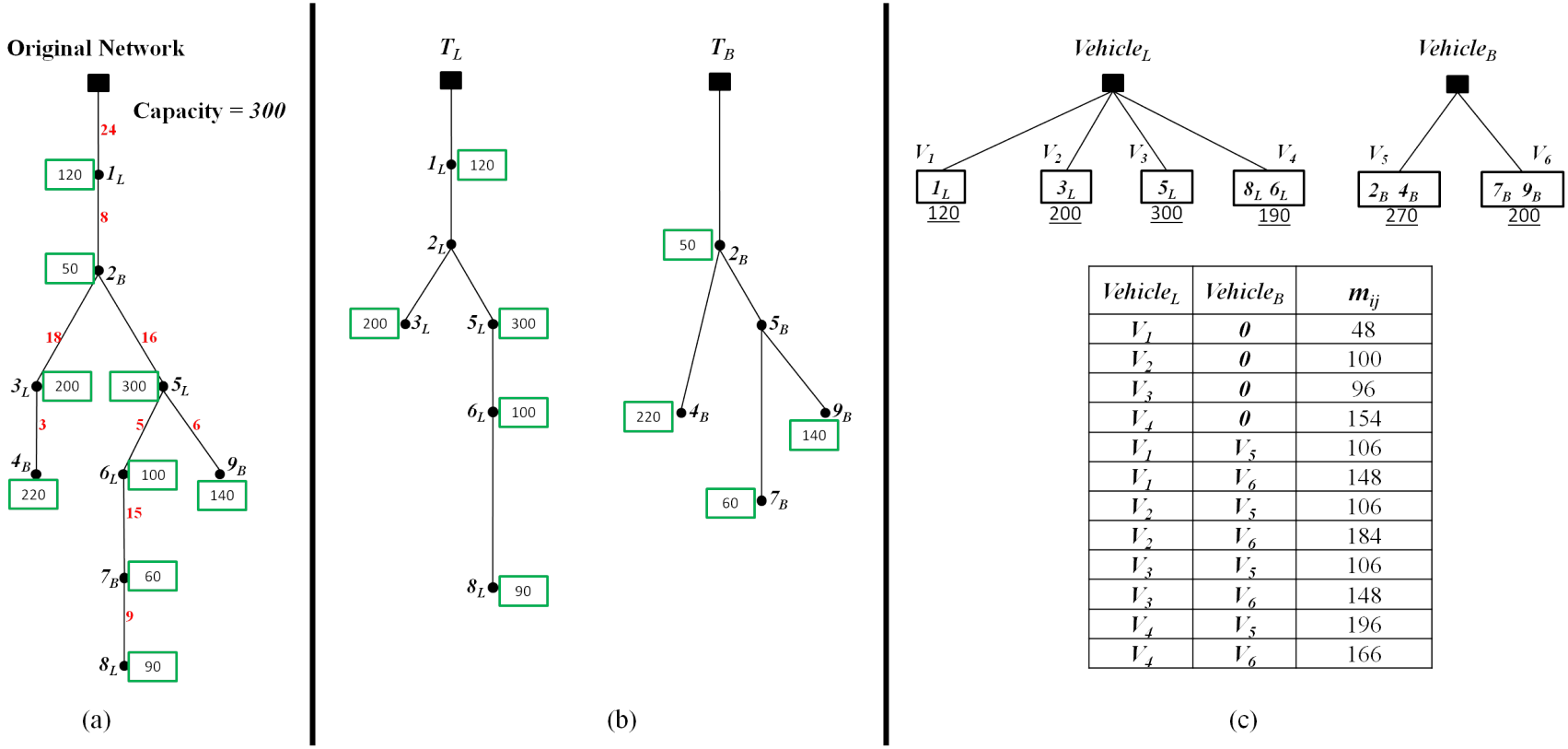
Algorithm 4.1 Heuristic #2 for TCVRPB

Input: $T = (N, A)$ $d_i \forall i \in N$ $c_{ij} \forall (i,j) \in A$
Output: $Vehicle_L, Vehicle_B, m_{ij}$

1: Use Algorithm 3.1 to construct $T_L = (N_L, A_L)$ and $T_B = (N_B, A_B)$
2: $LIST \leftarrow N_L$
3: **while** $LIST \neq \emptyset$ **do**
4: $NGP \leftarrow \{j \in LIST : (c(j) \neq \emptyset) \wedge (c(p) = \emptyset \forall p \in c(j))\}$
5: $i \leftarrow \max_{j \in NGP}\{L_j\}$
6: $(Vehicle, Bin) \leftarrow \texttt{BinPack}(i \cup c(i))$
7: $LIST \leftarrow LIST \setminus \{(i \cup c(i)) \wedge (LIST)\}$
8: $N \leftarrow N \cup Bin \setminus \{i \cup c(i)\}$
9: $P_j \leftarrow P_i \forall j \in Bin$
10: **end while**
11: $Vehicle_L \leftarrow Vehicle$
12: $LIST \leftarrow N_B$
13: **while** $LIST \neq \emptyset$ **do**
14: $NGP \leftarrow \{j \in LIST : (c(j) \neq \emptyset) \wedge (c(p) = \emptyset \forall p \in c(j))\}$
15: $i \leftarrow \max_{j \in NGP}\{L_j\}$
16: $(Vehicle, Bin) \leftarrow \texttt{BinPack}(i \cup c(i))$
17: $LIST \leftarrow LIST \setminus \{(i \cup c(i)) \wedge (LIST)\}$
18: $N \leftarrow N \cup Bin \setminus \{i \cup c(i)\}$
19: $P_j \leftarrow P_i \forall j \in Bin$
20: **end while**
21: $Vehicle_B \leftarrow Vehicle$
22: $Vehicle_{L_0} = \emptyset$ and $Vehicle_{B_0} = \emptyset$
23: **for** $i = 0 \rightarrow |Vehicle_L|$ **do**
24: **for** $j = 0 \rightarrow |Vehicle_B|$ **do**
25: $L \leftarrow Vehicle_{L_i}, B \leftarrow Vehicle_{B_j}$
26: $m_{ij} \leftarrow TTSPB_{cost}$ calculated using Algorithm 3.2
27: **end for**
28: **end for**

The variable w_{ij} will be equal to 1 if the linehaul nodes in bin i and the backhaul nodes in bin j are served by the same vehicle. Since this is an assignment problem, solving the LP will return binary values for w_{ij}. The first (second) constraint enforces that every linehaul (backhaul) bin is either matched to a backhaul (linehaul) bin or to the depot. The objective function returns the cost of this matching, which is equal to the heuristic solution value.

The heuristic is explained with the Figure 4.3. The original tree network is shown in Figure 4.3(a). The edge costs are denoted in red and the demand at every node is shown in a green box near the node. The linehaul and backhaul nodes are sub–scripted L and B, respectively. First, the linehaul and backhaul trees, T_L and T_B respectively, are constructed as shown in Figure 4.3(b) using the Algorithm 3.1. The clustering procedure is then carried out separately for each tree. This results in the sets $Vehicle_B = \{V_1, V_2, V_3, V_4\}$ and $Vehicle_B = \{V_5, V_6\}$ as shown in Figure 4.3(c). The resulting costs of grouping the linehaul and backhaul vehicles, m_{ij}, is shown in the table in Figure 4.3(c). Next, the assignment problem is solved using these m_{ij} costs and the LP–formulation given previously. In the final solution, the vehicles V_2 and V_5, and V_4 and V_6 are combined, while V_1 and V_3 serve only linehaul nodes. The resulting cost of the heuristic solution is 416 and the total number of vehicles used is 4.



$Vehicle_L$	$Vehicle_B$	m_{ij}
V_1	**0**	48
V_2	**0**	100
V_3	**0**	96
V_4	**0**	154
V_1	V_5	106
V_1	V_6	148
V_2	V_5	106
V_2	V_6	184
V_3	V_5	106
V_3	V_6	148
V_4	V_5	196
V_4	V_6	166

Figure 4.3: An Illustrative Example for the Second Heuristic

4.3.1 Additional Computational Results

The effect of reducing the number of vehicles that are input to the IP–formulation is studied here. The second heuristic that was described previously, intuitively speaking, will require lesser number of vehicles than the first heuristic. This is because each tree is treated separately, which ensures maximum packing. In order to better understand how the number of vehicles affects the exact solution time and performance, the test instances were modified as follows. Note that, the test instances defined previously gave an overall sense of the performance of the solution methods for different demand profiles and network sizes. However, to gain a better understanding of how the first and second heuristics affect the exact solution, the experiments are setup as follows.

Firstly, it was noted that as the mean and distribution of the demand increased, the IP solution time increased. Therefore, to gain a better insight into this behavior, for the computational tests performed here, the mean of the demand was kept constant while the demand was varied. The following demand profiles were tested: [1 100], [10 90], [20 80], [40 60], and [50 50]. Secondly, all the computational tests were carried out on 40 node networks, testing 10 randomly generated networks for each demand profile. The solution time and quality were tested separately for the cases where the IP was initialized using the first and second heuristic.

A summary of the computational results is presented in Table 4.2. The summary contains the following additional information:

$|K|_{H_1} - v$ Recall that v recorded the total number of vehicles used at optimality. Therefore, $|K|_{H_1} - v$ and $|K|_{H_2} - v$ report the difference between the total number of vehicles used by the heuristic (and input into the IP) and the total number of vehicles used at optimality for the first and second heuristic, respectively.

$\frac{LP}{IP_{H_1}}$ The ratio of the LP–relaxation value to the optimal value.

$\frac{LP0_{H_1}}{LP0_{H_2}}$ CPLEX is sometimes able to obtain an LP–value higher than the LP–relaxation value at the root node. The ratio of the LP–values at the root node when the IP is initialized by the first and second heuristic is reported by this metric.

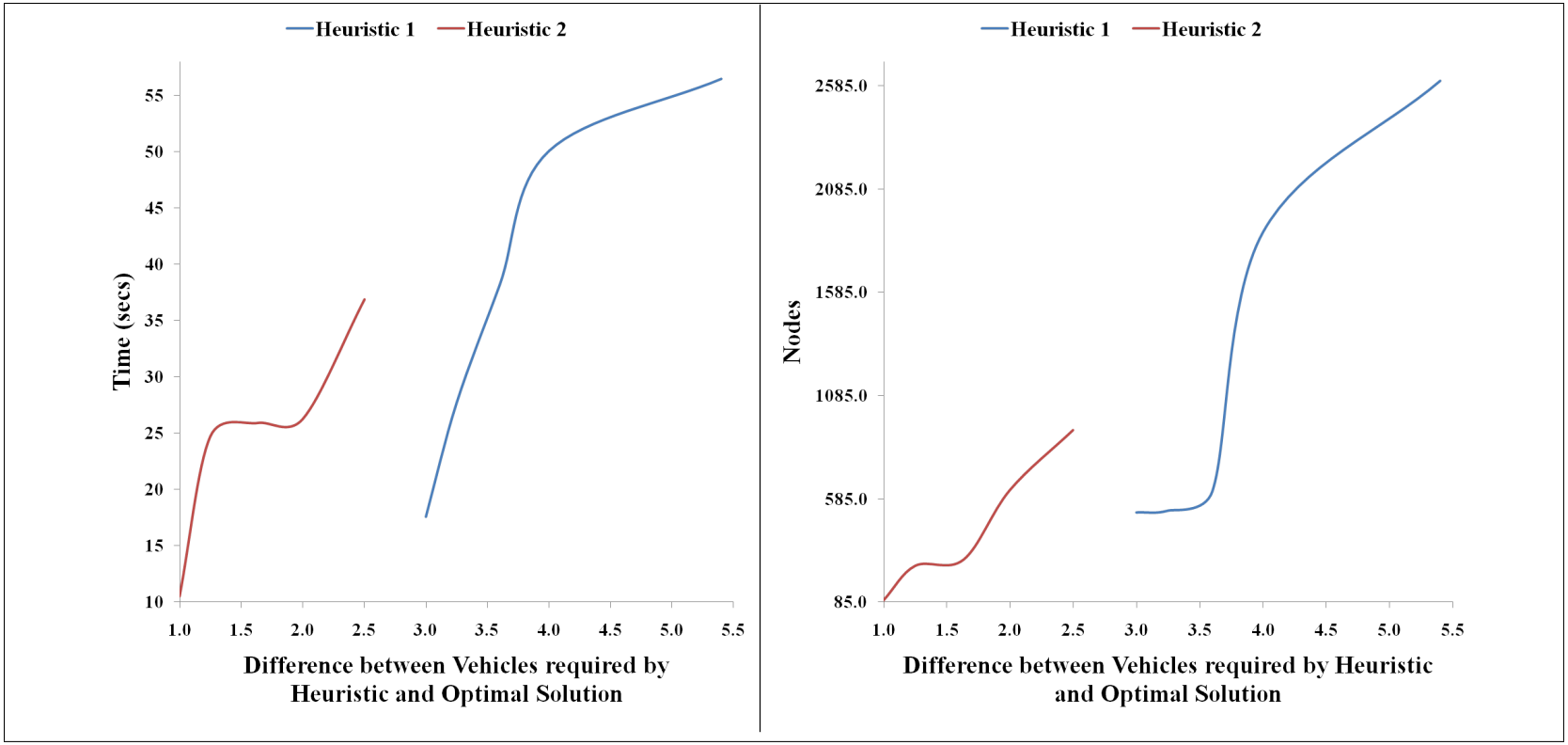
$\frac{UB_{H1}}{UB_{H2}}$ This is the ratio of the heuristic upper bound value found by the first and second heuristics. This also the first upper bound used by CPLEX as it is initialized with the heuristic solution.

Table 4.2 shows that the second heuristic consistently requires lesser number of vehicles than the first one. Moreover, as the demand variance decreases, the difference between the number of vehicles required by the optimal solution and the second heuristic also decreases. The initial solution provided to CPLEX by either heuristic does not seem to affect the LP–relaxation solution value. However, CPLEX seems to calculate a stronger LP–value at the root node when the number of vehicles in the IP is lower. Another interesting fact to note is that the optimal solution found by CPLEX remained the same

irrespective of the heuristic that was used to initialize it. This implies that the second heuristic was capable of finding an upper bound using a lower number of vehicles than the first one without changing the resulting optimal cost.

The number of vehicles input to the IP directly affects the computational performance. The time required by CPLEX and the nodes in the branch–and–bound process decrease with a decrease in the number of vehicles. Using the second heuristic as an initial solution reduces the time and nodes required by CPLEX as the number of vehicles in the second heuristic are lower. The graphs in Table 4.2 show the effect of the number of vehicles on the time and nodes required by CPLEX to find an optimal solution. One can conclude that a quicker solution can be obtained by warm–starting the IP using the second heuristic than the first one.

Demand	Solved$^{\alpha}$	$\lvert K\rvert_{H_1} - v$	$\lvert K\rvert_{H_2} - v$	$\frac{LP}{IP_{H_1}}$	$\frac{LP}{IP_{H_2}}$	$\frac{LP0_{H_1}}{LP0_{H_2}}$	$\frac{LP0_{H_1}}{LP}$	$\frac{LP0_{H_2}}{LP}$	$\frac{UB_{H1}}{UB_{H2}}$	Time-OPT$_{H_1}$ (secs)	Time-OPT$_{H_2}$ (secs)	Nodes$_{H_1}$	Nodes$_{H_2}$
[1 100]	10	5.4	2.5	0.968	0.968	0.986	1.009	1.014	1.014	56.515	36.897	2609.0	916.8
[10 90]	10	4.0	2.0	0.966	0.966	0.999	1.010	1.001	1.023	50.109	26.290	1876.5	627.2
[20 80]	10	3.6	1.6	0.955	0.955	0.988	1.013	1.013	1.088	38.202	25.934	621.6	291.0
[40 60]	10	3.3	1.3	0.938	0.938	0.995	1.008	1.013	0.968	27.724	24.750	528.2	261.0
[50 50]	10	3.0	1.0	0.967	0.967	0.983	1.017	1.014	1.099	17.586	10.535	519.5	94.5



$^{\alpha}$ All reported values are averaged over the 10 tested instances of each problem

Table 4.2: Summary of Numerical Results for the TCVRPB after Improvements

4.4 Conclusions

In the preceding two chapters, a variant of the TVRP in which the nodes are divided in to two subsets – linehaul and backhaul nodes was introduced and studied in detail.

A lower bound on the TCVRPB was derived by using its relation with bin–packing problems, and a few properties and key observations that hold true at optimality were delineated. Two IP formulations were proposed to solve the problem. The second IP was formulated by transforming the tree into an equivalent network. It was shown that this IP had a fewer variables and that its LP–relaxation was stronger than the first one. A few valid inequalities were added to the second IP formulation in an effort to increase its LP–relaxation value (or reduce the duality gap).

An algorithm, which, at every stage deletes at least one node from the network was presented. The algorithm had two main steps – finding the customers that are serviced by each vehicle, and constructing the optimal routes for that vehicle. The optimal vehicle routes were constructed by using the algorithm for the Traveling Salesman Problem on Trees with Backhauls (TTSPB), a polynomial algorithm for which was developed in this dissertation.

The second IP Formulation and the heuristic were tested on problem instances of varying sizes and demand profiles. 10 instances of each network size and demand profile was tested. Computational times for solving the IP depended not only on the network size, but also on the demand distribution

and the number of vehicles in the IP. Computational time increased with an increase in problem size, as supported by intuition. The computational time and number of branch–and–bound nodes increased with an increase in the number of vehicles and an increase in the demand variation. Further, the quality of the solution also decreased with an increase in the number of vehicles and an increase in the demand variation. The LP–relaxation was particularly tight for the [1 10] and [1 30] demand distributions. For the [1 10] demand, the LP value was at least 98.1% the optimal solution value. The lowest reported LP/IP ratio for the demands [1 100] and [20 80] was around 92.5%.For the 20–node and 40–node instances the heuristic performed consistently well and was within 3 – 5% of the optimal solution. But, for the 80–node and 140–node cases, the heuristic solution was around 10% of the optimal solution value.

