A LARGE-SCALE NEIGHBORHOOD SEARCH APPROACH TO VEHICLE ROUTING PICK-UP AND DELIVERY PROBLEM WITH TIME WINDOWS UNDER UNCERTAINTY

BY

PRAVEEN TUMULURI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Industrial Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2015

Urbana, Illinois

Adviser:

Assistant Professor Lavanya Marla

ABSTRACT

The vehicle routing problem with shipment pick-up and delivery with time windows (VRPPDTW) is one of the core problems that is addressed by a package delivery company in its operations. Most often, this problem has been addressed from the point of view of cost-cutting, to achieve the lowest cost possible under a given/predicted demand and service time scenario. This thesis aims to study a real-world VRPPDTW problem with side-constraints and build solutions that are cost-effective as well as robust to stochasticity in demands and service times. Even without the additional side constraints, the VRPPDTW is NP-hard. In particular, we consider the solution of VRPPDTW with side-constraints adopted by a carrier. Because of the nature as well as the size of the problem and the network, we demonstrate that the problem is combinatorially explosive. We therefore develop a large-scale neighbourhood search heuristic combined with a break-and-join heuristic and a clustering heuristic. We use this heuristic to build a set of schedules with far lower operating costs than the existing solution and effectively decrease the costs by 15% by reducing the number of routes needed to serve the shipments. We then build a framework to evaluate the performance of the solutions under stochasticity, and present results related to under stochasticity in service times.

ACKNOWLEDGEMENT

It is my privilege and honor to work with Dr. Lavanya Marla. I would like to express my deepest gratitude to her for her constant understanding and support. I would like to thank her whole-heartedly for being patient with me, even during the toughest of my time, for taking time out of her busy schedule in teaching me some aspects of the code and for her constant guidance. This thesis could not be completed without her invaluable advice. Thank you, Dr. Marla, for being a great mentor.

I am grateful to Ashish Khetan Date and Aaron Williams, who lent me their constant support and for their patience in hinting out some problems in my thesis. Thanks are due to Parasara Sridhar Duggirala for his insightful advice on Randomized Algorithms, providing some of the key structure to this thesis. I would also like to thank my friends Bhargava Reddy and Virajith Jalaparthi for constantly providing me with all the support I very much needed.

I would like to also thank Dr. Ramavarapu Sreenivas and Dr. Carolyn Beck for providing a new direction to my thought process in my thesis, my colleague Garrick Li to have the patience in waiting on the server until my code ran to completion. I would like to thank my friends Sree Kalyan, Sasidhar, Raghavendra, Sweta Yamini, Yashwant and Lakshmi, whose friendship made my stay at UIUC much more enjoyable and memorable. Also, I would like to thank all UIUC friends who have indirectly contributed in making this thesis possible.

I am deeply indebted to all of my teachers. Without any of them, I would not be where I am today. More importantly, I owe my deepest gratitude to my parents and family members for their love and support. They are my everything and I love them all.

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 Statement of the problem

In this thesis we study the problem of vehicle routing with pickup and delivery with time windows, commonly referred to as VRPPDTW. As defined in Toth [1], '*The* ***vehicle routing problem with pickup and delivery with time windows (VRPPDTW)*** *is a combinatorial optimization and integer programming problem which generalizes Travelling Salesman Problem (TSP). The objective of the VRPPDTW is to minimize the total route cost with the constraints that since a number of goods need to be moved from certain pickup locations to other delivery locations, we find optimal routes for a fleet of vehicles to visit the pickup and drop-off locations and the delivery locations have time windows within which the deliveries (or visits) must be made'*. The VRPPDTW problem has a fleet of vehicles that has to be routed to serve a set of shipments that have to be picked up and delivered at multiple points in the network. In particular, given a set of trailers with units in trailer loads denoted by their origins and destinations by their earliest available times and latest arrival times, we first find cost-minimizing routes and schedules for both the drivers and the tractors.

The VRPPDTW is NP hard. According to Leewuen [2], ' ***NP-hard*** *(**N**on-deterministic **P**olynomial-time hard), in computational complexity theory, is a class of problems that are, informally, "at least as hard as the hardest problems in NP" '*. More precisely, a problem *H* is NP-hard when every problem *L* in NP can be reduced in polynomial time to *H*. As a consequence, finding a polynomial algorithm to solve any NP-hard problem would give polynomial algorithms for all the problems in NP, which is unlikely as many of them are considered hard. Therefore, it is difficult to prove that there exists an algorithm that can produce an optimal or close-to-optimal solution in a bounded time, unless enumeration is used. Additionally, as we will describe in the future sections, the problem has several side-constraints which make the modeling and solution process even more complex.

1.1.1 Objective

The objective of this thesis is this to come up with a robust modeling and solution framework that can generate solutions that are cost-effective, and additionally, robust to uncertainties in the package demands and service times. In particular, our primary objective is to find the minimum number of vehicles needed to pick up and drop off the set of demands (shipments) within the specified time- windows. The constraints of the model are as follows: given a set of trailer demands and units in trailer loads denoted by their origins and destinations by their earliest available times and latest arrival times, and service times in the form of travel and transfer times, we first find routes and schedules for both the drivers and the tractors. We then study the set of solutions (routes) obtained by the modeling framework. As a secondary objective, we study the robustness properties of the solutions obtained from the model. That is, we would like to understand the performance of the solutions obtained, under stochasticity in trailer demands and service times.

1.1.2 Detailed description of the Problem and Side-Constraints

In this section, we will describe the main parameters and constraints involved in the vehicle routing problem with time windows. The underlying network is described as a directed graph *G=(N,A)* with *N* the set of nodes and *A* the set of arcs. The set of nodes correspond to the locations used by the company. These locations are of various types where some locations correspond to hubs, others to sorting facilities, railyards etc. The arcs connect two different locations such that the location at the head of the arc can be directly reached upon leaving the location of the tail of the arc, through the specified route. For each such arc in the network, both travel time and travel distances are known.

The next step involves the requirement to route correctly the different loads. Here, a load corresponds to a trailer full of packages. Each load is associated with a time window, that is an earliest available time (EAT) and a latest delivery time (LDT). The two times are determined apriori in order to follow this set of constraints: 1) the schedules of the carrier sorting facilities; 2) other time windows; for example if a load is sent over more than one arc length, the time windows affecting the load will be defined to satisfy any time restrictions of the regions involved. The EAT and LDT are used as input to our VRPPDTW model.

There exist two different types of trailers. There are short and long trailers. Due to its capacity constraints, a tractor can hold either one short trailer, one long trailer or two short trailers. When the tractor hauls only one load, Tractor/Trailer configuration is denoted as *Single*; where the tractor contains two trailers; its configuration is denoted as *Double*. In this thesis, we used the words ‘tractor’ and ‘load’ interchangeably. Thus, the adjective ‘short’ or ‘long’ is directly applied to the loads.

It is allowed that two drivers can exchange loads. This means that a driver can drop a load at a location (that is not the destination of the load), from which it is later picked up by another driver. Within a driver's route, various work rules have to be in compliance. First of all, a driver has to begin and to finish his work schedule from and to the same point which is called a domicile. Only a small subset of locations is allowable domiciles. Next, work rules also imply that the working time cannot exceed a working time limit which depended on the domicile to which the driver is associated. Furthermore, within a work day, a driver must have up to two breaks and one meal break. The length of the aforesaid breaks and the time at which they occur, also depend on the domicile associated with the driver.

Finally, other secondary constraints exist. In particular, when a truck arrives or exits a location, a parameter called turn time is associated with unloading and loading operations on the tractor and for turning around the location among other things. Furthermore, there may be local laws that govern on trailers that prohibit their allowance on some of the arcs. We describe these constraints in further detail in Chapter 3, along with their mathematical formulations.

1.2 Data Set and Computational Complexity

In this thesis, we focus our experiments on the large-scale, real-world network of a large postal delivery carrier. There are 98 locations and 347 shipments need to be transported from their origins to destinations. There are 496 arcs connecting possible direct paths between the 98 locations. For each arc, we are given the distance and mean travel time. Six of the locations are bases where vehicles should begin and end their routes, and also serve to transfer shipments across vehicles. Each trailer (shipment) can be picked up after its *EAT* and should be dropped off before its *LDT*. The loads are correspondingly given a *trlr Type ID* and also whether it could be included as *short* or *Long*, short requiring a capacity of one and long requiring a capacity of two. Overrides correspond to movements which are made at a particular location (moving trailers inside a facility). The thesis also incorporates parameters a. *isOverride*, which indicates whether there is some override time as a requirement for the load. b. *overrideTime*, which is the length of the override and c. *trlrTypeID*, ID of the trailer associated with the load.

The time frame in which the schedule is developed corresponds to a two day period. There are 6 domiciles out of 98 locations and that work rules are not homogenous within domiciles. There is a specified guaranteed paid day denoted *minDay* and a maximum legal work day denoted *maxDay*. Any work beyond *minDay* will be considered as over time work, and there is a maximum of *maxDay-minDay* of possible allowable overtime.

Due to the large size of the data set as well as the complex side-constraints, this becomes a computationally complex problem to solve. The primary complexity in the problem is initially observed through the time windows which dictate the feasibility of whether two loads can travel together (taking into account the waiting time, delivery time, pickup time and break constraints). The subsequent step involves combining the loads to form parts of routes, and subsequently, full routes. These combinations of loads further increase the complexity.

Specifically, we adopt an approach based on large-scale neighbourhood search within which an optimization module is embedded. This allows us to create combinations of loads that we refer to as metaloads. We then further combine the metaloads using clustering methods, which exponentially increases the complexity. We use combinations of break-and-rejoin heuristic, an insertion heuristic and a clustering heuristic.

Finally we check for the robustness of the solutions using a simulator, under varying demands and service times. The objective function itself poses an interesting problem because of non linearity of the feasibility function.

1.3 Outline of the Thesis

In Chapter 2, we present existing literature and discuss the various developments that have taken place over the years in the context of the VRPPDTW. Chapters 3 and 4 describe our modeling approach and solution approach respectively. Chapter 4 discusses the solutions. We conclude in Chapter 5 by summarizing our results and describing the future scope of this research.

CHAPTER 2 LITERATURE REVIEW

2.1 Genetic Algorithm and Grouping Algorithm Heuristics for a vehicle routing problem with delivery and pickup with time windows

The vehicle routing problem with pick-up and delivery time windows (henceforth referred to as VRPPDTW or PDPTW) has been one of the most exhaustively studied problems in literature. Because this problem is NP-hard [2], there have been a variety of approaches, including heuristic approaches, to solve it. This chapter is dedicated to discussing such existing approaches. Xiaolan and Rodriguez [4] give their insights using the vehicle routing formulation applied to health care departments. They consider a scenario with four types of demand leading upto the depot and medical lab respectively. In this model, the time windows between the patients and the vehicular capacity are considered crucial. Two mixed-integer programming models are proposed in this paper viz., an MIP derived from the model of Dell'Amico et al. [5] and Ropke and Cordeau [6] in which four types of decision variables are used. The paper also presents a Genetic Algorithm (GA) and a Tabu Search (TS) method to find the neighbourhood of all nearest materials required to carry to the patients, their respective pick-up (characterized as D1 and D2) and their treatment at the hospital. The limitation of this method is in effectively defining the stopping criteria for the Genetic Algorithm.

Li and Lin [7] propose a metaheuristic for the PDPTW problem. The main idea is to use a simulated annealing (SA) method which takes into account a current best solution and iterates upon a set of parameters to arrive at a non-improving best solution. The improvement in this paper is that the goods must be collected at a predetermined specified customer location. Hence two additional side constraints called precedence constraints and coupling constraints are introduced in this paper, which require that *"any paired pickup* *and delivery locations must be serviced by the same vehicle and the pickup location must be scheduled before the corresponding delivery location in the route"*. The authors argue that SA algorithm used in this paper is improved upon such that the simulated annealing procedure restarts from the current best solution after K simulated annealing iterations without any improvement.

Bent and Hentenryck [8] describe a two stage hybrid algorithm for the PDPTW. The first stage of the algorithm uses a simple simulated annealing algorithm to decrease the number of routes, while the second stage uses Large Neighborhood Search (LNS) to decrease total travel cost. The overall structure of the algorithm is motivated by the recognition that minimizing the objective function directly may not be the most effective way to decrease the number of routes in vehicle routing problems. The authors argue that the objective function often drives the search toward solutions with low travel cost, which may make it difficult to reach solutions with fewer routes but higher travel cost. To overcome this limitation, their algorithm divides the search in two steps: (1) the minimization of the number of routes and (2) the minimization of total travel cost using LNS. This two-step approach makes it possible to design algorithms tailored to each sub-optimization. They also answer positively the open issue in the original LNS paper, which advocated the use of LNS for the PDPTW and argue for the robustness of LNS with respect to side constraints. However, a solution to robustness under uncertainty was not discussed as the scope of the paper and the SA algorithm becomes useful only when the relationship between customers and the evaluation function is explicitly defined.

Jin et. al [9] proposed a particle swarm optimization for the VRPPDTW, which is a generalization of three existing time window formulations. A random key-based solution representation and decoding method is proposed for implementing (PSO) Particle Swarm Optimization for the problem, which means that the optimal solution is arrived at by iteratively trying to improve the candidate solution. The solution representation for the problem with n customers and m vehicles is a (n+2m)-dimensional particle. The decoding method starts by transforming the particle to a priority list of customers to enter the route and a priority matrix of vehicles to serve each customer. The vehicle routes are constructed based on the customer priority list and vehicle priority matrix. The particle search algorithm (PSO) is used to solve the VRPSPD, which consists of designing a set of at most *m* routes such that

- Each route starts and ends at the depot;
- Each customer is visited exactly once by exactly one vehicle; and
- The total vehicle load in any arc does not exceed the capacity of the vehicle assigned; the total duration of each route (including travel and service times) does not exceed a preset limit *D*; and
- The total routing cost is minimized.

The limitation of this method is that the metaheuristics such as PSO do not guarantee an optimal solution

A lot of advances have been made of which some of the advancements in the methods have been detailed so far. Solomon [10] described the various algorithms used in practice for scheduling problems with time window constraints by using Approximation algorithms. Christofides, Mingozzi and Toth [14] discuss state space relaxations for dynamic programming approaches to the traveling salesman problem with time windows, while Baker [12] and Baker and Rushinek [13] present a branch-and-bound algorithm for a new, time-oriented formulation of the problem. Developments have been made by Swersey and Ballard [15] where they discuss an optimal approach to this problem with the time window discretized. Desrosiers, Soumis and Derochers [16] developed exact methods for this problem. One algorithm uses a column generation approach in which the columns are generated by using a shortest-path-with-timewindows algorithm. Two other branch-and-bound algorithms involve relaxations of the time-window-related constraints. The nature of the heuristics are described as :

(a) Savings heuristics: This procedure begins with n distinct routes in which each customer is served by a dedicated vehicle. The parallel version of this tour-building heuristic is characterized by the addition at every iteration of a link of distinct, partially formed routes between two end customers, guided by a measure of cost savings

(b) Time oriented, Nearest neighbor Heuristic: Belongs to the class of sequential, tour-building algorithms. The nearest-neighbor heuristic starts every route by finding the unrouted customer "closest" (in terms of a measure to be described later) to the depot. At every subsequent iteration, the heuristic searches for the customer "closest" to the last customer added to the route. This search is performed among all the customers who can feasibly (with respect to time windows, vehicle arrival time at the depot, and capacity constraints) be added to the end of the emerging route. A new route is started any time the search fails, unless there are no more customers to schedule. The metric used in this approach tries to account for both geographical and temporal closeness of customers

(c) Insertion heuristics: Belongs to a class of sequential, tour-building heuristics initializes every route using one of several criteria to be described later After initializing the current route, the method uses two criteria, *cl(i, u, j)* and *c2(i, u, j),* at every iteration to insert a new customer u into the current partial route, between two adjacent customers *i* and *j* on the route

(d) Removal heuristics

(e) Time Oriented Sweep Heuristic: This heuristic can be viewed as a member of a broad class of approximation methods that decompose the problem into a

> clustering stage and a scheduling stage. In the first phase, customers are assigned to vehicles as in the original sweep heuristic. In the second phase, the authors create a one-vehicle schedule for the customers in this sector, using a tour-building heuristic. Due to the time window constraints, some customers in this cluster could remain unscheduled. After eliminating scheduled customers from further consideration, the clustering-scheduling process is repeated. The intuition for partitioning the unscheduled customers in the sector into two subsets is that the customers in the more clockwise half-sector will be relatively far away from the new cluster. By inserting these customers at a later stage, a better schedule may be created and the process is repeated until all customers have been scheduled.

The main drawback identified is the definition of the parameters that would lead to scheduling of the customers. It is quite difficult to quantify the nature of the clusters since there is a possibility that a local solution is continuously stuck.

Mingyong [17] found that an optimally integrating forward (good distribution) and reverse logistics (returning materials) for cost saving and environmental protection is the key for the problem at hand and thus proposed an improved differential evolution algorithm as a mixed integer programming model. The objective function seeks to minimize total distance traveled. Constraints ensure that each customer is visited by exactly one vehicle; and also guarantee that the same vehicle arrives and departs from each customer it serves. The author also defined restrictions that at most k vehicles are used restriction and are flow equations for pick-up and delivery demands, respectively constraints establish that pick-up and delivery demands will only be transported using arcs included in the solution. Finally, time windows constraints and maximum distance constraints are used. A fitness value is proposed based on total length of the route and the corresponding offspring replaces the parent chromosome in terms of best "fitness". Like the GA, it has difficulties in initialization and computational time complexity.

Catay [18] proposed an ant colony optimization (ACO) population-based metaheuristic. ACO applied to the Vehicle Routing Problem with Pickups and Deliveries (VRPPD) determines a set of vehicle routes originating and ending at a single depot and visiting all customers exactly once. The vehicles are not only required to deliver goods but also to pick up some goods from the customers. The objective is to minimize the total distance traversed. The author first provides an overview of the ACO approach.

Pankratz [19] proposed a grouping genetic algorithm which features a group-oriented genetic encoding in which each gene represents a group of requests instead of a single request. Till that time, very few approaches existed which applied Genetic Algorithms to variants of the time windows problem, most of them treating simplified special cases of the time windows problem. Probably the major reason why Genetic Algorithms for solving the PDPTW are rare is the fact that it is very difficult to find an appropriate genetic representation for this complex problem. The population size, *npop*, is a parameter of the group genetic algorithm (GGA). Unlike the classical GA, which employs generational replacement, the author mentioned that the population management is done following the steady-state approach without duplicates. According to this approach, each newly generated pair of offspring is inserted immediately into the current population where it replaces the two worst individuals. This incremental approach ensures the survival of the best solution over the whole search and prevents the occurrence of duplicate individuals.

Two main drawbacks are identified in this model: 1) Detection of duplicates is very complex. To alleviate the problem of detecting duplicates, a simple comparison of objective values proved satisfactory for the proposed GGA, so it was preferred over any other time consuming procedure seeking genotypical or phenotypical differences between individuals. The crossover operator is applied to each selected pair of parent chromosomes with probability *pcross*, whereas the mutation operator is applied to each offspring with probability *pmut*. Both *pcross* and *pmut* are parameters of the GGA. 2) Reproducibility of the parent in the child: If the crossover GGA for the PDPTW operator is not applied according to its execution probability *pcross*, the children are simply clones of their parents. Similarly, if the mutation operator is not applied, the offspring leave the mutation operator unchanged. The search terminates after a given total number, ˜*nmax*, of individuals has been generated without improvement but no later than after a given maximum number, *nmax*, of generated individuals has been reached.

The adaptation of the described genetic search scheme to the problem at hand involves the following components of the algorithm:

– The genetic encoding, i.e. the way solutions to the problem are represented by chromosomes (strings);

– The genetic operators, i.e. selection, crossover and mutation;

– The embedded heuristic, i.e. the subordinate heuristic procedure that is employed by the GGA in order to generate an initial population and to produce feasible offspring.

2.2 Branch-Cut-Price approaches, Insertion heuristics and Large Scale Neighbourhood Search methods for a vehicle routing problem with delivery and pickup with time windows

There were several new solution approaches implemented that use the concept of branch-and-price. Ropke et.al. [28] brought extensively delved in the pickup and delivery problem with time windows. The main principle is to formulate using three mathematical models for the PDPS and a branch-and-cut-and-price algorithm to solve it. The pricing sub-problem, an Elementary Shortest Path Problem with Resource Constraints

(ESPPRC), is solved with a labeling algorithm enhanced with efficient dominance rules. Three families of valid inequalities are used to strengthen the quality of linear relaxations.

Cortés et al. [31] present a mathematical formulation of the problem that is solved using a branch-and-cut algorithm. Scenarios with six requests and two vehicles are solved, where every request can be split and transferred from one vehicle to another at every node of the problem.

Gauvin, Desaulinears and Gendrau [20] propose a branch-cut-and-price algorithm with stochastic demands. The model of Christiansen and Lysgaard is adapted and formulated as a set partitioning model with additional constraints. Some important assumptions made in the paper is that a) goods are divisible and are all collected (or delivered) along routes.

b) Routes designed a priori must be feasible on average, i.e., the cumulative expected demand must not exceed the vehicle capacity.

c) Demands are independent, follow an additive probability distribution and have a positive expected value less or equal to the vehicle capacity. The demands are independent and follow well- known distributions such as the Normal or Poisson distributions. Feasible routes are generated using a dynamic programming algorithm executed over a state space graph. The method combines 2-cycle elimination with ng-routes. In addition, the pricing problem is significantly accelerated by the introduction of an aggregate dominance rule. Tabu search heuristic and a bi-directional labeling algorithm is also used in this paper. The authors also add capacity and subset-row inequalities dynamically in order to strengthen the linear relaxation of the master problem.

Desaulniers, Lessar and Hadjar [21] proposed the Tabu search, Partial Elementarity and Generalized k-Path Inequalities for the Vehicle Routing Problem with Time Windows. The approach is to develop a tabu search heuristic for the problem that allows the generation of negative reduced cost columns in a short computation time.

Second, to further accelerate the subproblem solution process, it is proposed to relax the requirements for a subset of the nodes. This relaxation, however, yields weaker lower bounds. Third, a generalization of the k-path inequalities and highlight that these generalized inequalities can, in theory, be stronger than the traditional ones. For the VRPTW, two main research streams were recently explored. First, dynamic programming algorithms for solving the ESPPRC were developed and improved. These algorithms, which can be relatively efficient for some of the difficult instances, offer the possibility of modeling the subproblem as an ESPPRC in branch-and-price algorithms, yielding tight lower bounds. On the other hand, they can still be impractical for very hard-problems that can occur almost at any iteration of the column generation process. Second, valid inequalities for the VRPTW were introduced to strengthen its formulation. These cutting planes have been used in branch-and-price methods and, more recently, in branch-and-cut methods that rely on a compact (non-decomposed) formulation of the problem.

To avoid as much as possible having to solve very difficult subproblems using dynamic programming, introduction of a simple tabu search algorithm that succeeds in rapidly generating negative reduced cost columns most of the time. Second, to further accelerate the subproblem solution process, relaxing the elementarity requirements for a subset of the nodes leads to a more optimal and fast solution. The resulting subproblem, called the partially elementary shortest path problem with resource constraints, offers a compromise between the difficulty of solving the problem and the quality of the lower bounds.

Potvin and Rosseau [22] proposed one of the earliest Tabu search heuristics for the vehicle routing problem with Backhauls and Time Windows with a greedy insertion heuristic and 2 opt procedure, which is derived from Solomon's work on the Vehicle Routing Problem with Time Windows (VRPTW), which describes many different route construction heuristics for the VRPTW. Among these, an insertion heuristic called II provided the best results. Heuristic II constructs the routes one by one. At the start, a "seed" customer is selected to create the first route. That is, the initial route only services this customer (i.e., the vehicle leaves the depot, services the seed customer and comes back to the de pot). Then, the remaining customers are inserted one by one in this route until it is full with respect to the capacity or time window constraints. At this point, a seed customer is selected to create a second route, and this route is filled again with the remaining unrouted customers. The procedure is repeated until all customers are serviced. At each step, the next customer to be inserted, as well as its insertion place within the current route, must be chosen. The best place for inserting a given customer u between two consecutive customers i and j in the route is obtained by minimizing a weighted sum of detour and service delay at customer j over all feasible insertion places. The next customer to be inserted is the one that maximizes a generalized savings measure.

Dumas and Desrosiers [23] propose an exact algorithm for PDPTW which can handle multiple depots and different types of vehicles. This algorithm works well for problems for which the demand at each customer is large, i.e., when the capacity constraints are restrictive. This algorithm uses a column generation scheme with a constrained shortest path as a subproblem. This algorithm can handle multiple depots and different types of vehicles.

Lu and Dessouky [24] propose a new insertion based heuristic for the VRPTW. Procedure differs from the classical insertion methods in two aspects. First, the classical insertion methods typically choose the next insertion by selecting a feasible insertion that has the minimal increase in travel distance or time, with respect to both the time window and capacity constraints. They do not directly take into consideration the degree of feasibility when determining which node and location to insert next. This characteristic prevents the insertion-based heuristic from constructing higher quality solutions, especially when more restricted feasibility constraints are considered such as time window constraints.

To overcome this characteristic, the authors discuss a new insertion evaluation function, which takes into consideration the increase of travel time as well as the reduction in the slack in the time window due to the insertion operation. The parameter is referred to the time difference between the time window and the service time as the slack in the time window. For example, instead of always choosing the node and location with the lowest cost as the next insertion, it may be better to select an insertion, which does not use much of the available slack so that more opportunities are left for future insertions. Second, in practice, operational planners tend to prefer more visually attractive solutions. This has been observed and confirmed by researchers who have implemented commercial routing software for industry. Their work reveals that more visually attractive solutions tend to have less total length of distance.

2.3 Robust Vehicle Routing Problem with Stochastic Demands and Uncertainty

Agra et al. [26] have addressed the robustness in VRPTW and have proposed two new formulations. The first formulation extends the well-known resource inequalities formulation by employing adjustable robust optimization. They propose two techniques, which, using the structure of the problem, allow to reduce significantly the number of extreme points of the uncertainty polytope. The second formulation generalizes a path inequalities formulation to the uncertain context. The uncertainty appears implicitly in this formulation, so that the authors develop a new cutting plane technique for robust combinatorial optimization problems with complicated constraints. The classical approach for robust programming relies on static models where the variables of the problem are not allowed to vary to account for the different values taken by the uncertain parameters. The travel times are not known with precision and belong to an uncertainty set. So the routes proposed for the ships are feasible in most situations. The main formulation is often called an adjustable robust program.

Agra et al. [27] also proposed a primitive version of the Robust formulation which is called as the Layered Formulation for the Robust Vehicle Formulation. They argue that the two stage formulation presents better results in comparison. They also state that the only known work in robustness of VRPTW assumption leads to all travel times taking their maximum values, which is an over-conservative model. In fact, it mainly focus on the robust capacitated vehicle routing problem. They consider the dualization approach to the robustness formulation.

Laporte et al. [29] provides an Adaptive Large Scale Neighbourhood Search Heuristic for Capacitated Arc-Routing Problem with Stochastic Demands. In this paper, the authors propose to minimize an expected cost of a solution with the help of heuristic namely, Large Neighbourhood (obtained via Shaw removal and insertion heuristics), Adaptive Search by weighted insertion and removal heuristics and the corresponding weight and score adjustments given for a penalty objective function (a pseudocode is provided in the Algorithm section). This method serves as the closest possible computational approach to estimating solutions under demand uncertainty in capacitated Arc Routing problems.