A direct relation between higher fleet sizes ($|K|$) and solution and computational time quality resulted in a few modifications to the heuristic which reduced the total fleet size. The clustering step was performed separately on the linehaul, T_L, and backhaul, T_B, trees. Then, an assignment problem was solved to determine the matching of the linehaul and backhaul vehicles. The time required by CPLEX and the nodes in the branch–and–bound process decreased with a decrease in the number of vehicles. Using the second heuristic as an initial solution reduced the time and nodes required by CPLEX, as the number of vehicles in the second heuristic are lower.

Future research includes exploring stochastic and online versions of the problem. Exploring other solution techniques – especially column generation

techniques, as tree structures might be particularly amenable to such techniques – is also a future research direction.

Chapter 5

TVRP with Fixed Fleets

A constrained case of the TVRP – where the vehicle fleet is capacitated and heterogeneous (HTCVRP) is studied in this chapter. A heuristic algorithm that explicitly considers the tree structure are presented here. No heuristic techniques exist for solving fixed fleet TVRPs.

5.1 Problem Definition

The problem considered in this chapter can be briefly defined as follows. Given a tree network, $T = (N_D, A)$; non–negative arc costs, $c_{ij} \forall (i,j) \in A$; demand at each node, $d_i \forall i \in N$; and a heterogeneous *fixed* fleet of vehicles located at a depot: find a collection vehicle routes, such that

(i) total distance traveled by (total operating cost of) all *used* vehicles is minimized

(ii) demand at each node is satisfied by *exactly* one vehicle

(iii) total demand serviced by a vehicle does not exceed its capacity

(iv) all vehicle routes begin and end at the depot

This problem is the Capacitated Heterogeneous Vehicle Routing Problem on Trees (HTCVRP). It is assumed that the arc costs remain constant over all vehicle types, and that the total fleet mix is given and finite. Heuristics for both cases – with and without fixed vehicle costs – are presented.

The HTCVRP has embedded within it the Generalized Assignment Problem (GAP), the Bin Packing Problem (BPP) and the Tree Traveling Salesman Problem (TTSP). The heuristic for the HTCVRP, proposed here, iteratively finds seed nodes using Upper Bound (UB) heuristics for the BPP, assigns a vehicle to each seed node, then uses a Lagrangian–based GAP algorithm to assign nodes to vehicles located at the seed node. The TTSP is solved to find the optimal route for every *used* vehicle. The TTSP is trivially solvable in polynomial time (Tsitsiklis, 1992). Finally, a refining operation based on the savings heuristic (Clarke and Wright, 1964a) is employed to further reduce the solution cost.

The rest of this chapter is organized as follows. In Section 5.3, some exact solution methods and their relation to the Generalized Assignment Problem and Bin Packing Problem are discussed. The heuristic algorithm is presented in Section 5.3. The computational results obtained by implementing the heuristic and exact methods are presented in Section 5.6.

5.2 Preliminaries

Let K be the set of vehicles, $Cap_k : k \in K$ denote each vehicle's capacity and $f_k : k \in K$ denote the fixed operating cost of each vehicle.

Assume that the vehicles are indexed in decreasing order of their capacities, that is $Cap_{k_1} \geq Cap_{k_2} \geq \ldots \geq Cap_{k_{|K|-1}} \geq Cap_{k_{|K|}}$. In this chapter, the number of vehicles in the fleet and the fleet mix is given. No heuristics or approximation algorithms for TVRPs exist in the open literature that deal with either heterogeneous fleets or fixed fleets. The robustness of the heuristic in dealing with non-fixed and homogeneous fleets is demonstrated in the next section. Recall that, the set of nodes in the unique Path from the Depot to a node i is denoted by PfD_i. That is, $PfD_i = N_R : R = \{i\}$. The cumulative demand of the nodes in PfD_i is given by $Dem_i = \sum_{j \in PfD_i} d_j$ and the cost of the path from the depot to i is given by $L_i = \sum_{j \in PfD_i} \sum_{q \in PfD_i} c_{j,q} : (j,q) \in A$. It is assumed, without loss of generality, that the depot has degree 1. Further, it is also assumed that the arc costs c_{ij} do not vary with respect to vehicle $k \in K$.

5.2.1 Lower Bounds

As the HTCVRP is closely related to the bin–packing problem, a very naive lower bound can be developed using this relation. The bin–packing problem is defined as follows. Given a set of p items with weights $w_1, w_2, \ldots, w_p$; and bins of capacity Q – pack the p items into bins such that the total number of bins are minimized. A simple lower bound on the number of bins required to *pack* all p items is given by:

$$\left\lceil \frac{\sum_{i=1}^{p} w_i}{Q} \right\rceil \tag{5.1}$$

Remember that according to our definition, vehicle k_1 is the vehicle with maximum capacity Cap_{k_1}. Now, consider a sub–tree S_i rooted at node i, the very minimum number of vehicles required to serve the nodes, say, $V_{\min_i}$ is given by:

$$V_{\min_i} = \left\lceil \frac{\sum_{j \in S_i} d_j}{Cap_{k_1}} \right\rceil \tag{5.2}$$

Then, a naive lower bound, $\underline{z}_{\min}$, on the optimal objective value, z^*, to the HTCVRP is given by:

$$\underline{z}_{\min} = \sum_{i \in N} V_{\min_i}(c_{P_i i} + c_{i P_i}) \tag{5.3}$$

At the very least $V_{\min_i}$ vehicles will enter the sub–tree S_i to serve its nodes. This implies that at least $V_{\min_i}$ vehicles will traverse the edge $\{P_i, i\}$ once on their way from the depot to the sub–tree, and once on their way back (after service) from the sub–tree to the depot.

The lower bound computed using Equation 5.3 can be improved as follows. Let the new minimum number of vehicles required to serve a sub–tree be V_i, the sub–tree demand at every iteration be `SubTD`, and the vehicle index be denoted as `Index`. Now, V_i can be computed using Algorithm 5.1. A better lower bound, $\underline{z}$, can be calculated as follows:

$$\underline{z} = \sum_{i \in N} V_i(c_{P_i i} + c_{i P_i}) \tag{5.4}$$

Since, $V_i \geq V_{\min_i} \ \ \forall i \in N$, it can be stated that $\underline{z} \geq \underline{z}_{\min}$. Therefore, Equation 5.4 is a better lower bound.

Algorithm 5.1 An Improved Lower Bound for HTCVRP

Input: $d_i \ \ \forall i \in N$
Input: $Cap_k \ \ \forall k \in K$
Output: $V_i \ \ \forall i \in N$

```
for i = 1 → n do
    SubTD = Σ_{j∈S_i} d_j
    V_i = 0
    Index = 1
    while SubTD > 0 do
        SubTD ← SubTD − Cap_{k_Index}
        V_i ← V_i + 1
        Index ← Index + 1
    end while
end for
```

5.3 Exact HTCVRP Solution Methods

Exact HTCVRP solution methods were developed by Chandran and Raghavan (2008) and Mbaraga et al. (1999). Chandran and Raghavan (2008) developed two integer programs for solving TCVRPs. The first one builds off the fact that, once the nodes that are part of a vehicle route are determined, nodes will be served in the order of their DFS index, a similar IP was also developed by Busch (1990). The second formulation uses the fact that there is only one path between a node and any other node in the depot, and that every node has a unique parent node. The second formulation (HTCVRP(1)), which will be further used in the heuristic proposed here, is given below:

Formulation: **HTCVRP(1)**

$$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ traverses arc } (i,j) \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{if vehicle } k \text{ serves node } i \\ 0 & \text{otherwise} \end{cases}$$

Minimize:

$$2 \times \sum_{i} \sum_{j} \sum_{k} c_{ij} x_{ijk} \tag{5.5}$$

Subject to:

$$x_{P_i ik} \geq x_{ijk} \qquad \forall k \in K, i \in N, j \in c(i) \tag{5.6}$$

$$x_{P_i ik} \geq y_{ik} \qquad \forall k \in K, \in N \tag{5.7}$$

$$\sum_{i \in N} y_{ik} d_i \leq Cap_k \qquad \forall k \in K \tag{5.8}$$

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N \tag{5.9}$$

$$y_{ik} \in \{0, 1\} \qquad \forall i \in N, k \in K \tag{5.10}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall (i, j) \in A, k \in K \tag{5.11}$$

The objective function minimizes the total distance traveled. It is multiplied by 2 because each vehicle has to return to the depot after service. Constraint 5.6 states that if a vehicle k traverses arc (i, j), then it should also have traversed the arc (P_i, i). This ensures that if a vehicle traverses an arc then it must first traverse the parent arc to reach that arc. Constraint 5.7 ensures that a vehicle decides to serve node i then it should travel along the arc leading to that node i, that is arc (P_i, i). Constraint 5.8 enforces that every node is serviced by only one vehicle, while constraint 5.9 ensures that the vehicle capacity is not exceeded. Some valid inequalities, to further expedite the convergence of the IP, were also proposed by Chandran and Raghavan (2008). The formulation can easily accommodate fixed costs by modifying the

objective function as follows:

$$2 \times \sum_{i} \sum_{j} \sum_{k} c_{ij} x_{ijk} + \sum_{k} f_k x_{12k} \tag{5.12}$$

Let the depot be denoted as node 1. Since the depot has a degree of 1, all used vehicles will always traverse arc$(1, 2)$. Thus, fixed costs can be incorporated into the formulation without defining new variables or constraints.

Mbaraga et al. (1999) define a set covering–based formulation for the HTCVRP, which is solved using column generation. The master problem of the column generation scheme solves the HTCVRP for a subset of variables, these variables are then parsed to the subproblem. The subproblem is a capacitated shortest path problem, whose arc costs are defined such that the shortest path will generate a column with the most negative reduced cost, which will in turn be parsed to the master problem. The tree arcs and costs are modified to form an acyclic graph for every vehicle as vehicles serve nodes in DFS order. The constrained shortest path (SP) is then solved on these acyclic graphs using the algorithm proposed by Desrosiers et al. (1995). The SP algorithm has a pseudopolynomial running time.

5.4 Heuristic for HTCVRP

Let $L_{i,j}$ be the cost of the path between nodes i and j. It is also known that, given a vehicle and the nodes it is going to serve, the least cost vehicle route will visit the nodes in increasing order of the DFS index (Tsitsiklis, 1992). Given this, the formulation HTCVRP(1) can be rewritten such that

the objective function enforces DFS movement, while the constraints enforce the capacity and service requirements. When no fixed costs are given, the new formulation, HTCVRP(2), with depot as node 1, is given as follows:

Formulation: **HTCVRP(2)**

Minimize:

$$\sum_{k \in K} \Delta(k) \tag{5.13}$$

Subject to:

Constraints 5.8 – 5.10

Where, $\Delta(k)$, for every vehicle $k \in K$ is defined as:

$$\Delta(k) = L_{1,p} + \sum_{i:y_{ik}=1} L_{i,q} + L_{r,1} \tag{5.14}$$

$$p = \min_{j:1<j} \{j | y_{jk} = 1\} \tag{5.15}$$

$$q = \min_{j:i<j} \{j | y_{jk} = 1\} \tag{5.16}$$

$$r = \max_{j:j>1} \{j | y_{jk} = 1\} \tag{5.17}$$

The objective function 5.14 is the sum of the costs of serving the lowest DFS indexed node from the depot; serving the non–depot nodes in DFS order; and traversing from the highest indexed node back to the depot for each vehicle. The constraints of the formulation HTCVRP(2) are the same as that of a generalized assignment problem (GAP). The objective function, however, is more complex. The objective is dependent not only on the cost accrued by a vehicle to serve a node, but also on the order in which the nodes are served. Therefore,

it is difficult to assign these costs *a priori*, because costs for every precedence order of nodes have to be defined. Therefore, Cap–HTVP(2) cannot be solved as a GAP.

When fixed costs, $f_k \forall k \in K$ are given, the formulation can be modified as given below.

Formulation: **HTCVRP(3)**

$$z_k = \begin{cases} 1 & \text{if vehicle } k \text{ is used} \\ 0 & \text{otherwise} \end{cases}$$

Minimize:

$$\sum_{k \in K} \Delta(k) + \sum_{k} f_k z_k \tag{5.18}$$

Subject to:

$$\sum_{i \in N} y_{ik} d_i \leq Cap_k z_k \qquad \forall k \in K \tag{5.19}$$

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N \tag{5.20}$$

$$y_{ik} \in \{0, 1\} \qquad \forall i \in N, k \in K \tag{5.21}$$

$$z_k \in \{0, 1\} \qquad \forall k \in K \tag{5.22}$$

$\Delta(k)$ is as defined in equations 5.14 – 5.17. The constraint set 5.19 – 5.22 is that of the capacitated facility location problem. However, the cost function $\Delta(k)$ in the objective is not a simple linear cost, because of which HTCVRP(3) cannot be solved as a CFLP.

However, current GAP and CFLP methods can be used to find a heuristic solution to HTCVRP if $\Delta(k)$ can be approximated as a linear function of y_{ik}. This is done using a method similar to the one described by Fisher and

Jaikumar (1981b). First the concept of a *seed node* for every vehicle is defined. A seed node of a vehicle is a node which is assigned to a vehicle *a priori*. That is, fix nodes $i_{k_1}, i_{k_2}, \ldots, i_{k_{|K|-1}}, i_{k_{|K|}}$ that will be served by vehicle $k_1, k_2, \ldots, k_{|K|-1}, k_{|K|}$ respectively. Next, compute a penalty, δ_{ik}, associated with inserting node i into the route of vehicle k. Thus, computing a heuristic solution to HTCVRP using GAP or CFLP solution methods consists of:

(a) finding seed nodes or customers

(b) computing $\delta_{ik} \forall i \in N, k \in K$

Once this is done, solving the GAP or CFLP will result in a set of nodes that every vehicle k will serve. Finding the optimal route for these set of nodes is trivial as it involves finding the TTSP for every vehicle k. Finally, in order to further reduce the HTCVRP heuristic solution value, a refining operation is also performed as described in Section 5.4.4.

5.4.1 Computing δ_{ik}

δ_{ik} can be interpreted as the penalty accrued by vehicle k for serving node i. This vehicle k already serves seed node i_k, so δ_{ik} computes the additional cost of serving i, given that k already serves i_k. Recall that the set of nodes in the unique path from the depot to the node i_k is denoted by PfD_{i_k}. Now, if $i \in PfD_{i_k}$ then this node i will already be *visited* by the vehicle k on its way to serving the seed node i_k. Therefore, the penalty associated with *serving* this node i, or inserting this node i into the vehicle route, will be zero.

For any node $i \notin PfD_{i_k}$, the penalty associated with inserting that node i into the vehicle route k is equal to twice the distance of that node from the last common node in PfD_{i_k} and PfD_i. This is explained in detail as follows. Consider Figure 5.1, let node i be the node to be inserted and node i_k be the seed node. Now, before node i is added, the cost of the vehicle route k is twice the cost of traversing from the depot to i_k, $2 \times L_{i_k}$. However, if node i is added to the route, the cost of vehicle k's route will be the sum of the costs of traversing from the depot to i_k, L_{i_k}, from i_k to i, $L_{i_k,i}$, and from i back to the depot, L_i. The penalty cost, δ_{ik} can now be computed. More formally:

$$\delta_{ik} = \begin{cases} 0 & \text{if } i \in PfD_{i_k} \\ L_{i_k} + L_{i_k,i} + L_i - 2 \times L_{i_k} & \text{if } i \in N \setminus PfD_{i_k} \end{cases} \tag{5.23}$$

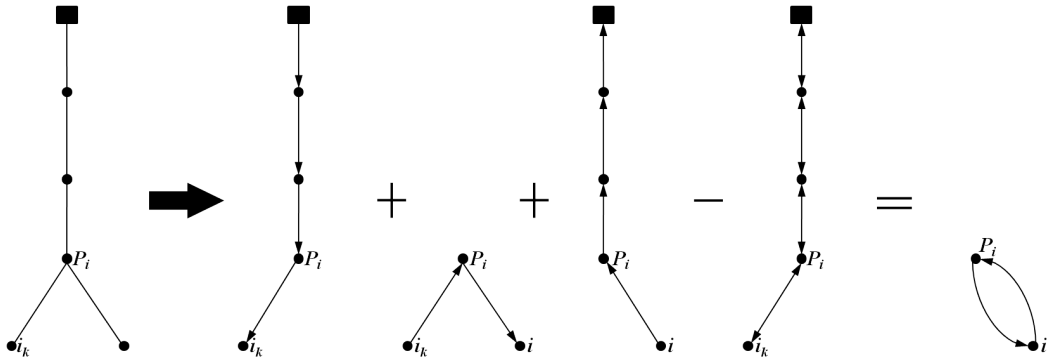


Figure 5.1: Computation of $\delta_{ik} \forall i \in N \setminus PfD_{i_k}, \forall k \in K$

When no fixed costs are given, the objective function 5.13 can be replaced with:

$$\sum_{i \in N} \sum_{k \in K} \delta_{ik} y_{ik} \tag{5.24}$$

and when fixed costs are given, the objective function 5.18 can be replaced with:

$$\sum_{i \in N} \sum_{k \in K} \delta_{ik} y_{ik} + \sum_{k} f_k z_k \tag{5.25}$$

5.4.2 Finding seed customers

Seed nodes have to be chosen such that the resulting GAP or CFLP solution is as close to the optimal solution as possible. Seed nodes should be selected such that, in the resulting GAP solution, the same vehicle does not serve two seed nodes. Secondly, seed nodes should be located such that total insertion cost, δ_{ik}, is minimized. Since the vehicle fleet is heterogeneous, the procedure assigns vehicles to nodes in decreasing order of capacity. The seed node selection procedure defined here has three steps:

Step 1 Assign vehicle k to seed node i_k

Step 2 Determine potential nodes that vehicle k will serve; delete these nodes

Step 3 Select the next seed node, i_{k+1}, from the remaining set of nodes

It can be seen that the number of iterations is equal to the number of vehicles. Starting with the entire tree, at every iteration of the procedure a sub–tree is deleted, so that the pool of seed node candidates keep decreasing. The node deletion at every stage ensures that no two seed nodes are served by the same vehicle. Secondly, nodes are deleted in such a manner that the resulting GAP

solution will be minimized. This is done by trying to guarantee that a vehicle k will choose to serve a node i such that $\delta_{ik} = 0$.