Marla [30] proposes a novel modeling framework called Decomposition Approach for Commodity Pickup and Delivery with Time Windows under Uncertainty. The decomposition approach follows a sequential process of network preprocessing by labeling the time windows. The next step is the Flow Master problem, which generates a set of cost-minimizing (but possibly schedule-infeasible) routes for shipments. The Scheduling Sub-problem checks if the routes that are output from the Flow Master Problem are schedule-feasible with respect to the time-windows. If the routes are schedule-infeasible, the subproblem generates a set of constraints to eliminate the infeasible solution(s). This process iterates between the Flow Master problem and Scheduling Sub-problem until a schedule-feasible solution is found or no solution exists. To make the iterations more efficient, the authors also use the notion of cliques. The approach states that when the algorithm terminates, we always find a feasible solution. The solution contains feasible schedules for the routes; and moreover, all possible feasible schedules are found. To find feasible solutions more easily, warm start procedures with solutions used by the carrier (even if partially infeasible) can be used. The algorithm, even if terminated midway, can still help generate solutions, though far from optimal.

CHAPTER 3 MODELLING APPROACH

In this chapter we first describe the problem and side constraints in greater detail. We then present an existing approach to this problem, proposed by Tardy [3], which focuses on minimizing the route-related costs. We then discuss our modeling approach, which can be created using a construction heuristic or by using Tardy's solution [3] as a warm-start solution and improving upon it using a set of improvement heuristics.

3.1 Description of the problem, dataset and model

We model the VRPPDTW problem along with the side constraints related to the carrier, on a network. Each node on the network is one of the following: the location of a domicile (start point of the tractors/vehicles), a pickup point for a shipment (trailer) or a delivery point for a shipment (trailer), or a transfer location where vehicles (tractors) can exchange shipments. Arcs connect nodes between which travel is possible. For each shipment that is to be picked up and dropped off, are associated an earliest arrival time of the load (EAT) and a latest delivery time (LDT). Tractors can be stationed at domiciles and should return to the domicile after executing their routes. Each load or shipment is denoted as short or long,

The dataset from the real-world operator of interest, contains 98 locations. 498 arcs represent possible connections between these locations, for which the distances and mean travel times are given. Among the locations, 6 are domiciles from which trailers should begin and end their routes. There are 347 loads that need to be transported between their origin and destination, within a 2-day time horizon. Among these 102 are long loads and 245 are short loads.

In order to describe the side-constraints, we first define the following notation.

a. *locID*: ID of the location
b. *toIn* : time to enter the location if we have a tractor without a trailer
c. *toOut*: time to depart at the location if we have a tractor without a trailer
d. *sIn*: time to enter the location if we have a tractor with one trailer
e. *sOut*: time to depart at the location if we have a tractor with one trailer
f. *dIn*: time to enter the location if we have a tractor with two trailers
g. *dOut*: time to depart at the location if we have tractor with two trailers
h. *doWash*: equals Y if we have to wash the tractor at the location; N otherwise;
i. *washTime*: duration of the washing

The loads are correspondingly given a trlr Type ID and also whether it could be included as short or Long. Overrides correspond to movements which are made at a particular location (moving trailers inside a facility). Additional parameters include (a). *isOverride*, which indicates whether there is some override time as a requirement for the load. (b). *overrideTime*, which is the length of the override and (c). *trlrTypeID*, ID of the trailer associated with the load.

The time frame in which the schedule is developed corresponds to a two day period. Additionally, the following parameters help define the work rules:

a. *domID*: ID of the domicile
b. *locID*: ID of the location corresponding to the domicile
c. *minDrivers*: minimum number of drivers located at the domicile
d. *maxDrivers*: maximum number of drivers located at the domicile
e. *maxDay*: max number of hours which can be worked in a day
f. *minDay*: min guaranteed number of hours paid
g. *sw*: time needed for starting the work
h. *fw*: time needed for finishing the work

i. *unpaidB*r: amount of non-paid time for the breaks

j. *paidBr*: amount of paid time for the breaks

Work rules are not homogenous within domiciles. There is a specified guaranteed paid day denoted *minDay* and a maximum legal work day denoted *maxDay*. Any work beyond *minDay* will be considered as over time work, and there is a maximum of *maxDay-minDay* of possible overtime.

Slack time *sw* represents the elapsed time between the arrival of the driver at work and the time the driver actually start working. The second slack time denoted *fw* equals the elapsed time between the moment the driver finishes work and the time at which the driver leaves work. Finally, there is at least 1 hour of non-paid break, denoted by *unpaidBr*, and 0.17 hour of paid break, denoted by *paidBr*.

3.1.1 Description of the breaks and Feasibility Function

The allowable break combinations are:

a. *B1Early*: minimum elapsed time of the route at which the first break can occur

b. *B1late*: maximum elapsed time of the route at which the first break can occur

c. *B1Dura*: duration of the first break

d. *B3Early*: minimum elapsed time of the route at which the third break can occur

Similarly, we have *B2Early, B2Late* and *B2Dura* as the corresponding elapsed time for the second break. There are three breaks namely, first second and third breaks which have a specified duration.

The cost function is slightly modified to incorporate the feasibility to include the loads in a particular route. The parameters were condensed to variables x_1 and x_2 to include the feasibility of the problem (i.e. how many loads can be carried on a particular route taking into account the slack and all the above parameters described above). The parameters that were combined include:

a. Cost per mile in tractor only configuration
b. Cost per mile in Single and Double configuration
c. Cost per hour of regular hours in Tractor only, Single and Double configurations
d. Cost per hour overtime in Tractor only, Single and Double configurations

x_1 is defined to be a boolean variable, which gives 1 if the load j is transported on leg i in the route in trailer position 1 and 0 otherwise. Similarly, x_2 is defined to be a Boolean variable, which gives 1 if load j is transported on leg i in the route in trailer position 2 and 0 otherwise. We also define time variables t_1 and t_2 which are also Boolean variables corresponding to the fact that if load j is carried in position 1 (or position 2 respectively) on leg i, which in turn, starts on day k.

3.2 Description of the cost function

The objective of the thesis is to describe a framework that helps characterize the features that lead to robustness. The primary objective is to minimize the total schedule cost and the secondary objective is to examine the level of robustness of the obtained solution(s) under stochasticity in the demand and service time parameters. In order to solve a large combinatorial problem with the use of real world data, we first characterize the total schedule cost as the sum of the costs of all the routes included in the schedule. The total schedule cost might be reasonably approximated as a linear function of the number of its drivers with the reasoning that the main component of the cost comes from the wages of the drivers and that the paid driver time is roughly the same for each driver. Paid time per driver is typically between 8 and 11 hours and if overtime, most drivers are paid at the double rate. True costs are then used after generating a solution to compute the actual cost of the schedule.

3.3 Model proposed by Tardy [3]

Tardy [3] builds the routes that the options for building the routes were through column generation by employing the cutting stock problem approach and through the pricing problem approach. A route can be described as the set of loads it carries. So assembling loads into routes and cutting smaller width rolls from a larger width roll in the cutting stock problem can be thought of in the same way. Just as there is a pricing problem for the cutting stock problem to find patterns, a pricing problem is solved to assemble loads into a route.

The routes to be present in a solution are chosen by solving the master problem

$$Minimize \sum_{j=1}^{n} x_j \qquad (3.1)$$

$$subject\ to,\ \sum_{j=1}^{n} a_{ij} x_j \geq 1,\ \forall\, i \in \{1 \dots m\} \qquad (3.2)$$

$$x_j = 0,\ \forall\, i \in \{1 \dots m\}, \forall\, j \in \{1 \dots n\}$$

Where n represents the number of routes generated, m is the number of loads; a_{ij} equals 1 if load i is carried by route j and equals 0 otherwise; and x_j is a decision variable equal to 1 if load j is included in the solution and equals 0 otherwise. This is often referred to as a set covering problem.

The initial set of routes is generated by solving a shortest path problem and assigning each load to the shortest cost route. However, this results in a very large number of tractor routes, which we would like to minimize. Therefore, the column generation approach adopts a pricing problem. Thus upon formulation, we have:

$$Maximize \sum_{i=1}^{m} p_i\, a_{\text{i}}, \qquad (3.3)$$

where p_i represents the dual value associated with the constraint i, which will define a route feasibility condition as a function of the loads it contains; a_i equals 1 if load is transported in the route and equals 0 otherwise. The approach is that the author tries to bring about a sequential approach to solve the pick-up and delivery problem by using clusters (compilations) of loads and hence we can better determine which of the choice of loads can be transported (as opposed to a single choice of loads) at the same time.

The concept of time windows is used, which is defined by the earliest departure times and latest arrival times. The objective is to come up with a series of optimal routes. By forming clusters, we can create instances where we determine which combinations of loads are feasible.

3.3.1 Pricing Problem Model

The pricing problem aims to generate a set of feasible routes to add to the Master Problem. Because of the side-constraints, the schedule for a set of routes can be extremely cumbersome to generate within the Master Problem. Therefore, the pricing problem considers a set of loads and examines if they can be fit into one single route in a schedule feasible manner. The Pricing Problem, thus, determines the feasibility of a given set of loads to travel in a single route and generates a feasible schedule for these loads, thus generating one feasible route. This route can then be used as a part of the Master Problem to find the best way to cover all loads.

A route is defines as “a succession of legs in which the first leg starts as a domicile and the last leg ends at the same domicile” (closed path). The leg takes into account all the information (time windows, distance, etc.) of the loads transported. The number of legs is decided to be 7 beforehand and is called as $nVar$, numbered chronologically. The following key rules are to be observed:

1. If the tractor hauls one short load, it is assigned to any trailer position
2. If the tractor hauls two short loads, then both trailer positions are occupied.
3. If the tractor hauls a long load, it is assigned by default to trailer position 1 and trailer position 2 is blocked and unable to accommodate another load.

With the conditionality that if a break occurs between any two legs, it could be assigned to either one of them and the load remains the same throughout the journey.

1. A route is consistent in time (departure should precede arrival in the same leg and arrival of the previous leg should precede departure of the next leg)
2. A route is consistent in space (arrival of previous leg is same as departure of next leg)
3. Removing one or more load to a route creates a feasible route, since it is independent of the truck configuration.
4. The concept of leading load limits the number of legs considered in generating the optimal routes. If both trailer positions are available (truck in Tractor Only configuration) then the driver must pick up at least one load at the next leg. If there is exactly one trailer position available (truck in Single configuration), the leading load is either the load currently being carried. If there is no trailer position available the destination of a given leg must correspond to the destination of one of the loads transported on that leg.

3.3.2 Description of the range of sets and Objective function

The general notation used in the Pricing Problem is described as:

1. $Loads$: set of loads
2. $Locations$: set of locations
3. $Domicicles$: set of domiciles
4. $Arcs$: set of arcs

5. $Legs$: set of legs in a route; $Legs = \{1 \ldots nVar\}$;

6. $\overline{Legs}$: set of legs in route excluding the last leg: $\overline{Legs} = \{1 \ldots (nVar - 1)\}$;

7. $\underline{Legs}$: set of legs in route excluding the first leg: $Legs = \{2 \ldots (nVar)\}$;

The objective function is described as

$$Maximize \sum_{j \in Loads} \left(xload_j^1 + xload_j^2 \right) \times DUAL_j \qquad (3.4)$$

We need to start off with a basic feasible solution to have initial values for $DUAL_j$, which can be done by assigning each load to a single route (which is highly suboptimal) or by using a warm-start technique from an existing feasible solution.

3.3.3 Description of the constraints involved in pricing problem

Similar to the Cutting Stock pricing Problem, routes should not violate any feasibility. Because, the number of constraints involved is large. So variables are defined and the necessary notations when needed and we aggregate constraints in different sets depending on their functions. The different sets of constraints are:

1. Shortest path;

2. Domicile;

3. Origin and destination of a leg;

4. Origin and destination of the leading load;

5. Choice of the leading load;

6. Pick-up and delivery;

7. Time in and out;

8. Work time;

9. Extra time;

10. Latest arrival and earliest available;

11. Break; and General Time

3.3.3.1 Shortest path constraints

Corresponding to the least travel time, the set of corresponding constraints are:

$timeLog_i \geq \sum_{k \in Arcs} arc_{i,k} \times TRAVELTIME_k, \forall\, i \in Legs$ (3.8)

$\sum_{k \in Arcs} arc_{i,k} \times ARCMATRIX_{n,k} = origin_{i,n} + dest_{i,n}, \forall\, i \in Legs, \forall\, n \in Locations$

(3.9)

where $TRAVELTIME_k$ is defined as the travel time along arc k; $ARCMATRIX_{n.k}$ equals -1 if location n is the origin of arc k, and equals 1 if location n if the destination of arc k; and equals 0 otherwise; $origin_{i,n}$ is a decision variable equal to -1 if the origin of leg i is location n and equal to 0 otherwise; $dest_{i,n}$ is a decision variable equal to 1 if the destination of leg i is location n and equal to 0 otherwise; $arc_{i,k}$ is a decision variable equal to 1 if arc k is present in the path of leg i and equal to 0 otherwise; and $timeLeg_i$ is a decision variable equal to the total travel time at leg i

3.3.3.2 Domicile constraints

$\sum_{i \in Domiciles} dom_i = 1$ (3.10)

$origin_{i,n} = -\sum_{i \in Domiciles} dom_i \times DOMICILE_{n,i}, \forall\, n \in Locations$ (3.11)

$dest_{nVar,n} = \sum_{i \in Domiciles} dom_i \times DOMICILE_{n,i}, \forall\, n \in Locations$ (3.12)

$\sum_{j \in Loads} x^1_{nVar,j} \times LOADIN_{n,j} \leq dest_{nVar,n}, \forall\, n \in Locations$ (3.13)

$\sum_{j \in Loads} x^2_{nVar,j} \times LOADIN_{n,j} \leq dest_{nVar,n}, \forall\, n \in Locations$ (3.14)

$\sum_{j \in Loads} x^1_{1,j} \times LOADOUT_{n,j} \leq origin_{1,n}, \forall\, n \in Locations$ (3.15)

$\sum_{j \in Loads} x^2_{1,j} \times LOADOUT_{n,j} \leq origin_{1,n}, \forall\, n \in Locations$ (3.16) where $DOMICILE_{n.i}$ equals 1 if domicile i corresponds to location n and equals 0 otherwise; $LOADOUT_{n.j}$ equals �1 if the origin of load j is location n and equals 0 otherwise;$LOADIN_{n.j}$ equals 1 if the destination of load j is location n and equals 0 otherwise; $x_{i,j}^{k}$ is a decision variable equal to 1 if load j is transported in trailer position k during leg i and equal to 0 otherwise; and dom_i is a decision variable equal to 1 if domicile i is the domicile of the route and equal to 0 otherwise.