Assume that the nodes in the tree at iteration k is N_k and $|N_k| = n_k$. This implies that $n - n_k$ number of nodes have been deleted by iteration $k-1$. Next, it's easy to see the tree at iteration k is actually a covering sub–tree with $|R| = n_k$, therefore, the tree at an iteration k is denoted by CS_{R_k}. The tree at the first iteration is the original tree network T, with $n_1 = n + 1$ (the additional node is the depot). Lastly, note that at iteration k, $|K| - k + 1$ vehicles are yet to be assigned.

The function to assign a vehicle to a seed node, at iteration k, is denoted by `VehicleAssign`(k). `Contract`$_k$ is the function that deletes nodes from the tree CS_{R_k} such that the next seed node, i_{k+1}, that is chosen is not assigned to the vehicle k by the GAP.

The first method, `VehicleAssign`$_{H1}(k)$ is as follows. At every iteration k, the n_k nodes are sorted in decreasing order of their cost L_i. The function **quicksort**$(L_i : i \in N_k)$ performs this function and stores the nodes in the array $SORT_k$. The p^{th} element of $SORT_k$ is denoted by $SORT_k(p)$ Now, from the first $|K| - k + 1$ elements of $SORT_k$, the node with the highest cumulative demand Dem_i is selected to the be seed node i_k and is assigned to vehicle k. The details are presented in Algorithm 5.2.

The second method of assigning seed nodes, `VehicleAssign`$_{H2}(k)$ is a slight modification of the first. From the first $|K| - k + 1$ elements of $SORT_k$,

Algorithm 5.2 Computation of $\texttt{VehicleAssign}_{H1}(k)$ at iteration k

Input: N_k, $L_i \forall i \in N_k$, $Dem_i \forall i \in N_k$, f_k
Output: i_k
1: **if** $f_k \neq 0$ **then**
2: **for** $i = 1 \rightarrow n_k$ **do**
3: $L_i \leftarrow L_i + f_k$
4: **end for**
5: **end if**
6: $SORT_k \leftarrow$ **quicksort**$(L_i : i \in N_k)$
7: $i_k = \underset{i}{\text{argmax}}\{Dem(i) : i = SORT_k(p) | 1 \leq p \leq |K| - k + 1\}$

instead of selecting the node with the highest cumulative demand Dem_i to be the seed node i_k, the node that is farthest away from the seed node i_{k-1} is selected.

Proposition 5.1. *At every iteration k, the seed node i_k is a leaf node of the covering sub–tree CS_{R_k}*

Proof. Assume that the chosen seed node i_k is not a leaf node of the tree CS_{R_k}. But, any node that is a direct descendant of i_k will also be in the first $|K| - k + 1$ elements of $SORT_k$. Therefore, step 7 of Algorithm 5.2 will always select a leaf node. □

The next step in iteration k of the seed node selection procedure deletes nodes from the tree CS_{R_k} that will potentially be served by the current vehicle k which has been assigned seed node i_k. $\texttt{Contract}_k$ contains the set of nodes whose deletion from the tree CS_{R_k} results in the tree for the next iteration $k+1$, $CS_{R_{k+1}}$. Let $bin_{UB}(V_i)$ be the bin–packing upper bound, with bin capacity

Cap_k, on the number of vehicles required to serve sub–tree S_i. Starting at the leaf node i_k, and moving upwards in the tree, a sub–tree S_i is deleted from the tree CS_{R_k} if $bin_{UB}(V_i) < 2$. The reason for this is straightforward as this procedure deletes nodes closest to the seed node which can be served by vehicle k. Here, the first fit decreasing (FFD) procedure is used to calculate $bin_{UB}(V_i)$. The steps of this procedure are described in Algorithm 5.3.

Algorithm 5.3 Computation of set $\texttt{Contract}_k$ at iteration k

Input: CS_{R_k} $d_i : i \in N_k$ Cap_k
Output: $\texttt{Contract}_k$ $CS_{R_{k+1}}$

1: $i \leftarrow i_{k+1}$
2: CurrentNode $= i_k$
3: **while** $bin_{UB}(V_i) < 2$ **do**
4: CurrentNode $= i$
5: $i \leftarrow P_i$
6: **end while**
7: $\texttt{Contract}_k = \{i : i \in S_{\text{CurrentNode}} \cap N_k\}$
8: $N_{k+1} = N_K \setminus \texttt{Contract}_k$
9: $A_{k+1} = A_k \setminus \{(i,j) \in A_k : i \in \texttt{Contract}_k\} \setminus \{(i,j) \in A_k : j \in \texttt{Contract}_k\} \setminus \{(i,j) \in A_k : i \in \texttt{Contract}_k, j \in \texttt{Contract}_k\}$
10: $CS_{R_{k+1}} = (N_{k+1}, A_{k+1})$

5.4.3 Solving the Generalized Assignment Problem

As stated perviously, after the seed nodes have been identified, the next step involves solving the GAP (the CFLP if fixed costs are given). δ_{ik} is calculated for very seed node and every vehicle. The formulation HTCVRP(2) is then used to solve the GAP. Since the GAP is a very well known problem in logistics and operations research, many polynomial techniques that provide close to optimal solutions exist in the literature. The Lagrangian based LBR

and RGT developed by Jeet and Kutanoglu (2007) is employed here. It is easy to see that after solving the GAP, a feasible solution to the HTCVRP is always obtained.

5.4.4 Refining Operation

A feasible solution to the HTCVRP is obtained by solving the GAP (or CFLP). This solution can be refined further by performing a myopic savings operation. Let the current best solution value obtained by solving the GAP be $\bar{z}_w$ and the current best solution be $\bar{X}_w, \bar{Y}_w$. Where w is the number of vehicles used by the solution and $1 \leq w \leq |K|$. The refining operation examines one node at a time. Thus, the refining operation has $|N|$ iterations. First, for every node the total savings in cost obtained by removing that node from the current route and placing it in one of the available $w-1$ routes or a new $(w+1)^{th}$ route is calculated, provided that the insertion is possible. Let the solution cost of removing node i from the current route and placing it in route k be given by $z_{w_{k_i}}$ for each node i. That is for every node i, the savings Sav_i is:

$$Sav_i = \max_k \{\bar{z}_w - \bar{z}_{w_{k_i}}\} \quad \text{where} \;\; k = \{1, 2, \ldots, w, w+1\} \tag{5.26}$$

$LIST$ is the set of nodes in non–increasing order of the Sav_i value. $LIST_r$ denotes the r^{th} element of $LIST$. Now, if the maximum savings $LIST_1$ is positive, then the node i which yields this savings is inserted into its new location and $\bar{z}_{w_{k_i}}$ is the new incumbent best solution. If the maximum savings is zero, then the node i which yields this zero savings is discarded and the next

iteration is initiated. Let $\bar{z}$ be the best heuristic solution value and $\bar{X}, \bar{Y}$ be the best heuristic solution obtained at the end of the refining operation. The details of the algorithm are presented in Algorithm 5.4.

Algorithm 5.4 Refining Operation to Improve Heuristic Solution

Input: $\bar{X}_w,\ \bar{Y}_w,\ \bar{z}_w$
Output: $\bar{X},\ \bar{Y},\ \bar{z}$

Calculate $Sav_i = \max_k\{\bar{z}_w - \bar{z}_{w_{k_i}}\}$ where $k = \{1, 2, \ldots, w, w+1\}$ $\forall i \in N$
$LIST = \{p : Sav_{p-1} \geq Sav_p \geq Sav_{p+1}\}$ $|LIST| = |N|$
$\bar{z} = \bar{z}_w$ $\bar{X} = \bar{X}_w$ $\bar{Y} = \bar{Y}_w$
while $LIST \neq \emptyset$ **do**
  if $LIST_1 > 0$ **then**
    $\bar{z} \leftarrow \bar{z} - Sav_{LIST_1}$
    Calculate w
    $\bar{z}_w \leftarrow \bar{z}$
    Update $\bar{X}$ and $\bar{Y}$
    $LIST = LIST \setminus LIST_1$
  else
    $LIST = LIST \setminus LIST_1$
  end if
  Calculate Sav_i $\forall i \in LIST$
  $LIST = \{p : Sav_{p-1} \geq Sav_p \geq Sav_{p+1}\}$
end while

5.4.5 Heuristic HTCVRP

The complete heuristic proceeds in the following manner:

Step 1 Find seed customers using functions `VehicleAssign`(k) and `Contract`$_k$ till either $|K|$ nodes have been assigned or all nodes N have been covered

Step 2 Calculate δ_{ik} for the seed customers i_k

Step 3 Solve GAP using HTCVRP(2) with objective function 5.24, if fixed costs are given then solve HTCVRP(3) with objective function 5.25

Step 4 Calculate the current best heuristic solution $\bar{X}_w$ and $\bar{Y}_w$; and best heuristic solution value $\bar{z}_w$

Step 5 Perform the refining operation described in Algorithm 5.4 to obtain $\bar{X}$, $\bar{Y}$ and $\bar{z}$

The pseudocode for the heuristic is given in Algorithm 5.5

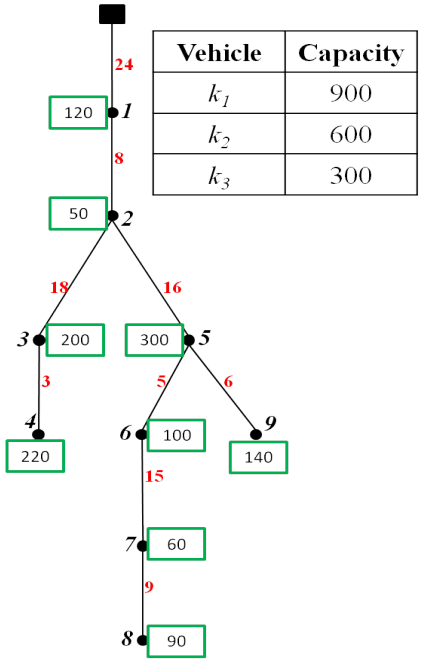
Algorithm 5.5 Heuristic Algorithm for HTCVRP

Input: $T = (N_D, A)$ $d_i \forall i \in N$ $c_{ij} \forall (i,j) \in A$ $Cap_k \forall k \in K$
Output: $\bar{z}$ $\bar{X}$ $\bar{Y}$

```
N_1 = N
for k = 1 → |K| do
    if N_k ≠ ∅ then
        Compute i_k using VehicleAssign(k)
        Compute δ_ik ∀i ∈ N
        Compute set Contract_k
    end if
    N_{k+1} = N_k \ Contract_k
end for
Solve GAP using HTCVRP(2) with objective function 5.24
Calculate z̄_w, X̄_w and Ȳ_w
z̄, X̄ and Ȳ using Algorithm 5.4
```

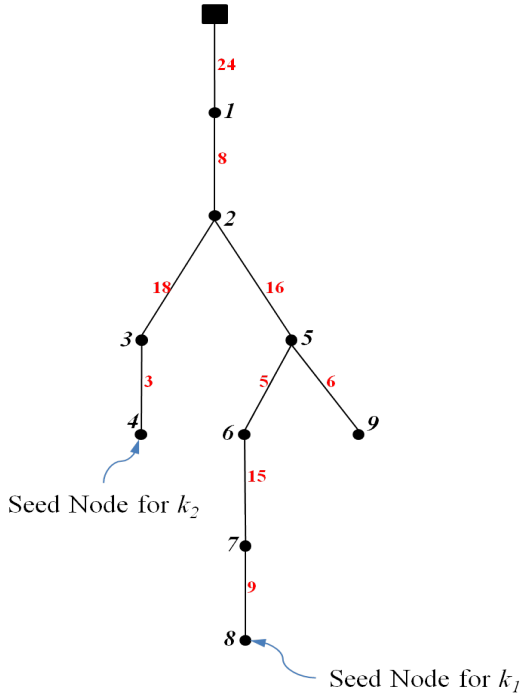
The heuristic developed in this chapter can be applied to homogeneous capacitated TVRPs and to capacitated TVRPs with *free* vehicles. Furthermore, the heuristic is also applicable when the arc costs are asymmetric and vehicle dependent.

The heuristic is explained by means of an example shown in Figure 5.2. The original network is shown in the left. The arc costs are shown on the edges (in red), and the demand at every node is shown in a green box next to the node. It is assumed that 3 vehicles with capacities 900, 600, and 300 are given. The first step, (I), involves assigning every vehicle to a seed node in T. The vehicle k_1 is assigned to node 8 and k_2 is assigned to node 4. Note that vehicle 3 remains unused. After the seed nodes have been assigned, δ_{ik} is calculated for the assigned vehicles (II). The GAP is then solved (III) using δ_{ik}, d_i and Cap_k as parameters. In the GAP solution, k_1 serves nodes 5, 6, 7, 8, and 9; and k_2 serves nodes 1, 2, 3, and 4. The solution cost is 272. The solution cannot be improved any further, hence, step (IV) is empty. The optimal solution to this problem is 272 too and contains the same assignment of nodes.



Vehicle	Capacity
k_1	900
k_2	600
k_3	300

(I) Find seed nodes for the Vehicles



(II) Compute δ_{ik}

N \ K	k_1	k_2
1	0	0
2	0	0
3	36	0
4	42	0
5	0	32
6	0	42
7	0	72
8	0	90
9	12	44

(III) Solve GAP and compute Cost

Vehicle	Nodes
k_1	5, 6, 7, 8, 9
k_2	1, 2, 3, 4
k_3	Unused

Cost	272

(IV) Check if solution can be improved

Figure 5.2: An Illustrative Example showing the steps in the Heuristic

5.5 Test Instances

The solution quality and computational efficiency of the proposed algorithm were tested on randomly generated test networks. The test networks were generated using a procedure similar to the one described by Labbé et al. (1991). Every node in the tree had between 1 to 5 children. Without loss of generality, the depot was forced to have a degree of one. 20, 40, 70 and 120–node networks were generated.

5.5.1 Parameter Generation

Each test network was tested for five demand profiles uniformly distributed between [1 10], [1 30], [1 100], [20 20] and [20 80]. The minimum vehicle capacity for each instance was randomly generated to be 1 to 2 times the maximum vehicle demand. The number of vehicles $|K|$ was the upper bound on number of vehicles required to serve the network when all vehicles have minimum capacity. This upper bound can either be a bin–packing upper bound or a TCVRP upper bound suggested by Chandran and Raghavan (2008). 0 to 10% of the vehicles in $|K|$ were assumed to be four times the minimum capacity, 0 to 20% thrice the vehicle capacity and 0 to 20% twice the vehicle capacity. The rest of the vehicles were assumed to be of minimum capacity. The fixed costs were assumed to be 0. The chapter studies the effectiveness of the seed node selection, contraction and refining operations which are not dependant on vehicle fixed costs. The method described here tries to ensure that the fleet size generated remain constaint over different demand

distributions for a particular node–size, this is done so that the fleet size does not affect computational performance. As a result, a direct comparison of the results here and the results presented in the other chapters cannot be directly compared. The fleet size is treated as a parameter here, whereas in the other chapters it is treated as a variable that affects the solution quality and computation time.

5.6 Computational Results

For every demand profile and for every node size, 10 instances were solved as suggested by Labbé et al. (1991). Thus, in total, the heuristic was tested on 200 networks for its solution quality and computational efficiency. The exact formulation, along with the valid inequalities, was solved using the CPLEX solver on a PC with a 32–bit architecture, 4GB RAM, and 2.93 GHz processor. The iteration limit for branch and bound was set to 100000, the time limit was set to 1000 seconds, and the optimality criterion for the MIP (relative gap between the best found integer solution value and best found LP value) was set to 0.0001. The time limit is only set to 1000 because the optimal solution method is not being tested here, moreover investigating how the IP performs after 1000 seconds gives an insight into the ability of the heuristic to generate good quality solutions within a fraction of the time. A summary of the computational results is presented in Table 5.1. The summary contains the following information:

Solved The number of instances (out of 10) that were solved to optimality

within 1000 seconds.

TimeOPT The time taken by CPLEX to solve the MIP formulation to optimality (reported only for solved instances).

TimeH1 and TimeH2 The time taken to solve the heuristic (using $\texttt{VehicleAssign}_{H1}$ and $\texttt{VehicleAssign}_{H2}$) respectively. It is the sum of times required to solve `VehicleAssign`, `Contract`, Generalized Assignment Problem and Refinement algorithms.

GapH1 and GapH2 This is the gap between the heuristic algorithm and the optimal solution (z^*). This gap is given by $\frac{\bar{z}-z^*}{z^*}$. It conveys how far off the algorithm solution value was from the optimal solution value (reported only for solved instances of the MIP).

First GapH1 and First GapH2 This is the gap between the heuristic algorithm solution value and the first best integer solution found by the CPLEX solver (z_F). It is computed as $\frac{\bar{z}-z_F}{\bar{z}}$. It is reported only for unsolved instances.

Last GapH1 and Last GapH2 This is the gap between the heuristic algorithm solution value and the last best integer solution found by the CPLEX solver (z_L). It is computed as $\frac{\bar{z}-z_L}{\bar{z}}$. It is reported only for unsolved instances.

5.6.1 Solution Quality

Table 5.1 shows that for solved instances, the gap varied between 2% to 8% for heuristic $H1$ and between 3% and 11% for heuristic H2. On an average, $H1$ seems to perform better than $H2$. Also, as problem size increased, the gap did not increase substantially for either $H1$ or $H2$. This is because the seed node selection depended on the network geometry and not on the network size. Moreover, δ_{ik} was computed based on tree geometry and not on network size. The demand distribution [1 10] was the easiest to solve for the heuristic and for CPLEX, while the networks with demand [20 20] were the hardest to solve. Only a few 70–node instances could not be solved to optimality, whereas all 120-node instances remained unsolved after 1000 seconds. This is because the method used to generate $|K|$ results in a very high $|K|$ value for larger problems thereby increasing the number of variables that need to be branched. The First Gap reported for the unsolved instances were atleast 53% lower than the best initial integer solution found by CPLEX. Therefore, the heuristic can be used to initialize the CPLEX solver very effectively. Finally, the Last Gap results suggest that the heuristic solutions were very close to the best last integer solution found by CPLEX at termination. In fact, for 120–node networks with [20 20] demand, the heuristic solution was better than the last CPLEX solution. This reaffirms that for tougher problems, the heuristic can be used to find very good solutions.

5.6.2 Computational Performance

As expected, the CPLEX solution time increases with problem size. For 120–node networks, CPLEX was unable to find a solution within 1000 seconds. On the other hand, the heuristic took much lesser time to find good solutions that were well within 10% of the optimal solution for most cases. $H1$ perfomed marginally better than $H2$ as $H2$ required one extra step compared to $H1$. It is interesting to note that the maximum reported time for solving the heuristic without applying the refining operation was only about 5 seconds. The refinement operation performs $|N|^2|K|$ iterations for every network. Therefore, due to their high $|K|$ value, the time take for instances with [20 20] and [20 80] demand was higher than the time taken for other demand profiles.

Network	Demand	Solved	Nodes	GapH1	GapH2	TimeOPT secs	TimeH1 secs	TimeH2 secs	First GapH1	First GapH2	Last GapH1	Last GapH2
20–node	[1 10]	10	398	0.027	0.029	1.35	0.52	0.51	–	–	–	–
	[1 30]	10	202	0.037	0.053	0.95	0.31	0.33	–	–	–	–
	[1 100]	10	339	0.050	0.051	1.67	0.57	0.61	–	–	–	–
	[20 20]	10	3790	0.027	0.064	2.93	1.23	1.23	–	–	–	–
	[20 80]	10	5625	0.026	0.059	5.25	0.89	0.93	–	–	–	–
40–node	[1 10]	10	4054	0.033	0.063	5.88	2.13	2.24	–	–	–	–
	[1 30]	10	1259	0.056	0.066	4.98	2.46	2.53	–	–	–	–
	[1 100]	10	1047	0.070	0.056	44.49	2.35	2.97	–	–	–	–
	[20 20]	10	3764	0.026	0.099	32.72	7.82	8.10	–	–	–	–
	[20 80]	10	3256	0.073	0.078	15.74	4.22	4.93	–	–	–	–
70–node	[1 10]	10	9244	0.079	0.103	195.68	11.86	12.54	–	–	–	–
	[1 30]	8	30005	0.070	0.082	358.5	12.42	13.24	-0.65	-0.58	0.047	0.044
	[1 100]	10	9722	0.083	0.113	133.9	7.45	7.79	–	–	–	–
	[20 20]	0	19028	–	–	–	48.65	53.57	-.718	-.564	0.056	0.133
	[20 80]	5	33719	0.085	0.087	585.8	39.65	40.67	-0.525	-0.488	0.133	0.156
120–node	[1 10]	0	10618	–	–	–	28.45	29.35	-0.574	-0.566	0.104	0.102
	[1 30]	0	30005	–	–	–	32.46	32.36	-0.589	-0.597	0.123	0.118
	[1 100]	0	12318	–	–	–	30.23	30.30	-0.749	-0.694	0.081	0.112
	[20 20]	0	2429	–	–	–	95.67	98.54	-743	-.615	-0.045	0.029
	[20 80]	0	9327	–	–	–	79.94	80.59	-0.614	-0.613	0.091	0.091

Table 5.1: Summary of Numerical Results

5.7 Conclusions

In this chapter a special case of the VRP – where the network is a tree and the vehicle fleet is fixed and heterogeneous was studied. Because of this heterogeneity, existing heuristic solution methods cannot be used to solve the problem.

The HTCVRP's relation to the Generalized Assignment Problem and the Capacitated Facility Location Problem was discussed. It was shown that when the nodes were ordered in DFS order, the existing IP formulation can be modified to have only GAP constraints. A linear approximation to this modified formulation was presented. A heuristic was then developed to use this GAP formulation to solve the HTCVRP. The heuristic iteratively finds seed nodes using Upper Bound (UB) heuristics for the BPP, assigns a vehicle to each seed node, then uses a Lagrangian–based GAP algorithm to assign nodes to vehicles located at the seed node. Two methods for finding seed nodes were also presented.

The heuristic was tested on 200 test networks of varying sizes. It was found that the heuristic performs consistently well irrespective of problem size with solutions ranging from 2 – 10% of the optimal solution value, while taking much lesser time than the CPLEX solver. It was also shown that significant improvements in optimal solution time can be achieved if the heuristic solution was used to initialize the CPLEX solver.

Future research includes exploring other variants of the problem, like

the TVRP with Time–Windows, and stochastic and online versions of TCVRPs. Exploring other heuristic solution techniques that explicitly take advantage of the tree structure must also be explored.

Chapter 6

TVRP with Time–related Constraints

In this chapter, new heuristics for two variants of the TVRP – vehicle time or duration constraints and customer time–window constraints – are proposed. Exact and heuristic methods and other details of the Duration Constrained Capacitated Vehicle Routing Problem on Trees (DTCVRP) are described in Sections 6.1 – 6.5, while Sections 6.6 – 6.8 deal with the Capacitated Vehicle Routing Problem on Trees with Time–Windows (TCVRPTW).

6.1 Problem Definition – DTCVRP

The Duration Constrained Capacitated TVRP is concerned with finding optimal time minimizing vehicle routes that serve customers that are located on a tree network. Each vehicle is specified a tour time limit which cannot be exceeded.

The problem is defined as follows. Given a tree network $T = (N_D, A)$ with customer nodes $i \in N$ and arcs $(i, j) \in A$ between the nodes with traversal times t_{ij}; demand at each node d_i; service time at each node s_i; and a set of vehicles $k \in K$ with capacity Cap and tour duration limit Q – find a collection of time–minimizing routes starting and ending at the depot, such that

(i) the customer's demand/supply at a node is satisfied by exactly one vehicle,

(ii) vehicle capacity restrictions are adhered to,

(iii) no vehicle exceeds its time duration constraint

It is assumed here, as usual, that the arc costs and arc traversal times are equal, that is $c_{ij} = t_{ij}$ (Laporte et al., 1985, 1984). Sometimes, this problem is referred to as the distance constrained vehicle routing problem or the time constrained vehicle routing problem. Further, the objective is to minimize the total time. For simplicity, and to prevent any notational overload, it is assumed that L_i is the time taken to travel from the depot to node i.

The DTCVRP is discussed in the following three sections. A lower bound condition for the problem is defined in Section 6.2 and some exact methods for solving the DTCVRP are discussed Section 6.3. A Heuristic is for the problem is proposed in Section 6.4. Section 6.5 contains some results of the computer implementation of this problem.

6.2 Lower Bound – DTCVRP

The lower bound for the DTCVRP defined by Mbaraga et al. (1999) is documented below for completeness. This lower bound is further used in the IP formulation that is described later on. The number of vehicles entering any sub–tree S_i for a node $i \in N$ is constrained by the total demand that has to be served in that sub–tree and also the total time it takes to serve the sub–tree. Let V_i^{Cap} be the minimum number of vehicles required to serve sub–tree

S_i when only the demands in the sub–tree are considered. Then, as per the discussion in chapters 3 and 5, it can be stated that:

$$V_i^{Cap} = \left\lceil \frac{\sum_{j \in S_i} d_j}{Cap} \right\rceil$$

Now, consider the duration constraints. The amount of time a vehicle spends serving a node i is:

$$r_i = s_i + t_{P_i i} + t_{i P_i}$$

That is, the sum of service time at node i and the time required to reach to that node from its parent and the time required to exit from that node and move towards its parent. This definition is possible because, in trees, every vehicle will enter a sub-tree only once for service. So, r_i is a duration constraint equivalent of the demand at every node.

Notice that, a vehicle $k \in K$ can spend upto $Q - 2L_{P_i}$ amount of time serving the nodes in S_i. Therefore, for every node i, the residual time available for serving nodes below it can be defined by:

$$q_i = Q - 2L_{P_i}$$

This can be interpreted as a duration constraint equivalent of the capacity restriction for every vehicle. Let V_i^D be the minimum number of vehicles required to serve sub–tree S_i. Then, if q_i is the capacity at i and r_i is the demand at i,

$$V_i^D = \left\lceil \frac{\sum_{j \in S_i} r_j}{q_i} \right\rceil$$

The minimum number of vehicles that serve a sub–tree S_i is then given by:

$$V_i = \max\left\{V_i^{Cap}, V_i^D\right\}$$

Proposition 6.1. *A lower bound, $\underline{z}$, on the optimal objective value, z^*, to the DTCVRP is given by:*

$$\underline{z} = 2\sum_{i \in N} V_i t_{\{P_i,i\}} \tag{6.1}$$

Proof. As described above, the minimum number of vehicles requires to serve S_i is V_i. Therefore, at least V_i vehicles will enter the sub–tree S_i to serve its nodes. This implies that at least V_i vehicles will traverse the edge $\{P_i, i\}$. These vehicles will traverse the arc $\{P_i, i\}$ once on their way from the depot to the sub–tree, and once on their way back (after service) from the sub–tree to the depot. The result follows. □

6.3 Exact Methods – DTCVRP

This section describes the exact methods for solving DTCVRPs and suggests an IP formulation for the same.

6.3.1 Branch–and–Bound and Column Generation

Mbaraga et al. (1999) used a branch–and–bound scheme for the DTVRP and the DTCVRP. The scheme proceeds in a breadth first order. If the upper bound solution value $\bar{z}$ equals the lower bound value $\underline{z}$, then the scheme terminates as it has found the optimal solution. Otherwise, at every iteration,

a branch is created only if the distance constraints are not violated. These are trivially checked as the total distance traveled by each vehicle is twice the length of tree with only the nodes in that vehicle considered. A new vehicle route is created for every node with capacity between $\frac{Cap}{2}$ and Cap. These nodes are defined to be I^1. Now, at an arbitrary iteration, x, an unassigned node, i, farthest from the depot is chosen. Branches are created from x for every vehicle route and a new route with only i in it, provided capacity constraints are satisfied. A lower bound is computed for every potential node p. If $\underline{z}^p \geq \bar{z}$, then p is not created. Otherwise, $I^p = I^r \cup \{k\}$. Thus, this procedure implicitly considers every node assignment to every vehicle, thereby exhausting all possibilities.

Mbaraga et al. (1999) also define a set covering–based formulation for the DTCVRP, which is solved using column generation. The master problem of the column generation scheme solves the CDTVRP for a subset of variables, these variables are then parsed to the subproblem. The subproblem of the CDTVRP is a capacitated and distance constrained shortest path problem, whose arc costs are defined such that the shortest path will generate a column with the most negative reduced cost, which will in turn be parsed to the master problem. The tree arcs and costs are modified to form an acyclic graph for every vehicle. The constrained shortest path (SP) is then solved on these acyclic graphs using the algorithm proposed by Desrosiers et al. (1995). The SP algorithm has a pseudopolynomial running time.

For smaller demands, bin–packing bounds are sharper and have smaller

search trees, and thus, branch–and–bound is more efficient than column generation. However, the column generation scheme is much more robust in handling heterogeneous vehicles. The reader is referred to paper by Mbaraga et al. (1999) for the details of the branch–and–bound and column generation techniques.

6.3.2 IP–Formulation

The IP–formulation for the DTCVRP is similar to the TCVRP formulation described in Chapter 5. In addition to those constraints, a duration constraint must be added for every vehicle. The variables in the formulation are described as follows:

Variables

$$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ travels on arc } (i,j) \in A \\ 0 & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1 & \text{if vehicle } k \text{ serves node } i \in N \\ 0 & \text{otherwise} \end{cases}$$

Firstly, the total demand serviced by a vehicle k cannot exceed its capacity. That is,

$$\sum_{i \in N : d_i > 0} y_{ik} d_i \leq Cap \qquad \forall k \in K$$

The next constraint imposes that every node is serviced by exactly one vehicle.

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N : d_i > 0$$

Next, movement–related constraints need to be defined. Since the given network is a tree, a vehicle k serves a node i by traversing along the arc connecting the parent of the node i to i. This is stated as follows:

$$y_{ik} \leq x_{P_i ik} \qquad \forall i \in N : d_i > 0, \forall k \in K$$

Next, it is essential to ensure that every vehicle begins its tour at the depot. A constraint that ensures that a vehicle never moves from a node to its child unless the arc from that node's parent to that node has been traversed will impose this requirement. So, if a vehicle k serves node $i \in N : d_i > 0$, this constraint enforces that all the arcs on the path that leads from the depot to this node i are traversed by k.

$$x_{P_i ik} \geq x_{ijk} \qquad \forall i \in N : d_i > 0, \forall j \in c(i), \forall k \in K$$

Finally, every vehicle k's route cannot exceed time Q. Given that the network is a tree, and given the nodes a vehicle is going to serve, the arcs traversed by that vehicle is known. Also, it is known that a vehicle spends s_i amount of time at every node i that it decides to serve. Therefore,

$$2 \sum_{i \in N} \sum_{j \in N} t_{ij} x_{ijk} + \sum_{i \in N} s_i y_{ik} \leq Q \qquad \forall k \in K$$

For completeness, the formulation is stated as follows:

Formulation: **DTCVRP**

Minimize:
$$z^* = 2\sum_i \sum_j \sum_k t_{ij} x_{ijk} \tag{6.2}$$

Subject to:
$$\sum_{i \in N: d_i > 0} y_{ik} d_i \leq Cap \qquad \forall k \in K \tag{6.3}$$
$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in N : d_i > 0 \tag{6.4}$$
$$y_{ik} \leq x_{P_i ik} \qquad \forall i \in N : d_i > 0, \forall k \in K \tag{6.5}$$
$$x_{P_i ik} \geq x_{ijk} \qquad \forall i \in N : d_i > 0, \forall j \in c(i), \forall k \in K \tag{6.6}$$
$$2\sum_{i \in N} \sum_{j \in N} t_{ij} x_{ijk} + \sum_{i \in N} s_i y_{ik} \leq Q \qquad \forall k \in K \tag{6.7}$$
$$x_{ijk} \in \{0, 1\} \qquad \forall (i, j) \in A, \forall k \in K \tag{6.8}$$
$$y_{ik} \in \{0, 1\} \qquad \forall k \in K, \forall i \in N : d_i > 0 \tag{6.9}$$

The total number of constraints and variables in the IP are the same as that for the IP used to solve the TCVRP plus the constraint 6.7, which is $O(|K|)$.

Adaptability of the Formulation

The formulation can be adapted as follows to solve the following variants of the DTCVRP:

(i) Minimize Number of Vehicles Used: Firstly, the total number of available vehicles is kept free. That is, $|K| = |N| - 1$. Then, the objective function

is changed to:

$$z^* = \sum_{j \in c(depot)} \sum_{k \in K} x_{depotjk}$$

Due to the assumption that, in T, the depot has only one child, simply calculating the total number of vehicles traversing the arc from the depot to its child will record the total number of vehicles used by the formulation.

(ii) Vehicle Fleet is Heterogeneous: In this case, vehicles of different capacities are employed. Modifying the parameter Cap to $Cap_k \forall k \in K$ and the duration limit for each vehicle to Q_k.

(iii) Vehicle Dependant Arc Costs: The arc costs are no longer $c_{ij} \forall (i,j) \in A$ but $c_{ijk} \forall (i,j) \in A, \forall k \in K$. Changing the arc costs to c_{ijk} wherever they appear in the formulation will account for these vehicle dependent arc costs.

(iv) Asymmetric Arc Costs or Traversal Times: The formulation assumes that the arc travel times are symmetric. This assumption can be relaxed by adding a flow conservation constraint as follows:

$$\sum_{j \in \{P_i \cup c(i)\}} x_{ijk} = \sum_{j \in \{P_i \cup c(i)\}} x_{jik} \qquad \forall i \in N, \forall k \in K$$

Valid Inequalities

The symmetry breaking inequalities, as defined previously, are added to the formulation. Additionally, the knapsack–like constraints are also added to the formulation. The first constraint states that if the sum of the demands of two nodes exceeds the vehicle capacity, then only one of these nodes is served by the vehicle. The second one states that if the time required to serve two nodes exceeds the vehicle time limit, then only one of those two nodes will be served by the vehicle.