3.3.3.3 Origin and Destination of leg and leading load: constraints

$$\sum_{n \in Locations} origin_{i,n} = -1, \forall\, i \in Legs \tag{3.17}$$

$$\sum_{n \in Locations} dest_{i,n} = 1, \forall\, i \in Legs \tag{3.18}$$

$$\forall\, n \in Locations, dest_{i,n} + origin_{i+1,n} = 0, \forall i \in \overline{Legs} \tag{3.19}$$

$$dest_{i,n} \geq \sum_{j \in Loads} leading_{i,j} \times LOADIN_{n,j}, \forall i \in Legs, \forall n \in Locations \tag{3.20}$$

$$-origin_{i,n} \geq \sum_{j \in Loads} leading_{i-1,j} \times LOADIN_{n,j}, \forall i \in \underline{Legs}, \forall n \in Locations \tag{3.21}$$

$leading_{i,j}$ is a decision variable equal to 1 if load j leads leg i and equal to 0 otherwise

3.3.3.4 Choice of the leading load: constraints

$$leading_{i,j} \geq x_{i,j}^{1} - choiceLeading_i, \forall\, i \in Legs, \forall j \in Loads \tag{3.22}$$

$$leading_{i,j} \geq x_{i,j}^{2} - 1 + choiceLeading_i, \forall\, i \in Legs, \forall j \in Loads \tag{3.23}$$

$$1 - choiceLeading_i \leq \sum_{j \in Loads} x_{i,j}^{1} + noLoad_i, \forall\, i \in Legs \tag{3.24}$$

$$choiceLeading_i \leq \sum_{j \in Loads} x_{i,j}^{2} + noLoad_i, \forall\, i \in Legs \tag{3.25}$$

$$\sum_{j \in Loads} leading_{i,j} \leq 1 - noLoad_i\ \forall\, i \in Legs \tag{3.26}$$

$$\sum_{j \in Loads} leading_{i,j} \geq \sum_{j \in Loads} x_{i,j}^{1}\ \forall\, i \in \overline{Legs} \tag{3.27}$$

$$\sum_{j \in Loads} leading_{i,j} \geq \sum_{j \in Loads} x_{i,j}^{2}\ \forall\, i \in \overline{Legs} \tag{3.28}$$

$$noLoad_i \leq 1 - \sum_{j \in Loads} \frac{(x_{i,j}^1 + x_{i,j}^2)}{2}, \forall\, i \in Legs \tag{3.29}$$

$$\sum_{j \in Loads} x_{i,j}^1 \leq 1 - noLoad_i\,, \forall\, i \in Legs \tag{3.30}$$

$$\sum_{j \in Loads} x_{i,j}^2 \leq 1 - noLoad_i\,, \forall\, i \in Legs \tag{3.31}$$

$$x_{i,j}^2 \leq 1 - \sum_{k \in Loads} x_{i,k}^1 \times LONG_k, \tag{3.32}$$

$$\sum_{j \in Loads} (x_{i,j}^1 + x_{i,j}^2) \leq 2,\ \forall\, i \in Legs \tag{3.33}$$

$$xload_j^i \leq \sum_{i \in Legs} x_{i,j}^1\ \forall\, j \in Loads \tag{3.34}$$

$$xload_j^2 \leq \sum_{i \in Legs} x_{i,j}^1\ \forall\, j \in Loads \tag{3.35}$$

$$xload_j^1 + xload_j^2 \leq 1\ \forall\, j \in Loads \tag{3.36}$$

$$x_{i+1,j}^1 \geq x_{i,j}^1 + choiceLeading_i - 1 - 2 \times bothFree_i \forall\, i \in \overline{Legs}, \forall j \in Loads \tag{3.37}$$

$$x_{i+1,j}^2 \geq x_{i,j}^2 - choiceLeading_i - 2 \times bothFree_i \forall\, i \in \overline{Legs}, \forall j \in Loads \tag{3.38}$$

$$bothFree_i \leq 1 + \sum_{j \in Loads} \left(x_{i,j}^1 - x_{i,j}^2\right) \times LOADIN_{n,j}\,, \forall\, i \in Legs, \forall n \in Locations \tag{3.39}$$

where $LONG_j$ equals 1 if load j is a long load and equals 0 otherwise; $choiceLeading_i$ is a decision variable equal to 1 if load in trailer position 1 can be the leading load at leg i and equal to 0 otherwise; $noLoad_i$ is a decision variable equal to 1 if there is no load transported on leg i and equal to 0 otherwise; $bothFree_i$ is a decision variable equal to 1 if trailer position trailer position 1 and 2 are free at the beginning of leg i; and $xload_j^k$ is a decision variable equal to 1 if load j is transported somewhere in the route in trailer position k and equal to 0 otherwise.

3.3.3.5 Pickup and delivery: constraints

$$\sum_{j \in Loads} x_{i,j}^1 \times LOADOUT_{n,j} \leq origin_{i,n} + 1 - pickup_i^1, \forall\, i \in Legs, \forall\, n \in Locations \tag{3.40}$$

$$\sum_{j \in Loads} x_{i,j}^{2} \times LOADOUT_{n,j} \leq origin_{i,n} + 1 - pickup_i^2, \forall\, i \in Legs, \forall\, n \in Locations \quad (3.41)$$

$$\sum_{j \in Loads} x_{i,j}^{1} \times LOADIN_{n,j} \leq dest_{i,n} - 1 + delivery_i^1, \forall\, i \in Legs, \forall\, n \in Locations \quad (3.42)$$

$$\sum_{j \in Loads} x_{i,j}^{2} \times LOADIN_{n,j} \leq dest_{i,n} - 1 + delivery_i^2, \forall\, i \in Legs, \forall\, n \in Locations \quad (3.43)$$

$$pickup_{i+1}^{1} \geq x_{i+1,j}^{1} - x_{i,j}^{1}, \forall\, i \in \overline{Legs}, \forall\, j \in Loads \quad (3.44)$$

$$pickup_{i+1}^{2} \geq x_{i+1,j}^{2} - x_{i,j}^{2}, \forall\, i \in \overline{Legs}, \forall\, j \in Loads \quad (3.45)$$

$$pickup_{i}^{1} \geq x_{i,j}^{1}, \forall\, j \in Loads \quad (3.46)$$

$$pickup_{i}^{2} \geq x_{i,j}^{2}, \forall\, j \in Loads \quad (3.47)$$

$$delivery_{nvar}^{1} \geq x_{nVar,j}^{1}, \forall\, j \in Loads \quad (3.48)$$

$$delivery_{nvar}^{2} \geq x_{nVar,j}^{2}, \forall\, j \in Loads \quad (3.49)$$

$$delivery_{nVar}^{1} \geq x_{i,j}^{1} - x_{i+1,j}^{1}, \forall\, i \in \overline{Legs}, \forall\, j \in Loads \quad (3.50)$$

$$delivery_{nVar}^{2} \geq x_{i,j}^{2} - x_{i+1,j}^{2}, \forall\, i \in \overline{Legs}, \forall\, j \in Loads \quad (3.51)$$

with $pickup_i^k$ is a decision variable equal to 1 if the load in trailer position k is picked up the beginning of leg i; and delivery i is a decision variable equal to 1 if the load in trailer position k is delivered at the end of leg i.

3.3.3.6 Time in and Time Out: constraints

$$nbLoadsInOut_i = \sum_{j \in Loads}\left(x_{i,j}^{1} + x_{i,j}^{2}\right), \forall\, i \in Legs \quad (3.52)$$

$$M \times moveInOut_i \geq timeLeg_i, \forall\, i \in Legs \quad (3.53)$$

$$toInOut_i \geq moveInOut_i - nbLoadsInOut_i \forall\, i \in Legs \quad (3.54)$$

$$2 \times toInOut_i \leq 2 - nbLoadsInOut_i, \forall\, i \in Legs \quad (3.55)$$

$$toInOut_i + sInOut_i + dInOut_i \leq 1, \forall\, i \in Legs \quad (3.56)$$

$$dInOut_i \geq -1 + nbLoadsInOut_i \; \forall i \in Legs \tag{3.57}$$

$$2 \times dInOut_i \leq nbLoadsInOut_i, \forall i \in Legs \tag{3.58}$$

$$M \times (toInOut_i + sInOut_i + dInOut_i) \geq timeLegs_i \; \forall i \in Legs \tag{3.59}$$

$$timeIn_i \geq \sum_{n \in Locations} dest_{i,n} \times TOIN_n - M \times (1 - toInOut_i) \; \forall i \in Legs \tag{3.60}$$

$$timeIn_i \geq \sum_{n \in Locations} dest_{i,n} \times SIN_n - M \times (1 - sInOut_i) \; \forall i \in Legs \tag{3.61}$$

$$timeIn_i \geq \sum_{n \in Locations} dest_{i,n} \times DIN_n - M \times (1 - dInOut_i) \; \forall i \in Legs \tag{3.62}$$

$$timeOut_i \geq -\sum_{n \in Locations} origin_{i,n} \times TOOUT_n - M \times (1 - toInOut_i) \; \forall i \in Legs \tag{3.63}$$

$$timeOut_i \geq -\sum_{n \in Locations} origin_{i,n} \times SOUT_n - M \times (1 - sInOut_i) \; \forall i \in Legs \tag{3.64}$$

$$timeOut_i \geq -\sum_{n \in Locations} origin_{i,n} \times DOUT_n - M \times (1 - dInOut_i) \; \forall i \in Legs \tag{3.65}$$

$$toInOut_i + toInOut_{i+1} \leq 1 \; \forall i \in \overline{Legs} \tag{3.66}$$

where M is the size of the largest time window, here roughly 50 hours; $TOIN_n$ is the time to enter location n when the truck is in Tractor Only configuration; SIN_n is the time for coming into location n when the truck is in Single configuration; DIN_n is the time for coming into location n when the truck is in Double configuration; $TOOUT_n$ is the time for coming out of location n when the truck is in Tractor Only configuration; $SOUT_n$ is the time for coming out of location n when the truck is in Single configuration; $DOUT_n$ is the time for coming out of location n when the truck is in Double configuration; $nbLoadsInOut_i$ is decision variable equal to the number of loads being transported between two different location at leg i; $moveInOut_i$ is decision variable equal to 1 if there is the leg's origin differs from its destination and equal to 0 otherwise; $toInOut_i$ is decision variable equal to 1 if the leg's origin differs from its destination and if the truck is in Tractor Only configuration and equal to 0 otherwise; $sInOut_i$ is decision variable equal to 1 if the leg's origin differs from its destination and if the truck is in Single configuration and equals 0 otherwise; $dInOut_i$ is decision variable equal to 1 if the leg's origin differs from its destination and if the truck is in Double configuration, and equal to 0 otherwise; *timeIn$_i$* is decision variable equal to the time needed to enter the location at leg *i*; and *timeOut$_i$* is decision variable equal to the time needed to exit the location at leg *i*.

3.3.3.7 Work Time: constraints

$$edpt_i - sw \geq 0 \tag{3.67}$$

$$sw = \sum_{j \in Domiciles} dom_j \times SW_j \tag{3.68}$$

$$fw = \sum_{j \in Domiciles} dom_j \times FW_j \tag{3.69}$$

where *SW$_j$* is the elapsed time for briefing at domicile *j* before a route can depart; *FW$_j$* is the elapsed time for debriefing at domicile *j*; *sw* is a decision variable equal to the elapsed time before a route can depart; and *fw* is a decision variable equal to the elapsed time before the driver's work day end.

3.3.3.8 Extra Time: constraints

$$washing_i \geq \sum_{n \in Locations} origin_{i,n} \times WASH_n - M \times (1 - moveInOut_i)\ \forall\, i \in Legs \tag{3.70}$$

$$override_i \geq \sum_{j \in Loads} \left(x_{i,j}^1 + x_j^2\right) \times OVERRIDETIME_j, \forall\, i \in Legs \tag{3.71}$$

$$extraTime_i = timeOut_i + timeIn_i + washing_i + override_i, \forall\, i \in Legs \tag{3.72}$$

where WASHn is the washing time at location n; OVERRIDETIMEj is the override time for load j; overridei is a decision variable equal to time spent in override at leg i; washingi is a decision variable equal to the time spent washing a vehicle after leg i; extraTimei is a decision variable equal to total extra times at leg i; and edpti is a decision variable equal to the earliest departure time from the origin of leg i:

3.3.3.9 Latest arrival and earliest Departure: constraints

$$edpt_i \geq \sum_{j \in Loads} x_{i,j}^1 \times EAVL_j, \forall\, i \in Legs \tag{3.73}$$

$$edpt_i \geq \sum_{j \in Loads} x_{i,j}^2 \times EAVL_j, \forall\, i \in Legs \tag{3.74}$$

$$larr_i \geq \sum_{j \in Loads} x_{i,j}^1 \times LARR_j, \forall\, i \in Legs \tag{3.75}$$

$$larr_i \geq \sum_{j \in Loads} x_{i,j}^2 \times LARR_j, \forall\, i \in Legs \tag{3.76}$$

$$larr_{nVar} - edpt_i \geq \sum_{i \in Domiciles} dom_i \times MAXDAY_i - fw - sw \tag{3.77}$$

$$extraTime_i + timeLeg_i \leq larr_i - edpt_i, \forall\, i \in Legs \tag{3.78}$$

where, EAT_j is the earliest available time for load j; LDT_j is latest arrival time for load j; $MAXDAY_i$ is the maximum working time for a route starting at domicile i; $edpt_i$ is a decision variable equal to the earliest departure time from the origin of leg i; and LDT_i is the decision variable equal to the latest arrival time at the destination of leg i.