In Section 6.2, V_i was described to be the minimum number of vehicles required to serve the sub–tree $S_i : i \in N$. Therefore, in the tree, the arc (P_i, i) will be traversed at least V_i times, or at least V_i vehicles will use arc (P_i, i).

$$\sum_{k \in K} x_{P_i ik} \geq V_i \qquad \forall i \in N \qquad (6.10)$$

6.4 Heuristic – DTCVRP

The heuristic described here is a modification of the Savings Algorithm first developed by Clarke and Wright (1964a). Initially, every node is assigned its own vehicle. Two vehicles are then merged together if the merging is feasible and results in a time saving. Now, consider two vehicles located at nodes i and j. Initially, the total time for routing these two vehicles is $2(L_i + L_j)$. That is, the total time is equal to the time spent in traveling from the depot to i and returning back to the depot, and the time spent in traveling from the depot to j and returning back to the depot. Assume that the vehicles at i and

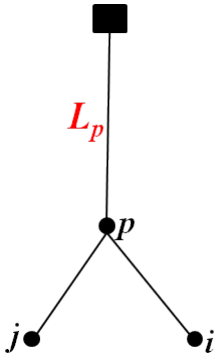


Figure 6.1: Maximum Savings Obtained by Merging Vehicles at i and j

j can be merged together into a single vehicle. Then the total time savings obtained from this merging is equal to twice the time required to travel from the depot to the last node that is common to both the paths i and j from the depot. More formally, if Sav_{ij} is the savings obtained from merging vehicle i and j, then:

$$Sav_{ij} = 2 \max_{p=\{PfD_i \cap PfD_j\}} L_p$$

Recall that PfD_i is the set of nodes in the unique path from the depot to node i. Figure 6.1 illustrates the savings that can be obtained by merging two vehicles.

In a nutshell, the heuristic proceeds as follows. First, each node is assigned its own vehicle. Therefore, initially, there are $|N| - 1$ vehicles. It is also assumed that after serving that node, the vehicle is *parked* at that node. At this stage the duration of the vehicle route that is parked at i is simply $2L_i$. In the next step, or, for that matter, at any intermediate step, the vehicle that is parked farthest from the depot is chosen for merging. If there is a tie, the

vehicle with the maximum sum of current demand served and route duration is chosen. Two vehicles can only be merged if they are both parked at the same node. Three cases exist:

Case 1 If there are no two vehicles parked at the same location, then the vehicle k that was parked at j is now moved from j and parked at P_j. The total demand and nodes served by the vehicle still remains the same, so does the route duration since it is just moving to the parent of the current node.

Case 1 If there are exactly two such vehicles, then these two vehicles are selected for merging. Assume that k_1 and k_2 are candidates for merging and they both are parked at some node j. Then, if the route duration and capacity constraints permit merging, these two vehicle routes are merged. A new vehicle route that contains the nodes served by both these vehicles is created and parked at node j. If the vehicles cannot be merged together, then the first vehicle that was selected is moved to its parent node.

Case 3 More than two vehicles are parked at the same node j. In this case, the vehicles are chosen in decreasing order of the sum of their current demand served and route duration. Each vehicle is tried for merging with another. If a vehicle cannot be merged, it is moved from its current node and parked at the parent of that node.

At termination, every vehicle will be parked at the depot. Another point of note is that at any iteration, all the nodes that are *below* the nodes where the vehicles are parked would have been served.

In Algorithm 6.1, which formally describes the heuristic, the following notation are used. Let $\texttt{park}_k$ denote the current location of the vehicle k. Let q_k denote the time required to serve all the nodes currently in k starting at $\texttt{park}_k$. Let d_k denote the total demand served by the vehicle k. Let $\texttt{served}(k)$ be the set of nodes that have been served by vehicle k. For notational convenience, assume that every time vehicles are merged, a new vehicle is created and the merged vehicles are discarded. That is, if the heuristic started with veh number of vehicles, then the merged vehicles will be $veh+1$, $veh+2$, ...

In the Algorithm 6.1, at any iteration, the variable `CurrentVehicle` keeps track of the vehicle which has been considered for merging. As seen in lines 12 – 24, on merging with v, both `CurrentVehicle` and v are removed from the set K and vehicle veh which contains nodes served by both these vehicles is added to K. veh is now considered for merging and that explains the variable `MergeCandidate` which keeps track of the vehicle to be merged if multiple vehicles are parked at the same node.

The set `CandidateVehicles` stores all the vehicles that are farthest from the depot and have been parked at the same node. The vehicles in this set are arranged in decreasing order of $q_k + d_k$. If this set contains only one vehicle, no merging is possible and the algorithm will not enter the loop from lines 13 to 24. The vehicle is then moved to the parent node, this is detailed

Algorithm 6.1 Heuristic for DTCVRP

Input: $T = (N, A),\ d_i \forall i \in N,\ t_{ij} \forall (i,j) \in A,\ Cap,\ Q$
Output: Upper Bound DTCVRP Cost, $\bar{z}$

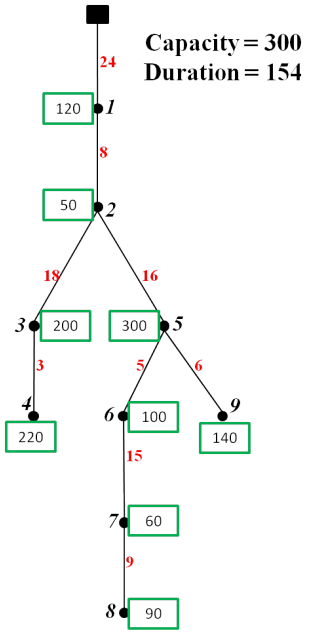
Set $K \leftarrow \{1, 2, \ldots, |N| - 1\}$
$\texttt{park}_k \leftarrow k\ \forall k \in K$
$d_k \leftarrow d_{\texttt{park}_k}\ \forall k \in K$
$q_k \leftarrow s_i + 2t_{\texttt{park}_k P_{\texttt{park}_k}}\ \forall k \in K$
$\texttt{served}(k) \leftarrow k$
$veh \leftarrow |K|$
while $\exists \texttt{park}_k \neq depot\ \forall k \in K$ **do**
  $\texttt{CandidateVehicles} \leftarrow \text{argmax}_{k \in K}\, L_{\texttt{park}_k}$
  SORT(CandidateVehicles) in non–increasing order of $q_k + d_k$
  $\texttt{CurrentVehicle} \leftarrow$ First Element of CandidateVehicle
  $\texttt{CurrentNode} \leftarrow \texttt{park}_{\texttt{CurrentVehicle}}$
  $\texttt{MergeCandidate} \leftarrow \texttt{CurrentVehicle}$
  for $iter = 2 \rightarrow |\texttt{CandidateVehicles}|$ **do**
    $v \leftarrow \texttt{CandidateVehicles}_{iter}$
    if $(q_{\texttt{MergeCandidate}} + q_v - 2t_{\texttt{CurrentNode}P_{\texttt{CurrentNode}}} - 2L_{P_{\texttt{CurrentNode}}} < Q + 1)(d_{\texttt{MergeCandidate}} + d_v < Cap + 1)$ **then**
      $veh \leftarrow veh + 1$
      $K \leftarrow \{K \cup veh \setminus \texttt{MergeCandidate} \setminus v\}$
      $q_{veh} \leftarrow q_{\texttt{MergeCandidate}} + q_v - 2t_{\texttt{CurrentNode}P_{\texttt{CurrentNode}}}$
      $d_{veh} \leftarrow d_{\texttt{MergeCandidate}} + d_v$
      $\texttt{park}_{veh} \leftarrow \texttt{CurrentNode}$
      $\texttt{served}(veh) \leftarrow \{\texttt{served}(\texttt{MergeCandidate}) \cup \texttt{served}(v)\}$
      $\texttt{MergeCandidate} \leftarrow veh$
    end if
  end for
  if $\texttt{CurrentVehicle} \in K$ **then**
    $\texttt{park}_{\texttt{CurrentVehicle}} \leftarrow P_{\texttt{CurrentNode}}$
    $q_{\texttt{CurrentVehicle}} \leftarrow q_{\texttt{CurrentVehicle}} + 2t_{\texttt{CurrentNode}P_{\texttt{CurrentNode}}}$
  end if
end while
$\bar{z} = \sum_{k \in K} q_k - \sum_{i \in N} s_i$

in lines 25 – 28. Finally, the upper bound can be found by just adding the q_k values of all the vehicles at termination. As, at termination, every vehicle is at the depot. The set `serve`(k) will contain the nodes served by each vehicle at termination.

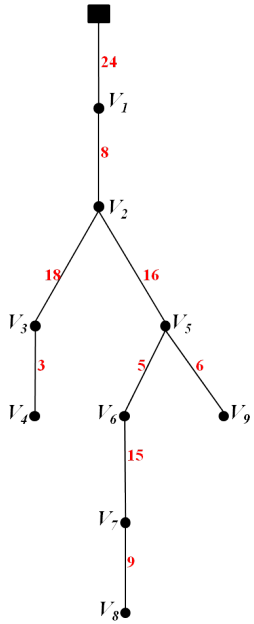
The Heuristic is explained using Figures 6.2 – 6.4. The original network is given in Figure 6.2(a). The arc costs are shown on the edges (in red), and the demand at every node is shown in a green box next to the node. The vehicle capacity is assumed to be 300 and the time constraint for each vehicle is assumed to be equal to twice the maximum distance from the depot, 154 here. Service times are assumed to be zero. The various iterations of the heuristic are shown in Figures 6.2 – 6.4. The current location of a vehicle, the accumulated duration and demand, and the nodes it has served is noted in the table below each tree. The column names in these tables are the same as the notation used to describe the heuristic.

Initially every node is assigned its own vehicle as shown in Figure 6.2(b), next the vehicle that is farthest from the depot is chosen, V_8 here, and moved to the parent node. As shown in Figure 6.2(c), vehicles 7 and 8 can be merged together without violating any constraints, the new vehicle, 10, is created after merging. The vehicle numbers are changed after every merging to prevent any confusion. `park`, `served`, q, and d for vehicle 10 are modified accordingly. The procedure continues in this manner. Consider the iteration shown in Figure 6.3(a), at this stage, V_4 or V_{11} can be chosen, however, the $d + q$ value of V_{11} is greater, and it is the vehicle that is chosen by the heuristic. Similarly,

at the third to last iteration, Figure 6.4(b), either V_{11} or V_9 can be merged with V_2, however, V_{11} has a greater $q + d$ value, and it is chosen to be merged with V_2. The upper bound value can be calculated by just summing the q values at the final iteration.

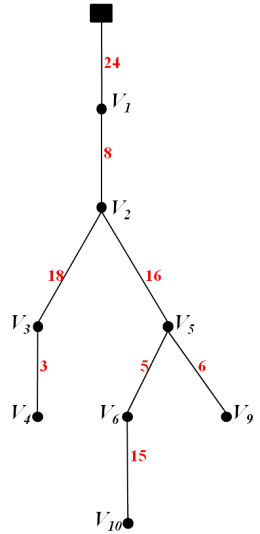


(a)



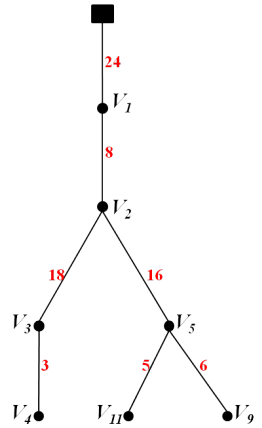
Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	50	3	3	36	200
4	53	4	4	6	220
5	48	5	5	32	300
6	53	6	6	10	100
7	68	7	7	30	60
8	77	8	8	18	90
9	54	9	9	12	140

(b)



Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	50	3	3	36	200
4	53	4	4	6	220
5	48	5	5	32	300
6	53	6	6	10	100
9	54	9	9	12	140
10	68	7,8	7	48	150

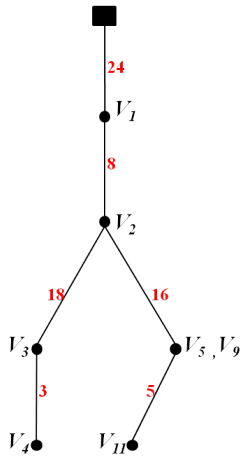
(c)



Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	50	3	3	36	200
4	53	4	4	6	220
5	48	5	5	32	300
9	54	9	9	12	140
11	53	6,7,8	6	58	250

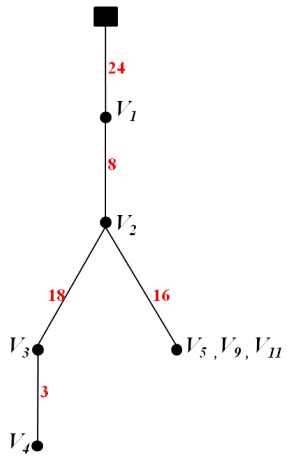
(d)

Figure 6.2: An Illustrative Example for the DTCVRP – 1



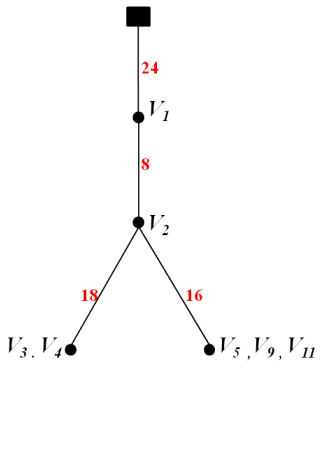
Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	50	3	3	36	200
4	53	4	4	6	220
5	48	5	5	32	300
9	48	9	5	44	140
11	53	6,7,8	6	58	250

(a)



Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	50	3	3	36	200
4	53	4	4	6	220
5	48	5	5	32	300
9	48	9	5	44	140
11	48	6,7,8	5	90	250

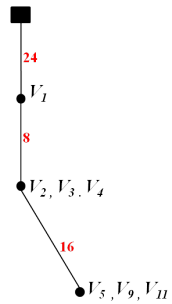
(b)



Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	50	3	3	36	200
4	50	4	3	42	220
5	48	5	5	32	300
9	48	9	5	44	140
11	48	6,7,8	5	90	250

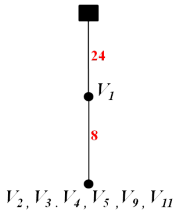
(c)

Figure 6.3: An Illustrative Example for the DTCVRP – 2



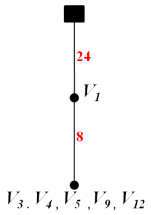
Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	32	3	2	52	200
4	32	4	2	58	220
5	48	5	5	32	300
9	48	9	5	44	140
11	48	6,7,8	5	90	250

(a)



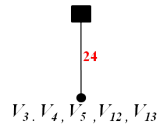
Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
2	32	2	2	16	50
3	32	3	2	52	200
4	32	4	2	58	220
5	32	5	2	48	300
9	32	9	2	60	140
11	32	6,7,8	2	106	250

(b)



Vehicle	Depot Distance	Served	Park	q	d
1	24	1	1	48	120
3	32	3	2	52	200
4	32	4	2	58	220
5	32	5	2	48	300
9	32	9	2	60	140
12	32	2,6,7,8	2	106	300

(c)



Vehicle	Depot Distance	Served	Park	q	d
3	24	3	1	100	200
4	24	4	1	106	220
5	24	5	1	96	300
13	24	1,9	1	108	260
12	24	2,6,7,8	1	154	300

(d)

Figure 6.4: An Illustrative Example for the DTCVRP – 3

6.5 Computational Results – DTCVRP

The heuristic and the formulation were tested on 50–node tree networks. This section describes the experimental setup and the computational results of this implementation. The test networks were generated such that every node in the tree had between 1 to 5 children. Without loss of generality, the depot was forced to have a degree of one.

Arc Times The arc times can affect the computational results as these are directly related to the duration constraints imposed on each vehicle. Two arc time profiles were tested – times uniformly distributed between [1 100] and [20 80].

Demand Each network was tested for five demand profiles uniformly distributed between [1 100], [10 90], [20 80], [40 60] and [50 50].

Capacity The vehicle capacity was set to 100.

Number of Vehicles The formulation requires the total number of vehicles $|K|$. This was set to be equal to the total number of vehicles that were used by the heuristic. The IP formulation does not impose that all the vehicles must be used. It works within the given $|K|$ and uses the optimal number of vehicles that will minimize the total cost. The number of vehicles are required because of the vehicle–index nature of the formulation. Providing the IP with the number of vehicles used by the heuristic also helps better compare the optimal solution value to the heuristic value.

Duration Constraints Two different duration constraints were tested. Since it is assumed that every node can be feasibly served, the first duration constraint was equal to twice the maximum time between the depot to any node, that is, $Q = 2 \max_{i \in N} L_i$. The second duration constraint tested was $Q = 2 \max_{i \in N} L_i + 50$. Service times were zero.

Since there are many parameters which can affect the solution and computational quality, the mean over which the demand was distributed was kept constant. All tests were performed on randomly generated 50–node networks. For every demand profile, for every arc time distribution and for every duration constraint, 10 instances were solved. That is, in total, 200 different instances of the problem were solved.

The IP formulation, along with the valid inequalities, was solved using CPLEX on a PC with a 32–bit architecture, 2GB RAM, and 2.93 GHz processor. The iteration limit for branch and bound was set to 100000, the time limit was set to 3600 seconds, and the optimality criterion for the MIP (relative gap between the best found integer solution value and best found LP value) was set to 0.0001.

The order of branching the variables was set as follows. First, a variable that represents the total number of vehicles is defined as follows:

$$v = \sum_{j \in c(depot)} \sum_{k \in K} x_{depotjk} \tag{6.11}$$

The variable v is branched first. Next, the branching rule for the variables y_{ik} is set such that these variables are branched in increasing order of the vehicle

index. That is, the variable y_{ik_1} is branched before the variable y_{ik_2} for all $k_1 < k_2$.

CPLEX was warm–started using the heuristic solution. That is, the heuristic solution was fed into CPLEX as the first feasible heuristic solution. So that this initial solution does not violate the symmetry constraints, the nodes served by each vehicle in the heuristic were ordered such that the lowest indexed node served by a vehicle increased with vehicle index and that a node is only served by a vehicle whose index is less than the node index. A summary of the computational results is presented in Table 6.1. The summary contains the same information as Table 4.1.

6.5.1 Solution Quality and Computational Performance

In general, it can be noted that, for all travel time distributions and duration constraints, the IP performs better as the demand variance decreases. It can also be seen that problem instances with travel times in [20 80] perform better than the instances with travel times in [1 100]. Further, for a given travel time distribution, the problems with a lower time constraint perform better. This is intuitive because when the duration constraints are lower, more number of variables and constraints can be eliminated.

The LP–relaxation values are very tight for all the problem instances. In fact, they are all above 95%. The LP–relaxations are slightly tighter when the variance of the travel time distribution is lesser. For the [50 50] demand instances, the LP–relaxation values are close to 100%.

Vehicle Duration	Travel Time	Demand	Solved[α]	$\|K\|_{UB} - v$	TimeUB secs	TimeOPT secs	LP/IP	UB/IP	LB/UB	LP/LB	Nodes
$2\max_{i\in N} L_i$	[1 100]	[1 100]	10	1.6	0.081	419.640	0.975	1.078	0.874	1.036	5228.3
		[10 90]	10	2.0	0.085	216.610	0.950	1.076	0.849	1.041	3686.8
		[20 80]	10	1.8	0.088	202.888	0.956	1.060	0.894	1.011	3412.0
		[40 60]	10	1.0	0.079	74.244	0.959	1.050	0.900	1.016	945.6
		[50 50]	10	0.6	0.079	4.248	0.999	1.024	0.887	1.109	0.0
$2\max_{i\in N} L_i + 50$	[1 100]	[1 100]	10	2.6	0.081	466.318	0.970	1.098	0.861	1.027	10075.6
		[10 90]	10	1.6	0.082	274.176	0.977	1.087	0.892	1.016	6518.3
		[20 80]	10	3.0	0.084	94.511	0.996	1.079	0.899	1.020	4469.0
		[40 60]	10	1.0	0.075	37.284	0.990	1.034	0.949	1.016	2023.5
		[50 50]	10	0.8	0.078	3.887	0.997	1.029	0.945	1.022	58.4
$2\max_{i\in N} L_i$	[20 80]	[1 100]	10	3.0	0.079	158.316	0.992	1.092	0.858	1.013	3455.5
		[10 90]	10	3.3	0.081	31.330	0.985	1.092	0.835	1.035	1824.0
		[20 80]	10	2.0	0.077	26.372	0.992	1.088	0.840	1.048	1193.0
		[40 60]	10	1.0	0.082	16.555	0.988	1.078	0.924	1.028	733.5
		[50 50]	10	1.6	0.079	7.053	1.000	1.041	0.907	1.024	211.2
$2\max_{i\in N} L_i + 50$	[20 80]	[1 100]	10	2.5	0.097	310.957	0.972	1.082	0.897	1.028	8566.0
		[10 90]	10	2.0	0.080	149.015	0.989	1.073	0.877	1.024	6484.0
		[20 80]	10	2.0	0.084	116.168	0.984	1.058	0.849	1.032	5490.0
		[40 60]	10	1.2	0.078	71.651	0.971	1.046	0.907	1.024	1672.0
		[50 50]	10	1.0	0.079	3.179	0.997	1.034	0.965	1.000	47.8

[α] All reported values are averaged over the 10 tested instances of each problem

Table 6.1: Summary of Numerical Results for the DTCVRP

Another interesting aspect is that the lower bound values are very close to the LP–relaxation values. This might suggest that using a branch–and–bound algorithm (Mbaraga et al., 1999) can yield very good results.

Here too, the solution quality depends on the number of vehicles required by the heuristic. As the difference between the number of vehicles required by the heuristic and IP increases, the solution quality decreases. This difference is higher for higher demand variance. This is because the vehicle merging step in the heuristic described earlier is more effective for lower demand variance.

The heuristic performs reasonable well for all instances tested. The heuristic quality is consistently between 2 – 9% of the optimal solution. The heuristic seems to perform better when the demand variance is lower. The heuristic performs best for the problem instances with travel time in [1 100] and duration constraint equal to $2 \max_{i \in N} L_i$. The heuristic solution time averages around 0.08 seconds for almost all the tested instances. This suggests that the heuristic is extremely quick to compute a feasilbe solution.

The IP solution times, for each travel time distribution and duration constraint, increase with an increase in demand variability. As noted in earlier chapters, there is a direct relation between number of vehicles, solution quality and solution time. The solution times for the travel time distribution [20 80] are lower compared to the travel time distribution [1 100]. For a given travel time distribution, the problems with lower time constraints are solved to optimality faster.

6.6 Problem Definition – TCVRPTW

The Capacitated TVRP with Time Windows (TCVRPTW) is concerned with finding cost minimizing vehicle routes beginning and ending at the depot such that the total demand served by each vehicle does not exceed its capacity and a vehicle serves a customer only during the time window specified by the customer.

A formal definition of the problem is as follows. Given a tree network $T = (N_D, A)$ with customer nodes $i \in N$ and arcs $(i, j) \in A$ between the nodes with traversal times t_{ij}; demand at each node d_i; service time at each node s_i; service time windows $[a_i\ b_i]$ for every customer $i \in N$; and a set of vehicles $k \in K$ with capacity Cap – find a collection of time–minimizing routes starting and ending at the depot, such that

(i) the customer's demand/supply at a node is satisfied by exactly one vehicle,

(ii) vehicle capacity restrictions are adhered to, and

(iii) a vehicle serves a customer i only during the specified time window $[a_i\ b_i]$.

It is assumed that the arc costs and arc traversal times are equal, that is $c_{ij} = t_{ij}$. Further, it is also assumed that if a vehicle arrives at a node it is going to serve before the service time window, then it can wait at that till it can start servicing it. It is assumed that the customer time windows are such that a feasible solution to the problem exists.

6.6.1 Additional Notation

L_i is the time taken to travel from the depot to node i. Let L_{ij} be the time required to travel between any two nodes i and j in N. The time window associated with the depot, $[a_{depot}\ b_{depot}] = [AB]$, where A is the earliest departure time from the depot and B is the latest arrival time at the depot. Clearly, a feasible solution to the problem exists only if $A \leq \min_{i \in n}\{b_i - L_i\}$ and if $B \geq \min_{i \in N}\{a_i + s_i + L_i\}$. Let e_i denote the time at which service starts at a node and let l_i be the time at which service ends at a node. Let w_i denote the waiting at a node before service begins. Then, $w_i = \max\{0, a_i - e_i\}$. Consequently,

$$l_i = \begin{cases} e_i + s_i & \text{if } e_i - a_i > 0 \\ a_i + s_i & \text{if } e_i - a_i \leq 0 \end{cases}$$

The TCVRPTW is discussed in the following two sections. A Heuristic is for the problem is proposed in Section 6.7. Section 6.8 contains some results of the computer implementation of this problem.

6.7 Heuristic – TCVRPTW

The heuristic proposed in this section is a route building heuristic. Every iteration of the heuristic builds the route for a vehicle. First, a node unassigned to any vehicle is chosen according to some rule. Then, the feasibility of accommodating an unassigned node closest to this node is investigated. For each node that can be accommodated, the best position within the vehicle

route is noted. The best position obviously is the one which minimizes distance. From all these candidate nodes, finally, the node whose addition causes the least increase in the vehicle route cost is finally added to the route. This procedure continues till no new nodes can be added to the vehicle. In the next iteration a new vehicle route is initialized. The heuristic is a modification of the algorithm proposed by Solomon (1987).

In order to insert a node into a vehicle route, both capacity and time window constraints have to be checked. Ensuring capacity feasibility is straightforward. Let $\texttt{Route}_k$ denote the route of vehicle k. For this heuristic, assume that the depot is denoted by node 0. Let $\texttt{Route}_k = \{0, 1_k, 2_k, \ldots, |k|_k, 0\}$. That is , let 1_k be the first node that vehicle k visits and let $|k|_k$ be the last node that vehicle k visits before going back to the depot. Now, consider a node i that has to be inserted into the route right after the p^{th} node, p_k, that vehicle k serves. Obviously, before the insertion the time the vehicle arrives at the $(p+1)^{th}$ node is given by $e_{p+1_k} = e_{p_k} + s_{p_k} + L_{p_k p+1_k}$. However, after the insertion of node i into the route,

$$e^{ins}_{p+1_k} = e_i + s_i + L_{ip+1_k}$$

. Therefore, the amount of time by which service start time at $p+1_k$ gets pushed forward is by

$$\texttt{Push}_{p+1_k} = \max\{0, (e^{ins}_{p+1_k} - e_{p+1_k}) - w_{p+1_k}\}$$

. That is, the amount by which the service start time gets pushed forward by is equal to time at which the service starts after insertion minus the time at

which the service started initially minus the waiting time at that node before the insertion or 0, whichever is greater. For the nodes in positions $p{+}2, \ldots, |k|$, the amount by which the service start time gets pushed forward can be stated recursively as

$$\texttt{Push}_{r+1_k} = \max\{0, \texttt{Push}_{r_k} - w_{r+1_k}\} \quad \forall p+1 \leq r \leq |k|$$

. Thus, in order for an insertion i into $\texttt{Route}_k$ between nodes p_k and $p{+}1_k$ to be time feasible, two conditions, as noted by Solomon (1987), must be satisfied:

(i) $e_i \leq b_i$

(ii) $e_{r_k} + \texttt{Push}_{r_k} \leq b_{r_k} \quad \forall p+1 \leq r \leq |k|+1$

Now, consider an intermediate stage of the route construction, when node i has to be inserted into vehicle k. Let the increase in the vehicle route cost after inserting i after the p^{th} node in the route be $Cost(p_k, i, p{+}1_k)$. Then,

$$Cost(p_k, i, p{+}1_k) = \begin{cases} 0 \quad \text{if } i \text{ is on the path from } p_k \text{ to } p+1_k \\ \\ L_{p_k i} + L_{i p+1_k} - L_{p_k p+1_k} \quad \text{otherwise} \end{cases} \qquad 1 \leq p+1 \leq |k| \tag{6.12}$$

This cost is computed for all feasible locations within the route where the node i can be inserted. The best position $p(i)$ after which the node i is actually inserted is given by:

$$Cost(p_k(i), i, p{+}1_k(i)) = \min_{1 \leq p+1 \leq |k|} \{Cost(p_k, i, p{+}1_k) + (e^{ins}_{p+1_k} - e_{p+1_k})\} \tag{6.13}$$

Thus, the best insertion position does not depend only on the cost savings, but also on how much time slack there is in the $p+1^{th}$ node after insertion. Obviously, lesser the time slack the better as that tries to eliminate other feasible possibilities.

So, to recap, when a node i is chosen for insertion into vehicle k, first the increase in route cost after insertion of i into all feasible positions is calculated. From these, the best possible location for insertion of the node i is chosen. Therefore, $Cost(p_k, i, p+1_k)$ is calculated for every node that can be inserted and for every position that node can be inserted in and from this, $Cost(p_k(i), i, p+1_k(i))$ is calculated for every node that can be inserted feasibly into the route. Now, recall that at every step of an iteration, only one node is inserted into the vehicle cost. This node is chosen as follows:

$$i^* = \underset{i}{\text{argmax}}\{2L_i - Cost(p_k(i), i, p+1_k(i))\} \tag{6.14}$$

Different strategies can be used for selecting the first node in the vehicle route. Some strategies include selecting the farthest unassigned node, selecting a node with the earliest deadline etc. The strategy used here is to select the leaf node with the earliest deadline. A leaf node is chosen so that in the insertion step of the heuristic, the parent of the leaf node is chosen first, followed by the children of the parent and so on. The heuristic is described in Algorithm 6.2.

Every vehicle route can be further improved by performing an exchange operation. The 2–opt method is used here. This method has been detailed

Algorithm 6.2 Heuristic for TCVRPTW

Input: $T = (N, A),\ d_i \forall i \in N,\ t_{ij} \forall (i,j) \in A,\ Cap,\ Q, [a_i\ b_i] \forall i \in N$
Output: Upper Bound TCVRPTW Cost, $\bar{z}$

$LIST \leftarrow N$
$k \leftarrow 1$
while $LIST \neq \emptyset$ **do**
  $1_k \leftarrow \underset{i \in N: c(i) = \emptyset}{\text{argmin}} \{b_i\}$
  $LIST \leftarrow LIST \setminus i_k$
  $\texttt{Route}_k \leftarrow \{0, 1_k, 0\}$
  $\texttt{RouteCapacity}_k \leftarrow d_{1_k}$
  $\texttt{RouteCost}_k \leftarrow 2L_{1_k}$
  $e_{1_k} \leftarrow \max\{a_{1_k}, L_{1_k}\}$
  $w_{1_k} \leftarrow \max\{0, a_{1_k} - L_{1_k}\}$
  $FLAG \leftarrow 1$
  while $FLAG = 1 LIST \neq \emptyset$ **do**
    for $i = LIST_1 \rightarrow LIST_{|LIST|}$ **do**
      $FLAG \leftarrow 0$
      for $p + 1 = 2 \rightarrow \texttt{Route}_k$ **do**
        if $d_i + \texttt{RouteCapacity}_k < Cap$ **then**
          if $e_{r_k} + \texttt{Push}_{r_k} \leq b_{r_k} \quad \forall p + 1 \leq r \leq |k| + 1 e_i \leq b_i$ **then**
            $FLAG \leftarrow 1$
            $e_i \leftarrow \max\{a_i, e_{p_k} + s_{p_k} + L_{p_k i}\}$
            $w_i \leftarrow \max\{0, a_i - e_i\}$
            Calculate $Cost(p_k, i, p + 1_k)$ using Equation 6.12
          end if
        end if
      end for
    end for
    if $FLAG = 1$ **then**
      $Cost(p_k(i), i, p + 1_k(i)) = \min_{1 \leq p+1 \leq |k|}\{Cost(p_k, i, p + 1_k) + (e^{ins}_{p+1_k} - e_{p+1_k})\}$
      $i^* = \text{argmax}_i\{2L_i - Cost(p_k(i), i, p + 1_k(i))\}$
      $\texttt{RouteCapacity}_k \leftarrow \texttt{RouteCapacity} + d_{i^*}$
      $\texttt{RouteCost}_k \leftarrow \texttt{RouteCost} + L_{p_k i^*} + L_{i^* p+1_k} - L_{p_k p+1_k}$
      $\texttt{Route}_k \leftarrow \{0, 1_k, \ldots, p_k(i^*), i^*, p + 1_k(i^*), \ldots, |k|_k\}$
      Recalculate $w_i, e_i \forall i \in \texttt{Route}_k$
      $LIST \leftarrow LIST \setminus i^*$
      Delete i from $T = (N, A)\ \forall i \in \texttt{Route}_k$
    end if
  end while
  $k \leftarrow k + 1$
end while
$\bar{z} = \sum_{k \in K} \texttt{RouteCost}_k$

extensively in the literature, for example Braysy and Gendreau (2005); Lin and Kernighan (1973); Koskosidis and Powell (1992). The main idea is to evaluate every pair of arcs used by a vehicle and verify if an exchange of the arcs and their directions will yield a cheaper solution. Within the tree context, the order of service of the nodes in a vehicle route is changed if it is feasible to serve the nodes in the order of increasing index. Since the nodes are labeled in DFS order, serving them in increasing order will ensure that the number of times a vehicle traverses an arc is minimized, thereby minimizing the cost.

It is assumed in this heuristic that every vehicle leaves the depot at time A, the earliest possible time it can depart from the depot. After the heuristic has run to completion, the departure time from the depot can then be changed to ensure minimum waiting at the nodes. Further, notice that this heuristic tries to fill up a vehicle before moving on to the next vehicle. The addition of nodes into the vehicles is done such that the route cost is minimized, however, in some sense precedence is given to minimizing the number of vehicles here. However, because of the way the cost functions are defined, greater priority is given to accommodating nodes closest to the last added node because they will have a greater savings than adding a node which is on a different branch of the tree.

A savings type heuristic that solely minimizes total cost, similar to the heuristic presented in Algorithm 6.1, can be developed for the time windows case too. In this savings type heuristic, vehicle routes start simultaneously and two vehicles can be merged if they are both at the same node. The merging

operation requires that time window and capacity constraints are not violated. This is done as follows. Let the two vehicles being merges be k and $k+1$, then the nodes in vehicle k can either be served before or after the nodes in vehicle $k+1$. The two vehicles can be merged if after the merging every node's start service time is still feasible. Although this method considers many vehicles simultaneously, its not clear that the savings operation will eventually decrease the total cost. This is because the time windows of the nodes might result in very few feasible merges. As a result, the total objective cost will go up.