3.3.3.10 Breaks: constraints

$$firstBreak_i^1 \leq breakChoice, \forall\, i \in Legs \tag{3.79}$$

$$mealBreak_i^1 \leq breakChoice, \forall\, i \in Legs \tag{3.80}$$

$$secondBreak_i^1 \leq breakChoice, \forall\, i \in Legs \tag{3.81}$$

$$firstBreak_i^2 \leq 1 - breakChoice, \forall\, i \in Legs \tag{3.82}$$

$$mealBreak_i^2 \leq 1 - breakChoice, \forall\, i \in Legs \tag{3.83}$$

$$secondBreak_i^2 \leq 1 - breakChoice, \forall\, i \in Legs \tag{3.84}$$

$$\sum_{i \in Legs} firstBreak_i^1 \leq breakChoice \tag{3.85}$$

$$\sum_{i \in Legs} mealBreak_i^2 \leq breakChoice \tag{3.86}$$

$\sum_{i \in Legs} secondBreak_i^1 \leq breakChoice$ (3.87)

$\sum_{i \in Legs} firstBreak_i^2 \leq 1 - breakChoice$ (3.88)

$\sum_{i \in Legs} mealBreak_i^2 \leq 1 - breakChoice$ (3.89)

$\sum_{i \in Legs} secondBreak_i^2 \leq 1 - breakChoice$ (3.90)

$M \times \sum_{i \in Legs}(firstBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_1^1, \forall i \in Legs$ (3.91)

$M \times \sum_{i \in Legs}(mealBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_4^1, \forall i \in Legs$ (3.92)

$M \times \sum_{i \in Legs}(secondBreak_j^1 + 1 - breakChoice) \geq edpt_i - edpt_1 - BREAK_7^1, \forall i \in Legs$ (3.93)

$M \times \sum_{i \in Legs}(firstBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_i^2, \forall i \in Legs$ (3.94)

$M \times \sum_{i \in Legs}(mealBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_4^2, \forall i \in Legs$ (3.95)

$M \times \sum_{i \in Legs}(secondBreak_j^2 + breakChoice) \geq edpt_i - edpt_1 - BREAK_7^2, \forall i \in Legs$ (3.96)

$noBreak_i = 1 - aBreak_i, \forall i \in Legs$ (3.97)

$\sum_{i \in Legs} firstBreak_i^1 \geq \sum_{i \in Legs} mealBreak_i^1$ (3.98)

$\sum_{i \in Legs} firstBreak_i^2 \geq \sum_{i \in Legs} mealBreak_i^2$ (3.99)

$\sum_{i \in Legs} mealBreak_i^1 \geq \sum_{i \in Legs} secondBreak_i^1$ (3.100)

$\sum_{i \in Legs} mealBreak_i^2 \geq \sum_{i \in Legs} secondBreak_i^2$ (3.101)

where, $BREAK_m^k$ describes the breaks for the combination k, now if

m = 1, it equals the minimum elapsed time of the route at which first break can occur;

m = 2, it equals the maximum elapsed time of the route at which first break can occur;

m = 3, it equals the duration of the first break;

m = 4, it equals the minimum elapsed time of the route at which the meal break can occur;

m = 5, it equals the maximum elapsed time of the route at which the meal break can occur;

m = 6; it equals the duration of the meal break;

m = 7, it equals the minimum elapsed time of the route at which the second break can occur;

m = 8, it equals the maximum elapsed time of the route at which the second break can occur; and

m = 9, it equals the duration of the second break;

$firstBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the first break occurs if the break occurs during leg *i* and if we have chosen break combination k, and equal to 0 otherwise; $mealBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the second break occurs if the break occurs during leg *i* and if we have chosen break combination k, and equal to 0 otherwise;

$secondBreak_i^k$ is a decision variable equal to the elapsed time of the route at which the second break occurs if the break occurs during leg *i* and if we have chosen break combination k, and equal to 0 otherwise; *breakChoice* is a decision variable equal to 0 if break combination 1 is chosen and equal to 0 if break combination 2 is chosen; *aBreak*$_i$ is a decision variable equal to 1 if there is a break occurring at leg *i* and equal to 0 otherwise; and *noBreak* is a decision variable equal to 0 if there is no break occurring at leg *i* and equal to1 otherwise.

3.3.3.11 Time: constraints

$$extraTime_i + timeLeg_i + firstBreak_i^1 \times BREAK_3^1 + mealBreak_i^1 \times BREAK_6^1 + secondBreak_i^1 \times BREAK_9^1 \leq larr_i - larr_{1-1}, \forall\, i \in \underline{Steps} \quad (3.102)$$

$$extraTime_i + timeLeg_i + firstBreak_i^2 \times BREAK_3^2 + mealBreak_i^2 \times BREAK_6^2 + secondBreak_i^2 \times BREAK_9^2 \leq larr_i - larr_{1-1}, \forall\, i \in \underline{Steps} \quad (3.103)$$

$$extraTime_{i-1} + timeLeg_{i-1} + firstBreak_{i-1}^1 \times BREAK_3^1 + mealBreak_{i-1}^1 \times BREAK_6^1 + secondBreak_{i-1}^1 \times BREAK_9^1 \leq edpt_i - edpt_{1-1}, \forall\, i \in \underline{Steps} \quad (3.104)$$

$$extraTime_{i-1} + timeLeg_{i-1} + firstBreak_{i-1}^2 \times BREAK_3^2 + mealBreak_{i-1}^2 \times BREAK_6^2 + secondBreak_{i-1}^2 \times BREAK_9^2 \leq edpt_i - edpt_{1-1}, \forall\, i \in \underline{Steps} \quad (3.105)$$

$$larr_i \geq firstBreak_i^1 \times (BREAK_1^1 + BREAK_3^1) + edpt_i, \forall\, i \in \overline{Legs} \quad (3.106)$$

$$larr_i \geq mealBreak_i^1 \times (BREAK_4^1 + BREAK_6^1) + edpt_i, \forall\, i \in \overline{Legs} \quad (3.107)$$

$$larr_i \geq mealBreak_i^2 \times (BREAK_4^2 + BREAK_6^2) + edpt_i, \forall\, i \in \overline{Legs} \quad (3.108)$$

$$larr_i \geq firstBreak_i^2 \times (BREAK_1^2 + BREAK_3^2) + edpt_i, \forall\, i \in \overline{Legs} \quad (3.109)$$

$$larr_i \geq secondBreak_i^2 \times (BREAK_7^2 + BREAK_9^2) + edpt_i, \forall\, i \in \overline{Legs} \quad (3.110)$$

$$larr_i \geq secondBreak_i^1 \times (BREAK_7^2 + BREAK_9^2) + edpt_i, \forall\, i \in \overline{Legs} \quad (3.111)$$

$$edpt_i \leq firstBreak_i^1 \times BREAK_2^1 + edpt_1 + (1 - firstBreak_i^1) \times M, \forall i \in Legs \quad (3.112)$$

$$edpt_i \leq firstBreak_i^2 \times BREAK_2^2 + edpt_1 + (1 - firstBreak_i^2) \times M, \forall i \in Legs \quad (3.113)$$

$$edpt_i \leq mealBreak_i^1 \times BREAK_5^1 + edpt_1 + (1 - mealBreak_i^1) \times M, \forall i \in Legs \quad (3.114)$$

$$edpt_i \leq mealBreak_i^2 \times BREAK_5^2 + edpt_1 + (1 - mealBreak_i^2) \times M, \forall i \in Legs \quad (3.115)$$

$$edpt_i \leq secondBreak_i^1 \times BREAK_8^1 + edpt_1 + (1 - secondBreak_i^1) \times M, \forall i \in Legs \quad (3.116)$$

$$edpt_i \leq secondBreak_i^2 \times BREAK_8^1 + edpt_1 + (1 - secondBreak_i^2) \times M, \forall i \in Legs \quad (3.117)$$

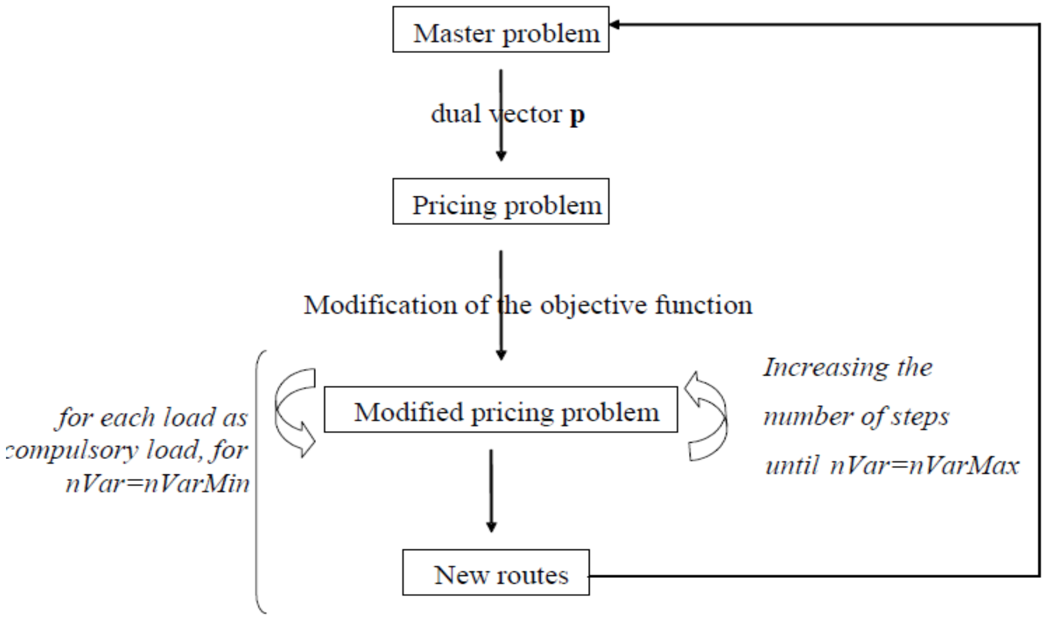

Master problem
dual vector **p**
Pricing problem
Modification of the objective function
for each load as compulsory load, for nVar=nVarMin
Modified pricing problem
Increasing the number of steps until nVar=nVarMax
New routes

Fig 3.1 Flowchart of solution process used by Tardy [3]

3.4 Our Modelling Approach

3.4.1 Break and Rejoin Heurisitic

The model explained in this thesis is based on break and rejoin type of heuristic. The solution obtained from Tardy [3] gave a minimum of 90 routes from the iterative framework or the pricing problem. However, the algorithm was terminated due to time constraints. Therefore, we believe a quicker heuristic can improve significantly upon the solution proposed by Tardy [3]

So in order to improve the solution starting from this starting solution, we came up with the break and rejoin type of heuristic where the first step is to break up the routes from Tardy's original solution and create sub-routes or clusters of loads. We then insert other loads into these sub-routes to create routes that we refer to as metaloads. We describe how we generated the metaloads.

For the set of loads in the cluster, we find the corresponding time windows (earliest possible arrival and latest possible departure) at each location using the matrix of the shortest paths, between any two locations in the network. We use the Djikstra's algorithm with arc costs as the travel times to find the shortest paths between each pair of locations. Because the time needed for traveling along an arc depends on the direction of travel, therefore we generate a non-symmetrical matrix of shortest path. The table of shortest paths will thus have the form as the first column denotes the origin and the first line the destination. We compute for each load a vector representing the earliest visit times and the latest visit times at every location in the network. Based on loads earliest availability and latest delivery time, the earliest visit time corresponds to the earliest time at which the load can be at a given location and the latest visit time corresponds to the latest time at which a load can be at a given location. We take into accounts the amount of time to enter and exit each location and compute the earliest and latest visit vectors at location X for load i as follows:

$$earliest(X) = EAT + d(X, origin) + Out(origin) + In(X) + Wash(X) \qquad (3.118)$$

$$latest(X) = LDT - d(X, destination) - Out(X) - In(destination) - Wash(destination) \qquad (3.119)$$

where EAT is the earliest availability of the load; LDT is the latest delivery time of the load; *earliest(X)* is the earliest visit time at location *X*; *latest(X)* is the latest visit time at location *X*; origin is the origin of the load; destination is the destination of the load; *d(X; Y)* is the shortest path in time between location *X* and location *Y* ; *In(X)* is the time to enter location *X*; *Out(X)* is the time to exit location *X*; and *Wash(X)* is the washing time at location *X*

Knowing the time windows (EAT, LDT) for a load and the shortest-path matrix, we can determine time windows for visiting all the other locations in the graph. We note that *In(X)* and *Out(X)* not only depends on the location but also on the configuration of the tractor. In a cluster, the tractor will never travels empty, hence, we only have to consider the Single and Double configurations. We do not know beforehand, however, what the configuration will be.

We construct clusters in a gradual manner. The first way is to start with a load which defines the first cluster and subsequently adding loads one by one. The second way is to cut the routes in the solution obtained by Tardy [3] into groups of loads that can travel together simultaneously. Note that because the capacity of the tractor is two trailers, the sub-routes obtained from these routes contain at most two loads. We then perform insertion heuristics to increase the size of the cluster and create the metaloads. The property of each metaload is such that it represents loads that are 'close' in time and space, with small gaps between the pickup and delivery load of another, or the loads can be simultaneously transported by a trailer. This restricts the number of combinations of loads in the insertion procedure. We perform two types of insertions, which we refer to as Type 1 and Type 2 insertions.

Denote load 1 as the existing set of loads (called pseudo load) of the cluster. A Type 1 insertion involves the addition of load 2 whose path is totally included in the path of the pseudo load. A Type 2 insertion involves the addition of load 2 whose path is not totally included in the path of the pseudo-load. Successive loads are added to the cluster on the basis of compatibility of loads which is checked in the following manner

The first thing we do is to check whether the load added to the cluster is compatible with the cluster. The conditions of compatibility differ with respect to the type of addition considered. Let P designate the pseudo-load; L designate the load to be added; *earliest(U;X)* designate the earliest visit time for load *U* at location *X*, *latest(U;X)* designate the latest visit time for load *U* at location *X*; *origin(U)* designates the origin of load *U* and *destination(U)* the destination of load *U*; and *d(X; Y)* designate the shortest path between location *X* and location *Y*.

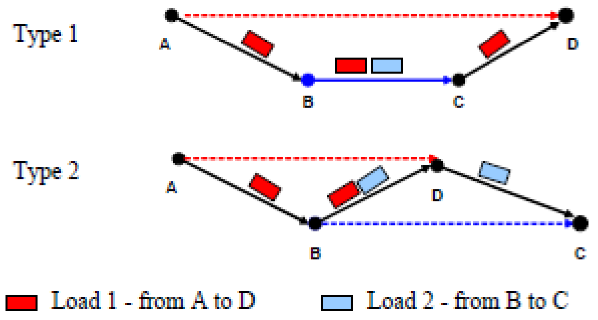


Figure 3.2 Types of insertion heuristics [3]

For a Type I addition, it must be ensured that:

$$earliest\big(P, origin(L)\big) \leq latest\big(L, origin(L)\big); and$$

$$earliest\big(P, destination(L)\big) \leq latest(L, destination(L)) \qquad (3.120)$$

Note that if the pseudo-load and the load to be added have the same origin, we compute the earliest visit vector with *In()* and *Out()* for a Double configuration. If the origins are different then we use the values for the Single configuration. Moreover, if the pseudo-load of the cluster and the load to be added have the same destination, we compute the latest visit vector with *In()* and *Out()* for a Double configuration; if the origins are different then we use the values for the Single configuration.

For a type II addition, it must be ensured that

$$earliest\big(P, origin(L)\big) \leq latest\big(L, origin(L)\big); and$$

$$\max\Big(earliest\big(P, destination(P)\big), earliest\big(L, destination(P)\big)\Big) +$$
$$d\big(destination(P), destination(L)\big) + Out\big(destination(P)\big) - In\big(destination(L)\big) -$$
$$Wash\big(destination(L)\big) \leq latest(L, destination(L)) \quad (3.121)$$

However, to avoid 'too long' clusters, given that multiple clusters are combined into a route, we restrict clusters to a maximum of seven loads. Note that in spite of time-related considerations being incorporated in the creation of the metaloads, they might still be feasible if the other side-constraints described in Section 3.3.3 are not satisfied. Therefore, for each of the clusters thus obtained, we perform a feasibility check on the cluster by running the Pricing Problem. That is, we check if the loads in the cluster have a feasible solution by running the constraints described in Section 3.3.3.

A simple approach is to construct an initial solution in which each route remains at a domicile during *nVar* legs. Although this initial basis indeed does provide an initial solution, it is far away from optimal. Another idea is to build routes in a gradual manner. We start from a small value of *nVar* for which we can easily find a good route with. Next, we translate this route into a set of constraints (which will be detailed subsequently) and we plug these constraints into a new model with a larger value of *nVar*. We increase *nVar* by one or two units and solve the model again.

In order to generate the desirable instance of metaloads, we can add a penalty term to the objective function and generate a set of metaloads, within which all loads lie in the same temporal neighbourhood. That is, once the solver has found a route, it can generate additional routes of interest to us by adding to the objective function the following penalty term.

$$\delta \sum_{i \epsilon Legs}(larr_i - \ edpt_i), \ \delta < 0 \ and \ |\delta| \ \ is \ small \qquad (3.122)$$

Using this process, 796 Metaloads were identified. Also, because the loads within each metaload are clustered closely in time, the Pricing Problem runs quickly for small values of *nVar*. Also, a particular load can be present in multiple metaloads, because the metaloads represent feasible combinations of loads. Hence the number of metaloads is higher than the number of loads. By the construction of the metaloads, the loads contained in a metaload are close to eah other temporally or spatially in terms of pickup and delivery times. Among the 796 metaloads, the first 36 metaloads contain 7 loads and the remaining 740 metaloads contain either one or two loads. We now combine these 796 metaloads using a graphical approach, as we describe next.

3.4.2 Combining Metaloads into Feasible Routes

In the previous subsection, we described how the metaloads are created. When we solve the set-covering Master Problem to cover all the loads, using the metaloads as routes, we see that more than a hundred routes are obtained. Therefore we see that the metaloads have to be further combined to create feasible routes that will result in a smaller number of total routes.

We model the combinations of the 796 metaloads as a graph network. Each of the nodes represent a metaload and we create an arc between two nodes if the two metaloads can be combined to form one route or sub-route with a feasible schedule. The first feasibility matrix is modeled as a graph network.

To construct this network, we need to determine the arcs in the network. Suppose you are trying to determine if there should exist an arc between nodes *i* and *j,* we solve the Pricing problem with a zero objective function with all the loads contained in the metaloads *i* and *j*. Once we determine the set of arcs in this graph, we then find the sets of metaloads that can be combined further to generate feasible routes. Figures 3.2, 3.3 and 3.4 describe the graph with nodes as metaloads and arcs describing route-and-schedule-feasible pairings of metaloads

On the metaload graph, to minimize the number of routes, we can find candidate routes by finding cliques on this graph. A clique or a strongly connected component of the graph is a set of nodes among which each pair has an arc present between them. Because each pair is a feasible combination, a clique presents a candidate for a route in which the loads present in all the nodes (metaloads) are present in the same route. Therefore, we now try to solve the k-clique problem on this graph.

We now consider the notion of a k-clique. A k-clique on this graph is a set of k nodes (represented by metaloads) in which each pair of nodes is schedule-feasible with each other. Ideally, the larger the size of cliques we can find, the more number of metaloads (and correspondingly, loads) that can be combined into one route. This will help minimize the number of routes. However the problem of finding and enumerating cliques in itself is NP-hard and the computation time required to compute 3- cliques is of the order of $O(3^{\frac{n}{3}})$ where n is the number of nodes present (796 in this case). Therefore, we restrict our search to 3-cliques in this network, for two reasons: (i) because of the involved complexity in enumerating all the cliques, (ii) because each metaload contains atleast two loads, the number of loads in a 3-clique will be at least 6, which is a large number ,from the routes observed .

From the graphs in Figures 3.3, 3.4 and 3.5, we can see that the metaload network is densely connected and it is very difficult to segregate the parts that are of interest for creating 3-clique combinations of metaloads. Figures 3.8 and 3.9 provide pictorial views of the 3-clique graph. The number of combinations that can arise is of the order of $796C_3$ and the computational time required to complete one iteration is one hour on a 20GB RAM Java-integrated IBM OPL model program, which is prohibitive. Hence we apply Randomized Algorithms to simplify the feasibility problem.

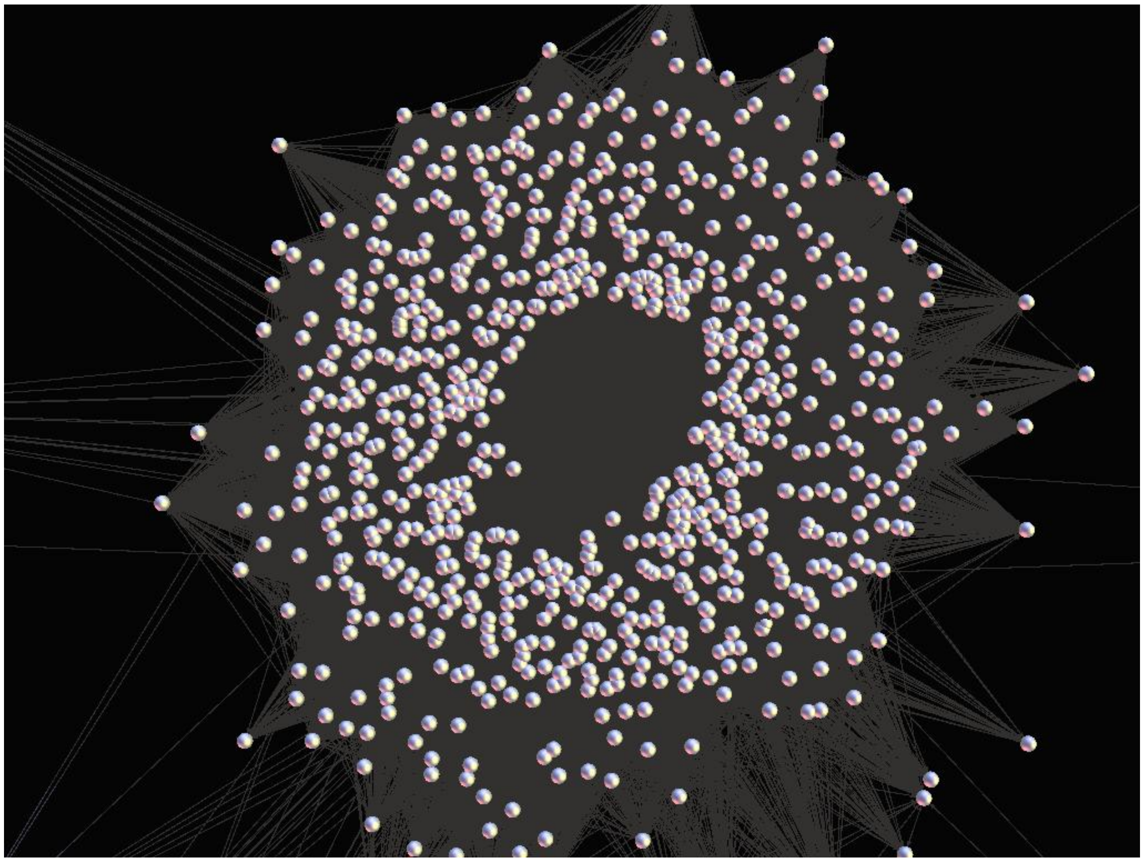


Figure 3.3: Expanded view of the metaload graph

Figure 3.4 Graph with node labels for the metaload graph

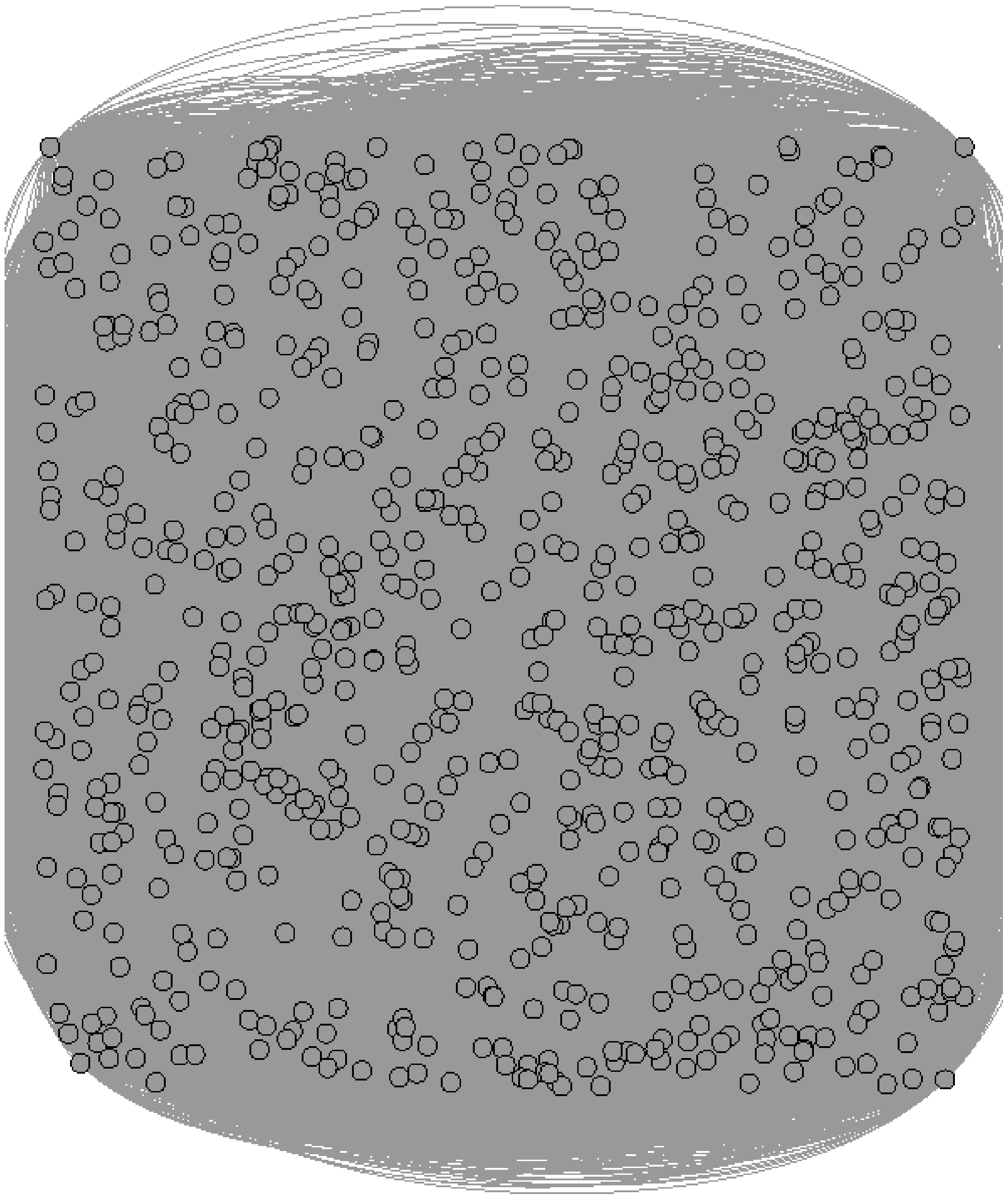


Figure 3.5 Condensed view with nodes for metaload graph

3.4.3 Randomized Algorithms for Graphs

A randomized algorithm is an algorithm that employs some randomness as a part of its construction. Because of the complexity and dense connectedness of the metaloads graph seen in Figures 3.3 – 3.5, we would like to use algorithms that can efficiently explore the cliques in the graph. Our randomized algorithm is inspired by Karger's algorithm for finding minimum cuts.

To illustrate we present Karger's algorithm [32], which is a randomized algorithm to compute the minimum cut of the graph. By sectioning (partitioning) the graph at strategic edges, we can condense the properties of the graph into the required subset. This is called as edge contraction, which is best described in the figure below.

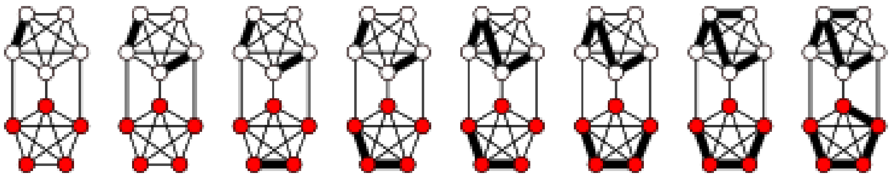


Figure 3.6: Successful spanning of Karger's algorithm [32]

Given a graph, we address the problem of how many edges have to be loosened (broken) before the graph breaks into two disconnected components. There are deterministic methods to finding the arcs in the minimum cut, however, we examine a randomized way of picking the min-cut, as follows.