6.8 Computational Results – TCVRPTW

6.8.1 Test Instances

The heuristic was tested on 50–node tree networks. The optimal solution for the problem was obtained by solving the VRPTW IP–formulation in CPLEX. Since the exact solution method is not the focus here, this method was used. Other sophisticated methods like column generation and branch–and–cut methods were not implemented. This section describes the experimental setup and the computational results of this implementation. The test networks were generated such that every node in the tree had between 1 to 5 children. Without loss of generality, the depot was forced to have a degree of one.

Node Time Windows

One of the considerations in generating the node–time windows was that if the problem was solved with the time–windows set to the absolute earliest and latest arrival times for each node, then the solution obtained should equal that of the solution obtained by solving the TCVRP to optimality. This was done as follows. The time at which vehicles can start from the depot, A was set equal to zero. Now, in the TCVRP optimal solution, every vehicle visits its customers in increasing DFS order. Therefore, if a single vehicle were to visit all the nodes in the tree, then it would start at the node labeled 1 and, before returning to the depot, end its route at the node marked $|N| - 1$. The total time required for this TSP tour was set to be the time by which every vehicle must end its tour, B. Now, every node can be feasible served if $a_i \geq L_i$ and $b_i \leq B - s_i - L_i$. Therefore, any time window within this range will result in a feasible solution. The service times s_i were set to zero for all the nodes. Now, setting $a_i = L_i$ and $b_i = B - s_i - L_i$ for all the nodes will result in a TCVRP solution because of the way A and B were set, and because service times are zero. The service start times, a_i, for every node was randomly generated to be 1 to 1.2 times the absolute earliest time the service can start at that node. The service end times, b_i, for every node was randomly generated to be 0.8 to 1 times the absolute latest time service can begin at that node. Finally, cases were tested – when 50% of the nodes have time windows and when 75% of the nodes have time windows. The nodes which were selected to have time windows was done in the same manner backhaul nodes were selected

earlier. If $f\%$ nodes had time windows, first, $|N|$ numbers in `Unif(0,1)` were generated. A two–dimensional array containing these numbers and the node numbers from 1 to $|N|$ was then created. The nodes corresponding to the first $f|N|$ smallest `Unif(0,1)` numbers generated were selected to have time windows. The other parameters were generated as follows.

Arc Times The arc times were uniformly distributed in [1 100].

Demand Each network was tested for five demand profiles uniformly distributed between [1 100], [10 90], [20 80], [40 60] and [50 50].

Capacity The vehicle capacity was set to 100.

Number of Vehicles The formulation requires the total number of vehicles $|K|$. This was set to be equal to the total number of vehicles that were used by the heuristic. Providing the IP with the number of vehicles used by the heuristic helps better compare the optimal solution value to the heuristic value.

Since there are many parameters which can affect the solution and computational quality, the mean over which the demand was distributed was kept constant. All tests were performed on randomly generated 50–node networks. For every demand profile, for every arc time distribution and for every duration constraint, 10 instances were solved. That is, in total, 100 different instances of the problem were solved.

The IP formulation was solved using CPLEX on a PC with a 32–bit architecture, 2GB RAM, and 2.93 GHz processor. The iteration limit for

branch and bound was set to 100000, the time limit was set to 3600 seconds, and the optimality criterion for the MIP (relative gap between the best found integer solution value and best found LP value) was set to 0.0001.

6.8.2 Computational Performance and Solution Quality

The computational results are shown in Table 6.2. For both the cases – 50% and 75% nodes with time windows – the heuristic solution quality decreases and the demand variance increases. However, it is interesting to note that for the 50% case the solution quality actually decreases between the [20 80] and [40 60] demand distributions. The drop is only about 1% and could be attributed to the specific problem instances that were generated. Overall, however, the heuristic performs better for the 50% case than for the 75% case. The heuristic seems to perform the best when the demand variance is 0. Overall, the heuristic solution value was between 5 – 12% of the optimal value. Another interesting aspect is the difference in the number of vehicles used by the exact and heuristic methods. The number of vehicles required by the heuristic is very close to number of vehicles required by the optimal solution. The heuristic seems to predict the number of vehicles exactly when the demand variance is 0. However, the difference is slightly higher for the [20 80] and [40 60] demand distributions.

The time required by the heuristic (which included the 2–opt) implementation ranged from 7 to 18 seconds. Obviously, the time decreased ans the demand variance decreased because fewer vehicles are required when the de-

Network	Time Window %	Demand	Solved[α]	UB/OPT	$\lvert K\rvert_{UB} - v$	TimeUB secs
50–node	75 %	[1 100]	10	1.128	1	18.718
		[10 90]	10	1.091	0.8	15.160
		[20 80]	10	1.098	1.2	13.618
		[40 60]	10	1.077	1.6	11.610
		[50 50]	10	1.067	0	7.823
50–node	50 %	[1 100]	10	1.122	1.4	17.235
		[10 90]	10	1.103	1.4	14.101
		[20 80]	10	1.070	0.6	12.898
		[40 60]	10	1.082	1.6	10.717
		[50 50]	10	1.057	0	7.701

[α] All reported values are averaged over the 10 tested instances of each problem

Table 6.2: Summary of Numerical Results for the TCVRPTW

mand variance is lower. The time required to solve the problems to optimality is not reported here because the method used here to find the optimal solution is a fairly naive one. The new and sophisticated techniques that have been developed for solving the VRPTW report much lower times than the method used here.

6.9 Conclusions

In this chapter, new heuristics for two variants of the TVRP – vehicle time or duration constraints and customer time–window constraints – were proposed. A lower bound for the DTCVRP was suggested in this chapter. Additionally, the IP–formulation was also modified to accommodate duration

constraints. The heuristic for the distance–constrained problem was a savings–type heuristic. Initially, every node is assigned its own vehicle. Two vehicles are then merged together if the merging is feasible and results in a time saving. The procedure starts at the leaf node and moves upward. The heuristic and exact solution methods were implemented on 50–node networks with varying demand and travel time profiles. The problems were tested for two different duration constraint values. Totally, 200 problem instances were tested. In general, the proposed methods performed better when the travel time variance was lower. It was found that the IP performs better as the demand variance decreases. The LP–relaxation values were very tight for all the problem instances and exceeded 95% for most of the instances. The IP solution quality depended on the number of vehicles required by the heuristic. As the difference between the number of vehicles required by the heuristic and IP increased, the solution quality worsened. The heuristic performed reasonably well for all instances tested. The heuristic quality was consistently between 2 – 9% of the optimal solution. The heuristic performed better when the demand variance is lower.

The heuristic proposed for the TCVRPTW built a single vehicle route at a time. When no more nodes can be accommodated into the current vehicle, a new route is created. Nodes were added to the vehicle route based on some cost and waiting time minimizing criteria were defined. A method for generating test instances was also described. the heuristic was implemented on 50–node networks of varying demand profiles. Two types of problems were

generated – 50 ad 75% of the nodes having time windows. In all, 100 problem instances were solved. It was found that the 50% case performed marginally better than the 75% case. The heuristic solution value was between 5 – 12% of the optimal solution value.

This chapter provided only a preliminary analysis of the TCVRPTW. Future research includes developing exact solution methods for this problem, especially column generation techniques, as tree structures might be particularly amenable to such techniques.

Chapter 7

Summary and Scope for Future Work

In this dissertation, a special case of the VRP was considered – where the underlying network has a tree structure (TVRP). The dissertation focused on developing customized algorithms and solution techniques for some unexplored variants of the TVRP. It was shown that the capacitated version of the TVRP is NP–hard, while the uncapacitated version reduces to solving the TSP on trees. The TVRP differs from the VRP in that the network in a VRP contains an edge between every pair of nodes. The main complicating factors in TVRPs is that a vehicle can *visit* a node without actually *serving* it. This is not the case with VRPs, where, whenever a node is visited by a vehicle, it is also served. Some real–world examples of tree networks encountered in river networks, railway networks, and rural areas were provided.

Properties, lower bounds, heuristics, and exact solution methods for the following four variants of the TVRP were discussed: (a) TVRPs with Backhauls (b) TVRPs with Heterogeneous Fixed Fleets, (c) TVRPs with Duration/Time or Distance constraints, and (d) TVRPs with Time Windows. The adaptibility of the solution techniques for different versions of the same variant – capacitated, multiple vehicle types, cost minimization objective, and

vehicle minimization objective – was also demonstrated. An overview of the dissertation contributions is provided in the next section, followed by directions for future research.

7.1 Dissertation Contributions and Conclusions

- **TVRP with Backhaul Customers**: In capacitated TVRPs with Backhauls (TCVRPB), the customers are partitioned into two subsets. The first subset consists of customers who have placed an order for a given quantity of product to be delivered from the depot – the *linehaul* customers. The second subset consists of customers who require a given quantity of product to be picked up from their location and delivered to the depot – the *backhaul* customers. Furthermore, in a vehicle tour, the linehaul customers have to be served before backhaul customers.

  A lower bound for the TCVRPB was derived by using its relation with bin–packing problems, and a few properties and key observations that hold true at optimality were delineated. Two IP formulations were proposed to solve the problem. The second IP was formulated by transforming the tree into an equivalent network. It was shown that this IP had fewer variables and that its LP–relaxation was stronger than the first one. A few valid inequalities were added to the second IP formulation in an effort to increase its LP–relaxation value.

  An algorithm, which, at every stage deletes at least one node from the network was presented. The algorithm had two main steps – finding the

customers that are serviced by each vehicle, and constructing the optimal routes for that vehicle. The optimal vehicle routes were constructed by using the algorithm for the Traveling Salesman Problem on Trees with Backhauls (TTSPB), a polynomial algorithm for which was developed in this dissertation.

The second IP Formulation and the heuristic were tested on problem instances of varying sizes and demand profiles. 10 instances of each network size and demand profile was tested. Computational times for solving the IP depended not only on the network size, but also on the demand distribution and the number of vehicles in the IP. Computational time increased with an increase in problem size, as supported by intuition. The computational time and number of branch–and–bound nodes increased with an increase in the number of vehicles and an increase in the demand variation. Based on this insight, further improvements were suggested and a new heuristic which reduces the total number of vehicles was proposed.

- **Capacitated TVRP with Fixed Fleets**: A constrained case of the TVRP – where the vehicle fleet is capacitated, heterogeneous, and fixed (HTCVRP) was studied in this dissertation. The HTCVRP's relation to the Generalized Assignment Problem and the Capacitated Facility Location Problem was discussed. It was shown that when the nodes were arranged in DFS order, the existing IP formulation can be modified to have only GAP constraints. A linear approximation to this modified

formulation was presented. A heuristic was then developed to use this GAP formulation to solve the HTCVRP. The heuristic iteratively finds seed nodes using Upper Bound (UB) heuristics for the BPP, assigns a vehicle to each seed node, then uses a Lagrangian–based GAP algorithm to assign nodes to vehicles located at the seed node. Two methods for finding seed nodes were also presented.

The heuristic was tested on networks of varying sizes and demand distributions. It was found that the heuristic performs consistently well irrespective of problem size while taking much lesser time than the optimal solution method. It was concluded that significant improvements in optimal solution time can be achieved if the heuristic solution was used to initialize the optimal solution method.

- **Duration–constrained TVRP**: The Duration Constrained Capacitated TVRP is concerned with finding optimal time minimizing vehicle routes that serve customers that are located on a tree network. Each vehicle is specified a tour time limit which cannot be exceeded. A lower bound for the DTCVRP was suggested in this chapter. Additionally, the IP–formulation was also modified to accommodate duration constraints. The heuristic for the distance–constrained problem was a savings–type heuristic. Initially, every node is assigned its own vehicle. Two vehicles are then merged together if the merging is feasible and results in a time saving. The procedure starts at the leaf node and moves upward.

  The heuristic and exact solution methods were implemented on networks

with varying demand and travel time profiles. The problems were tested for different duration constraint values. In general, the proposed methods performed better when the travel time variance was lower. It was found that the IP performed better for lower demand variations. The LP–relaxation values were very tight for all the problem instances and the IP solution quality depended on the number of vehicles required by the heuristic. As the difference between the number of vehicles required by the heuristic and IP increased, the solution quality worsened. The heuristic performed consistently well for all instances tested. The heuristic performed better when the demand variance was lower.

- **TVRP with Time–Windows**: The objective of the Capacitated TVRP with Time Windows (TCVRPTW) is to find cost minimizing vehicle routes beginning and ending at the depot such that the total demand served by each vehicle does not exceed its capacity and a vehicle serves a customer only during the time window specified by the customer. The heuristic proposed for the TCVRPTW built a single vehicle route at a time. When no more nodes can be accommodated into the current vehicle, a new route is created. Nodes were added to the vehicle route based on some cost and waiting time minimizing criteria were defined. A method for generating test instances was also described. the heuristic was implemented on networks of varying demand profiles. Two types of problems were tested. It was found that lower the number of customers with time–windows better the heuristic performance. The heuristic so-

lution value was between 5 – 12% of the optimal solution value.

7.2 Scope for Future Work

- **TVRP with Time Windows**: This dissertation provided a heuristic for the TVRP with time windows. This heuristic too was a modification of an earlier heuristic applied to VRPs on general networks. Developing better heuristics for the TVRP with time windows is a future research direction. Further, there are currently no exact methods for solving the TCVRPTW which explicitly take advantage of the tree structure. A couple of different ideas for solving the problem to optimality are suggested here. First, the main complicating factor with taking advantage of a tree structure is that depending on how the time windows for the problem are defined, a vehicle can enter a branch multiple times. In the worst case once to serve every node in the branch. Using this, one can transform the network to contain as many copies of a node as there are nodes below it. This, however, might have more variables than constructing $|N|^2$ arcs and solving the normal VRPTW. A more viable approach might be to use a time–space expanded network to formulate the problem. Depending on the peak time period and the customer time–windows, it might be possible to discretize the network and even reduce the number of copies of each node.

- **Clustering of Nodes**: An important application of tree–networks arise in areas where clustering some nodes results in an overall tree structure.

The cumulative demand of the clustered nodes is less than vehicle capacity. So, given a general graph, if there exists a method to intelligently cluster nodes so that the resulting tree structure will closely approximate the general network costs, the TVRP solution method can be used to solve the problem. Of course, the clustering method has to be quick and efficient and the method should be superior to the current sophisticated metaheuristic approaches for solving VRPs.

- **Special Network Types**: Since this dissertation focused on solving VRPs for a special network type, a question that arises is whether customized VRP solution techniques can be devised for other special network types. Some of these special network types include grid networks and shoreline networks. The literature on VRPs for grid–like networks is very scant and definitely a future research direction. Uncapacitated problems have been solved on shoreline networks, but the capacitated versions of such problems are unexplored.

- **Other Variants** Many other variants of the TVRP are yet to be explored. Using the concepts developed in this dissertation, solution strategies for variants which combine two or more of the variants in this dissertation can be developed. Additionally, the stochastic and dynamic versions of the problems in this dissertation can also be explored.

Bibliography

Achuthan, N., Caccetta, L., 1991. Integer linear programming formulation for a vehicle routing problem. European journal of operational research 52 (1), 86–89.

Agarwal, Y., Mathur, K., Salkin, H., 1989. A set-partitioning-based exact algorithm for the vehicle routing problem. Networks 19 (7), 731–749.

Ahuja, R., Magnanti, T., Orlin, J., 1993. Network flows: Theory, algorithms, and applications.

Anily, S., 1996. The Vehicle Routing Problem with Delivery and Backhaul Options. Naval Research Logistics 43 (3), 415–434.

Anily, S., Mosheiov, G., 1994. The traveling salesman problem with delivery and backhauls. Operations Research Letters 16 (1), 11–18.

Asano, T., Katoh, N., Kawashima, K., 2001. A New Approximation Algorithm for the Capacitated Vehicle Routing Problem on a Tree. Journal of Combinatorial Optimization 5 (2), 213–231.

Bailey, E., Unnikrishnan, A., Lin, D.-Y., 2010. Models for Minimizing Backhaul Costs through Freight Collaboration. In: 90^{th} Annual Conference of the Transportation Research Board. Washington, D.C., U.S.A.

Balakrishnan, A., Magnanti, T., Mirchandani, P., 1996. Heuristics, lps, and trees on trees: Network design analyses. Operations Research, 478–496.

Balakrishnan, A., Ward, J., Wong, R., 1987. Integrated facility location and vehicle routing models: Recent work and future prospects. American Journal of Mathematical and Management Sciences 7 (1), 35–61.

Baldacci, R., Christofides, N., Mingozzi, A., 2008. An Exact Algorithm for the Vehicle Routing Problem based on the Set Partitioning Formulation with Additional Cuts. Mathematical Programming 115 (2), 351–385.

Balinski, M., Quandt, R., 1964. On an Integer Program for a Delivery Problem. Operations Research 12 (2), 300–304.

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., Vance, P., 1998. Branch-and-price: Column generation for solving huge integer programs. Operations Research, 316–329.

Basnet, C., Foulds, L., Wilson, J., 1999. Heuristics for Vehicle Routing on Tree–like Networks. Journal of the Operational Research Society 50, 627–635.

Beasley, J., Christofides, N., 1997. Vehicle routing with a sparse feasibility graph. European Journal of Operational Research 98 (3), 499–511.

Ben Atkinson, J., 1994. A greedy look–ahead heuristic for combinatorial optimization: An application to vehicle scheduling with time windows. Journal of the Operational Research Society, 673–684.