1. Choose a link in the network (uniformly) at random.

2. Combine/merge the two hosts on either sides of this link, and remove any selflinks that result from this merge.

3. Repeat the above two steps till only two hosts are left in the network, and present the links between them as the Min-cut.

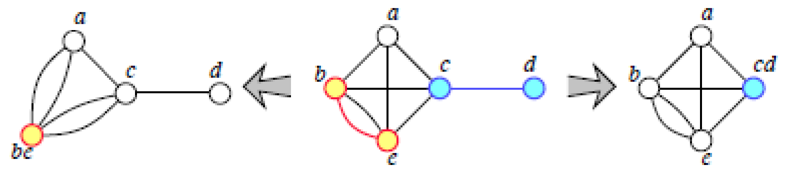


Figure 3.7 Step wise depiction of the Karger's algorithm [32]

Hochbaum [33] describes the randomized algorithm as *"An illustration of step 2 is shown in Fig 3.7. We notice that the set of edges presented in the final step are definitely a cut of the network (i.e. if you remove them the network will break into disconnected components). But there is no guarantee that it is going to be the cut with the smallest size (i.e. it is going to be a Min-Cut). Stated differently, we might have more edges in the cut presented in the last-step of the above procedure as compared to a Min-Cut of the network. If the random selection resulted in merging b and e the min-cut of the new graph is also a min-cut of the old graph. But, if we merged c and d, the min-cut of the resulting graph says nothing about the min-cut of the original graph."*

If we want to find the Min-Cut with probability (1 - €), then we run the above procedure $k \times n^2/_2$ times, where k > - ln €. That is, we can make the probability of failure as small as we like by running many copies of the procedure.

The Min-Cut algorithm presented above falls into a class of Randomized Algorithms called Monte Carlo Algorithms. The running-time of these algorithms are not random, but the solutions they provide are not always the optimal solutions. That is, the randomness in the procedure is not associated with running-time, instead it is associated with the quality of the solution. In contrast, the Randomized version of QuickSort falls into the class of Las Vegas Algorithms. With these algorithms, the randomness is associated with the running-time, but the solution is always guaranteed to be the best/optimal. A Las Vegas algorithm is called efficient if the average running-time is a polynomial in the input size. A Monte Carlo Algorithm is called efficient if the worst-case running-time on any instance is bounded by a polynomial in the input size.

In the context of our problem, to find the 3-cliques in the metaload network, we first try to enumerate the possible combinations with each metaload present. On an average, there are 100,000 combinations present, that is, each metaload (node) can be a part of about 100,000 3-cliques. Notice that due to the density of the graph, this is combinatorially explosive. However, also notice that not all these combinations will result in feasible routes and schedules. Therefore, we plan to examine those 3-cliques among the 100,000 that are more likely to generate feasible routes and schedules. In particular, we wish to choose 3-cliques such that each metaload is contained in a good number of these cliques, to ensure the presence of each load in some subset of feasible schedules. Note that for each metaload , a 3-clique forms a triangle with the metaload of interest. Suppose we are interested in generating cliques containing node (metaload) *i.* Therefore, we use Karger's algorithm to repeatedly condense parts of the network not containing node *i*, until only a small number of nodes are remaining. We then easily find cliques on this graph using a standard algorithm. We repeat this randomized procedure for each node (metaload) in the network. Because the graph is very dense, on an average we can reduce the 100,000 combinations of cliques that each metaload is a part of, to about 20,000. We then run the feasibility check constraints in Section 3.3.3 to identify schedule-feasible cliques among these 20,000. After completion of this procedure, we identify a total of 23,869 feasible 3-cliques as shown in Figure 3.8 and Figure 3.9.

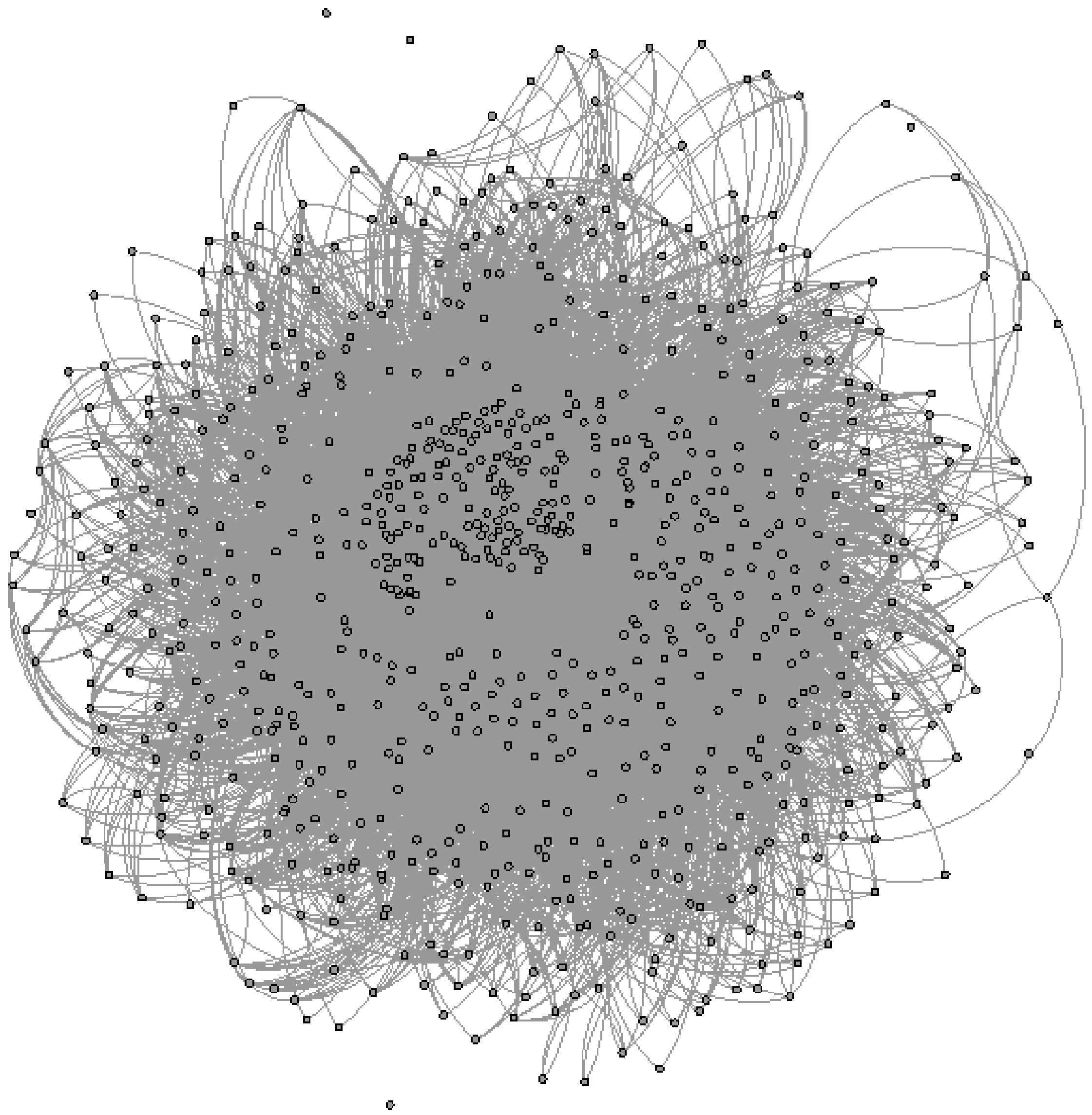


Fig 3.8 Three clique graph in the original form

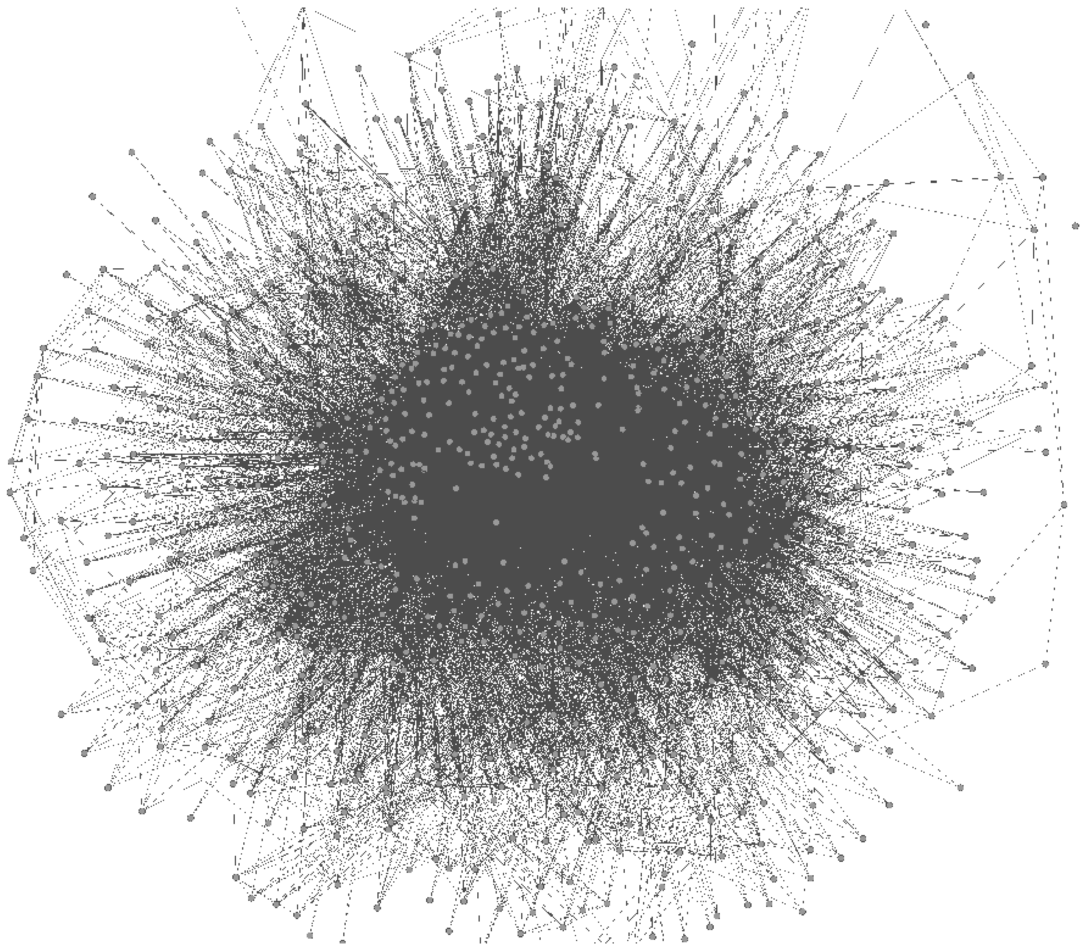


Fig 3.9: Expanded view of the three cliques graph

3.4.4 Set Covering Model

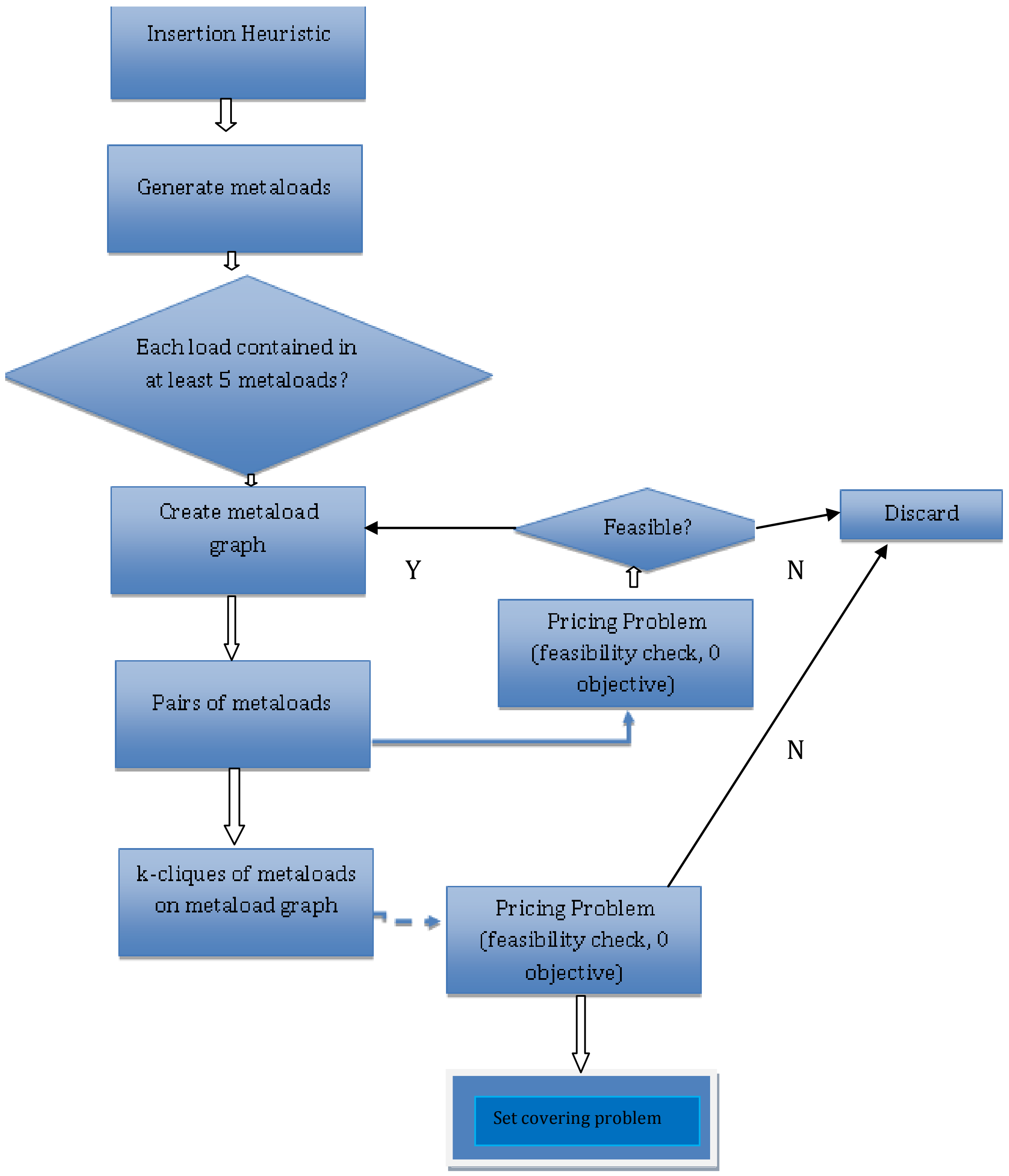
After we obtain the feasibility of 23,869 feasible cliques, we propose to solve the set covering model on these feasible set of cliques. Note that each clique can be a route, as it satisfies the constraints for feasibility, described in Section 3.3.3. We solve the following set covering model with the set *Routes* containing all the feasible cliques.

$$minimize \sum_{j \in Routes} x_j \qquad (3.124)$$

$$subject\ to, \sum_{j \in Routes} a_{ij} x_j \geq 1 \ \forall \ loads\ i \qquad (3.125)$$

Each element a_{ij} takes on value 1 if load i is present in route j and 0 otherwise. Constraint (3.125) specifies that each load should be present in at least one route in the solution. Additionally, because each load is contained in multiple cliques, it can belong to multiple routes, and hence there could be multiple optimal solutions to this problem, all with the same number of total routes, but with different routes and schedules associated with each of them. A summary of our solution approach to this problem is given in the form of a flow chart in Figure 3.10

In order to solve the integer program, we use Java-integrated IBM OPL STUDIO v3.5. Our solutions contain 76 routes, which significantly improves upon the existing solution to this real-world problem. The improvement is of the order of 15% cost reduction from 90 routes to 76 routes. Also, we find 20 multiple optimal solutions, which are schedule different. To differentiate these solutions, we present a framework to evaluate them under uncertainty, as we discuss in the following chapter.