Berger, I., Bourjolly, J.-M., Laporte, G., 1992. Branch–and–bound algorithms for the multi–product assembly line balancing problem. European Journal of Operational Research 58 (2), 215 – 222.
URL http://www.sciencedirect.com/science/article/pii/0377221792902080

Bhattacharya, B., Carmi, P., Hu, Y., Shi, Q., 2008. Single Vehicle Scheduling Problems on Path/Tree/Cycle Networks with Release and Handling Times. Algorithms and Computation, 800–811.

Bodin, L., Golden, B., Assad, A., Ball, M., 6 1983. Routing and scheduling of vehicles and crews – the state of the art. Computers and Opertions Research 10 (2), 63–212.

Bodin, L. D., 1990. Twenty years of routing and scheduling. Operations Research 38 (4), pp. 571–579.
URL http://www.jstor.org/stable/171075

Braysy, O., Gendreau, M., 2005. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. Transportation Science 39 (1), 104–118.

Brodie, G., Waters, C., 1988. Integer linear programming formulation for vehicle routing problems. European journal of operational research 34 (3), 403–404.

Brown, G. G., Ellis, C. J., Graves, G. W., Ronen, D., 1987. Real-time, wide area dispatch of mobil tank trucks. Interfaces 17 (1), 107–120. URL http://interfaces.journal.informs.org/cgi/content/abstract/17/1/107

Busch, I., 1990. Vehicle Routing on Acyclic Networks. Ph.D. dissertation, Johns Hopkins University, Department of Applied Mathematics.

Casco, D. O., Golden, B. L., Wasil, E. A., 1988. Vehicle Routing: Methods and Studies. In: Golden, B. L., Assad, A. A. (Eds.), Vehicle Routing: Methods and Studies. North–Holland, Amsterdam, pp. 127–147.

Cassidy, P. J., Bennett, H. S., 1972. Tramp – a multi-depot vehicle scheduling system. Operational Research Quarterly (1970-1977) 23 (2), pp. 151–163. URL http://www.jstor.org/stable/3008264

Chandran, B., Raghavan, S., 2008. Modeling and Solving the Capacitated Vehicle Routing Problem on Trees. In: Golden, B. L., Raghavan, S., Wasil, E. A. (Eds.), The Vehicle Routing Problem: Latest Advances and New Challenges. Vol. 43. Springer–Verlang, New York, pp. 239–261.

Choi, E., Tcha, D., 2007. A column generation approach to the heterogeneous fleet vehicle routing problem. Computers & Operations Research 34 (7), 2080–2095.

Christofides, N., 1985. Vehicle Routing. The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, 431–448.

Christofides, N., Mingozzi, A., Toth, P., 1981a. Exact Algorithms for the Vehicle Routing Problem, based on Spanning Tree and Shortest Path Relaxations. Mathematical programming 20 (1), 255–282.

Christofides, N., Mingozzi, A., Toth, P., 1981b. State–space Relaxation Procedures for the Computation of Bounds to Routing Problems. Networks 11 (2), 145–164.

Clarke, C., Wright, J. Q., 1964a. Scheduling of Vehicle from a Central Depot to a Number of Delivery Points. Operations Research 12 (4), 568–581.

Clarke, G., Wright, J. W., 1964b. Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12 (4), pp. 568–581. URL http://www.jstor.org/stable/167703

Dantzig, G., Ramser, J., 1959. The Truck Dispatching Problem. Management Science 6 (1), 80–91.

Deif, I., Bodin, L., 1984a. Extension of the Clarke and Wright Algorithm for Solving the Vehicle Routing Problem with Backhauling. In: Proceedings of the Babson Conference on Software Uses in Transportation and Logistics Management. pp. 75–96.

Deif, I., Bodin, L., 1984b. Extension of the clarke and wright algorithm for solving the vehicle routing problem with backhauling. In: Proceedings of the babson conference on software uses in transportation and logistics management. Babson Park, MA, pp. 75–96.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., 1994. A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems. Cahiers du GERAD.

Desrochers, M., Desrosiers, J., Solomon, M., 1992. A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows. Operations Research 40 (2), 342–354.

Desrochers, M., Verhoog, T., 1991. A new heuristic for the fleet size and mix vehicle routing problem. Computers & Operations Research 18 (3), 263–274.

Desrosiers, J., Dumas, Y., Solomon, M., Soumis, F., 1995. Time Constrained Routing and Scheduling. Handbooks in operations research and management science 8, 35–139.

Eilon, S., Watson-Gandy, C., Christofides, N., de Neufville, R., 2007. Distribution Management – Mathematical Modelling and Practical Analysis. Systems, Man and Cybernetics, IEEE Transactions on 4 (6), 589.

Eksioglu, B., Vural, A., Reisman, A., 2009. The Vehicle Routing Problem: A taxonomic review. Computers & Industrial Engineering 57 (4), 1472–1483.

Fekete, S. P., Schepers, J., 2001. New classes of fast lower bounds for bin packing problems. Mathematical Programming 91, 11–31, 10.1007/s101070100243.

URL http://dx.doi.org/10.1007/s101070100243

Fisher, M., 1994a. A Polynomial Algorithm for the Degree–constrained Minimum k–tree Problem. Operations Research 42 (4), 775–779.

Fisher, M., 1994b. Optimal Solution of Vehicle Routing Problems using Minimum k–trees. Operations Research 42 (4), 626–642.

Fisher, M., 1995. Vehicle routing. Handbooks in operations research and management science 8, 1–33.

Fisher, M., Jaikumar, R., 1981a. A Generalized Assignment Heuristic for Vehicle Routing. Networks 11 (2), 109–124.

Fisher, M., Jaikumar, R., 1981b. A Generalized Assignment Heuristic for Vehicle Routing. Networks 11 (2), 109–124.

Fisher, M., Jörnsten, K., Madsen, O., 1997. Vehicle Routing with Time Windows: Two Optimization Algorithms. Operations Research 45 (3), 488–492.

Foster, B., Ryan, D., 1976. An Integer Programming approach to the Vehicle Scheduling Problem. Operational Research Quarterly 27 (2), 367–384.

Frederickson, G., Hecht, M., Kim, C., 1976. Approximation Algorithms for some Routing Problems. In: 17th Annual Symposium on Foundations of Computer Science. IEEE, pp. 216–227.

Garey, M., Johnson, D., 1979. Computers and Intractability. A guide to the theory of NP–completeness. A Series of Books in the Mathematical Sciences. WH Freeman and Company, San Francisco, Calif.

Gendreau, M., Laporte, G., Séguin, R., 1996. Stochastic Vehicle Routing. European Journal of Operational Research 88 (1), 3–12.

Gendreau, M., Potvin, J., 1998. Dynamic Vehicle Routing and Dispatching. Tech. rep., Centre de recherche sur les transports and Departement d'informatique et de recherche operationnelle, Universite de Montreal.

Gilbert, D., et al., 1991. Improvements and Extensions to the Miller–Tucker–Zemlin Subtour Elimination Constraints. Operations Research Letters 10 (1), 27–36.

Goetschalckx, M., Jacobs-Blecha, C., 1993. The Vehicle Routing Problem with Backhauls: Properties and Solution Algorithms. Tech. Rep. MHRC–TR–88–13, Georgia Institute of Technology.

Golden, B., Assad, A., Levy, L., Gheysens, F., 1984. The fleet size and mix vehicle routing problem. Computers & Operations Research 11 (1), 49–66.

Golden, B., Magnanti, T., Nguyen, H., 1977. Implementing Vehicle Routing Algorithms. Networks 7 (2), 113–148.

Golden, B. L., Raghavan, S., Wasil, E. A., 2008. The Vehicle Routing Problem: Latest Advances and New Challenges. Springer–Verlang, New York.

Golden, B. L., Wasil, E. A., 1987. Or practice – computerized vehicle routing in the soft drink industry. Operations Research 35 (1), 6–17.
URL http://or.journal.informs.org/cgi/content/abstract/35/1/6

Hadjiconstantinou, E., Christofides, N., Mingozzi, A., 1995. A New Exact Algorithm for the Vehicle Routing Problem based on q–paths and k–shortest paths Relaxations. Annals of Operations Research 61 (1), 21–43.

Hamaguchi, S., Katoh, N., 1998. A Capacitated Vehicle Routing Problem on a Tree. Algorithms and Computation 1533, 399–407.

Hinton, T., 2010. The vehicle routing problem. Ph.D. thesis, University of Bristol.

Houck, D., 1978. The Traveling Salesman Problem as a Constrained Shortest Path Problem: Theory and Computational Experience. Ecole polytechnique (Montréal, Québec). Département de génie industriel.

Jean-Marie, B., et al., 1992. Branch–and–Bound Algorithms for the Multi–product Assembly Line Balancing Problem. European Journal of Operational Research 58 (2), 215–222.

Jeet, V., Kutanoglu, E., 2007. Lagrangian relaxation guided problem space search heuristics for generalized assignment problems. European journal of operational research 182 (3), 1039–1056.

Kallehauge, B., Larsen, J., Madsen, O., Solomon, M., 2005. Vehicle Routing Problem with Time Windows. Column Generation, 67–98.

Kallrath, J., 2004. Modeling languages in mathematical optimization. Vol. 88. Springer.

Kara, I., Laporte, G., Bektas, T., 2004. A note on the Lifted Miller–Tucker–Zemlin Subtour Elimination Constraints for the Capacitated Vehicle Routing Problem. European Journal of Operational Research 158 (3), 793–795.

Karuno, Y., Nagamochi, H., 2001. A Polynomial Time Approximation Scheme for the Multi–vehicle Scheduling Problem on a Path with Release and Handling times. Algorithms and Computation, 36–48.

Karuno, Y., Nagamochi, H., 2003. 2-Approximation Algorithms for the Multi–vehicle Scheduling Problem on a Path with Release and Handling Times. Discrete Applied Mathematics 129 (2-3), 433–447.

Karuno, Y., Nagamochi, H., Ibaraki, T., 1997. Vehicle Scheduling on a Tree with Release and Handling Times. Annals of Operations Research 69, 193–207.

Katoh, N., Yano, T., 2006. An Approximation Algorithm for the Pickup and Delivery Vehicle Routing Problem on Trees. Discrete Applied Mathematics 154 (16), 2335–2349.

Knödel, W., 1981. A bin packing algorithm with complexity o(n log n) and performance 1 in the stochastic limit. In: Proceedings on Mathematical Foundations of Computer Science. Springer-Verlag, London, UK, pp. 369–378.

Koskosidis, Y., Powell, W., 1992. Clustering algorithms for consolidation of

customer orders into vehicle shipments. Transportation Research Part B: Methodological 26 (5), 365–379.

Kumar, R., Unnikrishnan, A., Waller, Travis, S., 2011. The capacitated vehicle routing problem with backhauls on trees: Model, properties, formulation, and algorithm, transportation Research Record (accepted).
URL http://amonline.trb.org/12kltu/1

Labbé, M., Laporte, G., Mercure, H., 1991. Capacitated Vehicle Routing on Trees. Operations research 39 (4), 616–622.

Laporte, G., 1992. The Vehicle Routing Problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59 (3), 345–358.

Laporte, G., 2009. Fifty years of vehicle routing. Transportation Science 43 (4), 408–416.

Laporte, G., Desrochers, M., Nobert, Y., 1984. Two exact algorithms for the distance–constrained vehicle routing problem. Networks 14 (1), 161–172.

Laporte, G., Mercure, H., Nobert, Y., 1986. An Exact Algorithm for the Asymmetrical Capacitated Vehicle Routing Problem. Networks 16 (1), 33–46.

Laporte, G., Mercure, H., Nobert, Y., 1992. A Branch and Bound Algorithm for a class of Asymmetrical Vehicle Routing Problems. Journal of the Operational Research Society 43 (5), 469–481.

Laporte, G., Nobert, Y., 1983. A Branch and Bound Algorithm for the Capacitated Vehicle Routing Problem. OR Spectrum 5 (2), 77–85.

Laporte, G., Nobert, Y., Desrochers, M., 1985. Optimal Routing under Capacity and Distance Restrictions. Operations Research 33 (5), 1050–1073.

Laporte, G., Semet, F., 2002. Classical Heuristics for the Capacitated VRP. The vehicle routing problem, 109–128.

Lenstra, J., Kan, A., 1981. Complexity of Vehicle Routing and Scheduling Problems. Networks 11 (2), 221–227.

Letchford, A., Eglese, R., Lysgaard, J., 2002. Multistars, Partial Multistars and the Capacitated Vehicle Routing Problem. Mathematical Programming 94 (1), 21–40.

Lin, S., Kernighan, B. W., 1973. An effective heuristic algorithm for the traveling-salesman problem. Operations Research 21 (2), pp. 498–516.
URL http://www.jstor.org/stable/169020

Liu, S., Ng, K., Ong, H., 2008. Branch–and–bound algorithms for simple assembly line balancing problem. The International Journal of Advanced Manufacturing Technology 36, 169–177, 10.1007/s00170-006-0821-y.
URL http://dx.doi.org/10.1007/s00170-006-0821-y

Lysgaard, J., Letchford, A., Eglese, R., 2004. A New Branch–and–Cut Algorithm for the Capacitated Vehicle Routing Problem. Mathematical Programming 100 (2), 423–445.

Martello, S., Toth, P., July 1990. Lower bounds and reduction procedures for the bin packing problem. Discrete Appl. Math. 28, 59–70.
URL http://portal.acm.org/citation.cfm?id=89011.89025

Mbaraga, P., Langevin, A., Laporte, G., 1999. Two Exact Algorithms for the Vehicle Routing Problem on Trees. Naval Research Logistics 46 (1), 75–89.

McCarl, B., Meeraus, A., van der Eijk, P., Bussieck, M., Dirkse, S., Steacy, P., Nelissen, F., 2008. Mccarl gams user guide. Gams Development Cooperation, Washington, USA.

Miller, C., Tucker, A., Zemlin, R., 1960. Integer Programming Formulation of Traveling Salesman Problems. Journal of the ACM (JACM) 7 (4), 329.

Mingozzi, A., Giorgi, S., Baldacci, R., 1999. An Exact Method for the Vehicle Routing Problem with Backhauls. Transportation Science 33 (3), 315–329.

Murty, K., 1985. Linear and combinatorial programming. ROBERT E. KRIEGER PUBLISHING COMPANY, MELBOURNE, FL(USA), 1985, 592.

Muslea, I., 1997. The Very Offline k–Vehicle Routing Problem in Trees. Computer Science Society, 1997. Proceedings., XVII International Conference of the Chilean, 155–163.

Nagarajan, V., Ravi, R., 2008. Approximation algorithms for distance constrained vehicle routing problems.

Nemhauser, G., Wolsey, L., 1988. Integer and Combinatorial Optimization. Vol. 18. Wiley New York.

Orloff, C., 1976. Route Constrained Fleet Scheduling. Transportation Science 10 (2), 149.

Psaraftis, H., Solomon, M., Magnanti, T., Kim, T., 1990. Routing and Scheduling on a Shoreline with Release Times. Management Science 36 (2), 212–223.

Rao, M., Zionts, S., 1968. Allocation of Transportation units to Alternative Trips – A Column Generation Scheme with out–of–kilter Subproblems. Operations Research 16 (1), 52–63.

Renaud, J., Boctor, F., 2002. A sweep-based algorithm for the fleet size and mix vehicle routing problem. European Journal of Operational Research 140 (3), 618–628.

Rennie, S., 1995. Optimal Dispatching and Routing of Milk Tanker for Northland Dairy Board. In: 30^{th} Annual Conference of the Operational Research Society of New Zealand. ORSNZ: Wellington, New Zealand, pp. 95–102.

Salhi, S., Rand, G., 1993. Incorporating vehicle routing into the vehicle fleet composition problem* 1. European Journal of Operational Research 66 (3), 313–330.

Solomon, M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. Operations research, 254–265.

Taillard, É., 1999. A heuristic column generation method for the heterogeneous fleet vrp. RAIRO-Operations Research 33 (01), 1–14.

Tan, K., Lee, L., Zhu, Q., Ou, K., 2001. Heuristic Methods for Vehicle Routing Problem with Time Windows. Artificial Intelligence in Engineering 15 (3), 281–295.

Toth, P., Vigo, D., 1996. A Heurisic Algorithm for the Vehicle Routing Problem with Backhauls. Advanced Methods in Transportation Analysis: Proceedings, Second TRISTAN Conference, 585–608.

Toth, P., Vigo, D., 1997. An Exact Algorithm for the Vehicle Routing Problem with Backhauls. Transportation Science 31, 372–3852.

Toth, P., Vigo, D., 2002. The Vehicle Routing Problem. SIAM, Philadelphia.

Tsitsiklis, J., 1992. Special Cases of Traveling Salesman and Repairman Problems with Time Windows. Networks 22 (3), 263–282.

Wan, Z., 2009. Freight transportation planning: Container transportation network within china's yangtze river. Ph.D. thesis, University of California, Davis.

Wren, A., Holliday, A., 1972. Computer scheduling of vehicles from one or more depots to a number of delivery points. Operational Research Quarterly 23 (3), 333–344.

Yu, W., Liu, Z., 2010. Single–vehicle Scheduling Problems with Release and Service Times on a Line. Networks.
URL http://dx.doi.org/10.1002/net.20393

Vita

Roshan Kumar was born in Bombay, India on March 7th, 1984 to Mahalakshmi Kumar and T. V. Kumar. He received his Bachelor of Engineering degree in Industrial (Production) Engineering from the University of Bombay, India in June, 2006 and his Master of Science degree in Operations Research and Statistics from the University of North Carolina at Chapel Hill in May, 2008. He began his doctoral studies in Transportation Engineering under Dr. S. Travis Waller in August, 2008.

Permanent address: 1020 E. 45th, Apt. # 159
Austin, TX 78751

This dissertation was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.