Insertion Heuristic
Generate metaloads
Each load contained in at least 5 metaloads?
Create metaload graph
Feasible?
Y
N
Discard
Pricing Problem (feasibility check, 0 objective)
Pairs of metaloads
N
k-cliques of metaloads on metaload graph
Pricing Problem (feasibility check, 0 objective)
Set covering problem

Fig 3.10 Flowchart for our modeling and solution approach

CHAPTER 4 SIMULATION FRAMEWORK FOR EVALUATION OF ROBUSTNESS

4.1 Simulation Framework

The resultant set of 20 solutions (each containing 76 routes) are compared using a simulator in order to evaluate their properties based on robustness criteria. In particular, we are interested in the performance of the solutions in scenarios of travel time uncertainty and demand uncertainty. The simulator generates builds scenarios in which the uncertain parameters vary according to different distributions. Our simulator framework samples from a specified random distribution to generate scenarios of realized uncertainty. This framework is built in a JAVA based environment integrated with IBM OPL STUDIO. The scenarios are generated using a randomized function in MATLAB for 10%, 20% and 30% variation in traveltime and the corresponding data files were created with the help of MATLAB. The feasibility of all the scenarios is tested by checking each solution against the model in Section 3.3.3 with parameters corresponding to the realized scenario (under stochasticity).

4.2 Simulation Results for Robustness under Travel Time Uncertainty

Scenarios have been generated by varying the travel-time between the locations to $\pm$ 10% , $\pm$ 20% and $\pm$ 30% respectively. Each of these variations was created assuming different distributions - Normal distribution, Poisson distribution and Gamma distribution. The scenarios generated are described in Figures 4.1 and 4.2.

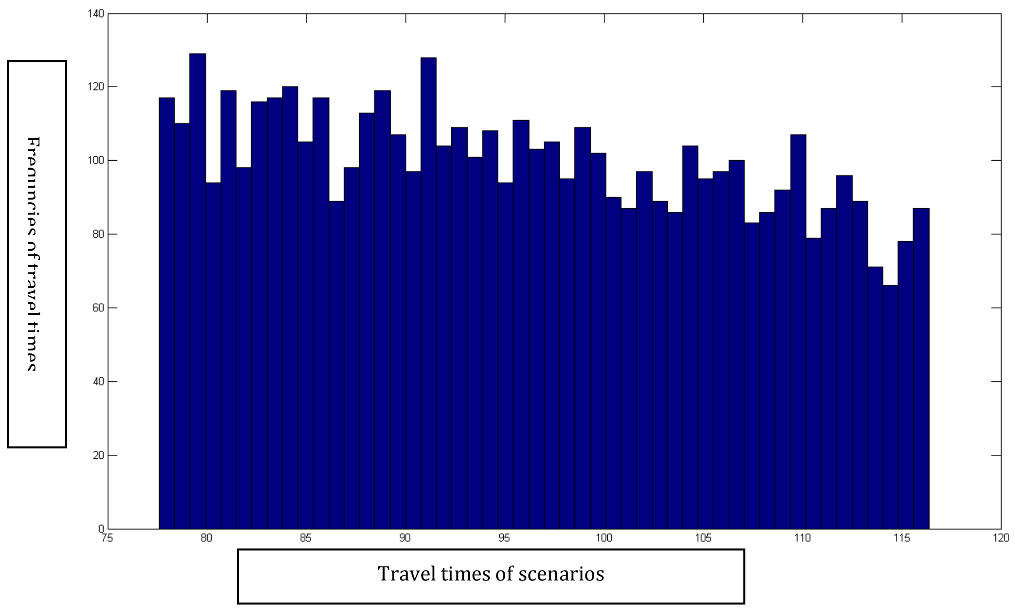


Figure 4.1: Gamma distribution scenarios for travel times

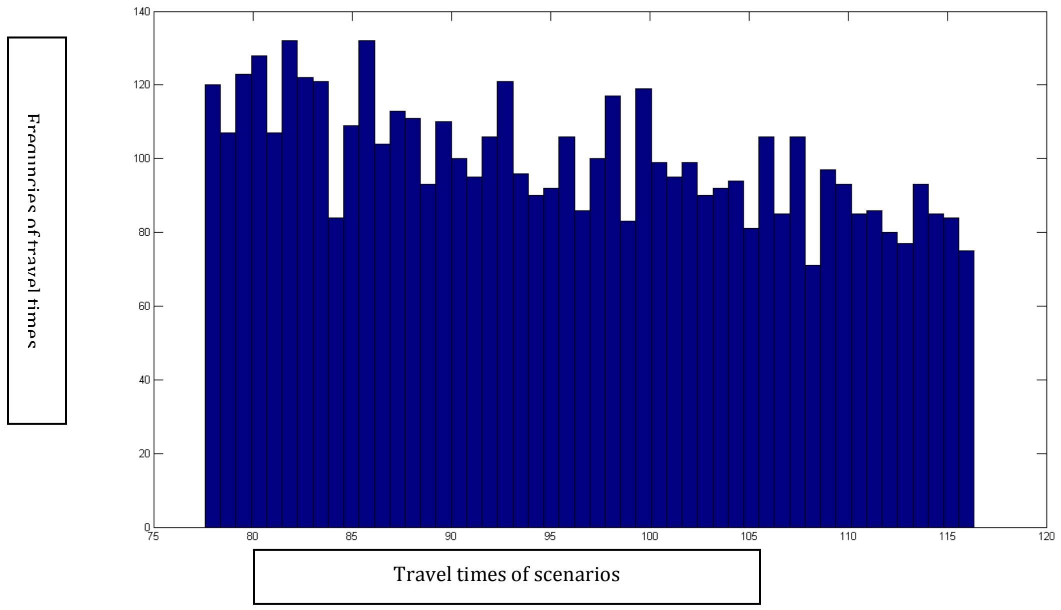


Figure 4.2 : Normal distribution scenarios for travel times

In total, 200 scenarios for each variation and each type of distribution was generated (200 x 3 x 3) and the set of 20 solutions (each containing 76 routes) is tested against each of the scenarios for feasibility. The results are tabulated in Tables 4.1, 4.2 and 4.3.

Solution 1	10% variation	20% variation	30% variation
Normal	98.05%	98.04%	98.04%
Poisson	96.49%	96.37%	96.19%
Gamma	95.95%	95.94%	95.91%

Table 4.1: Feasibility Percentages for all scenarios corresponding to solution1 for three distributions in travel times

Solution 2	10% variation	20% variation	30% variation
Normal	98.05%	98.03%	98.02%
Poisson	96.48%	96.37%	96.18%
Gamma	95.95%	95.91%	95.91%

Table 4.2: Feasibility Percentages for all scenarios corresponding to solution2 for three distributions in travel times

Solution 3	10% variation	20% variation	30% variation
Normal	98.34%	98.34%	98.33%
Poisson	96.33%	96.33%	96.2%
Gamma	96%	95.98%	95.99%

Table 4.3: Feasibility percentages for all scenarios corresponding to solution3 for three distributions in travel times

It is seen that the performance of the different solutions does not vary much with the changing scenarios. Also each solution has a high level of robustness, with a percentage feasibility of above 95% even under 30% uncertainty in travel times. We propose to evaluate these solutions also under scenarios of demand uncertainty.

CHAPTER 5 CONCLUSIONS AND FUTURE SCOPE OF RESEARCH

In this thesis, we have developed a large-scale neighborhood search-based approach to VRPPDTW with side-constraints. The inherent complexity is NP-hard and we attempt at solving the problem using a combination of heuristic approaches. Our modeling and solution approach involves an insertion heuristic followed by generation of load clusters called metaloads. We then model combinations of metaloads by modeling it as a graph problem and using a randomized algorithm to find feasible cliques of metaloads. After generating feasible combinations of metaloads, we solve a set covering problem to obtain feasible routes. Because of the dense connectivity of the graph, we find that there exist multiple optimal solutions to the set-covering problem. We build upon the approach used by Tardy [3] and help improve considerably upon the solutions obtained by the earlier approach.

We then evaluate the obtained solutions using a simulation framework. We also propose to investigate more sophisticated ways of exploring the metaload graph by more efficient algorithms. Additionally, we propose to explore the performance of the different solutions obtained under highly skewed distributions of travel time uncertainty as well as under scenarios of demand uncertainty, to study their robustness properties. Additionally we can investigate if the existence of transfer points in a network can help generate more robust solutions that are less vulnerable to uncertainty. We also hope to link properties of the solutions such as total slack time in schedule, distribution of slack time, and spare capacity on legs of the network to relate these features to the level of robustness of the solution. We envision that this will help in identifying rules of thumb to identify and develop robust and cost-effective solutions to VRPPDTW and other problems arising in transportation and logistics.

REFERENCES

1) Toth, P. , 2002. The vehicle routing problem. Philadelphia: Society for Industrial and Applied Mathematics. pp. 157-193

2) Leeuwen, Jan van, ed. (1998). *Handbook of Theoretical Computer Science*. Vol. A, Algorithms and complexity. Amsterdam
Elsevier . ISBN 0262720140. OCLC 247934368

3) Tardy, Raphael, 2005. *Optimization Models and Algorithms for Large-Scale, Capacity Constrained Pick-up and Delivery Problems with Time Windows* (Masters dissertation). Retrieved from http://dspace.mit.edu/handle/1721.1/7582

4) R. Liu, Xiaolan X., Augusto,V., Rodriguez C., 2013. Heuristic algorithms for a vehicle routing problem with simultaneous delivery and pickup and time windows in home health care**,** European Journal of Operational Research, 230 (3), pp. 475–486

5) Dell'Amico, M., Righini, G., Salani, M., 2006. A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection.
Transportation Science 40, 235–247.

6) Ropke, S., Cordeau, J.F., 2009. Branch and cut and price for the pickup and delivery problem with time windows. Transportation Science 43, 267–286.

7) Li, H., Lim ,A., 2001. A Metaheuristic for the Pickup and Delivery Problem with Time Windows, 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01), pp.07 -09

8) Bent, R., & Hentenryck, P. 2004. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research 40,* 875-893.

9) Ai, T., & Kachitvichyanukul, V. 2009. A particle swarm optimization for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research 36,* 1693-1702.

10)Solomon, M. 1987. Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints. *Operations Research 35,* 254-265.

11) Toth, P., Vigo, D., 2001. The vehicle routing problem. Philadelphia, PA: SIAM Monographs on Discrete Mathematics and Applications, Society for Industrial and Applied Mathematics

12) Baker, E. 1983. An Exact Algorithm for the Time- Constrained Traveling Salesman Problem. Opns. Res. 31, 938-945.

13) Baker, E., S. Rushinek. 1982. Large Scale Implementation of a Time Oriented Vehicle Scheduling Model. U.S. Department of Transportation, Urban Mass Transit Administration

14) Christofides, N., Mingozzi, A., Toth, P. 1979. The Vehicle Routing Problem. In Combinatorial Opti-mizations, N. Christofides, R. Mingozzi, P. Toth, and C. Sandi (eds.). John Wiley & Sons, New York

15) Swersey, A., Ballard, W.. 1982. Scheduling School Buses. Working Paper, Yale School of Organization and Management.

16) Desrosiers, J., Soumis F., Desrochers, M.. 1983a. Routing With Time Windows by Column Generation. Working Paper 277, Centre de Reserche sur les Transports, University of Montreal

17) Mingyong, L., Erbao, C., 2010. An improved differential evolution algorithm for vehicle routing problem with simultaneous pickups and deliveries and time windows. Engineering Applications of Artificial Intelligence 23, 188–195

18) Çatay, B. 2009. Ant Colony Optimization and Its Application to the Vehicle Routing Problem with Pickups and Deliveries. *Natural Intelligence for Scheduling, Planning and Packing Problems Studies in Computational Intelligence 250,* 219-244.

19) Pankratz, G. 2005. A Grouping Genetic Algorithm for the Pickup and Delivery Problem with Time Windows. *OR Spectrum,* 21-41.

20) Gauvin, C., Desaulniers, G., & Gendreau, M. 2014. A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands. *Computers & Operations Research,* 141-153.

21) Desaulniers, G., Lessard, F., Hadjar, A. 2008. Tabu search, partial elementarity and generalized k-path inequalities for the vehicle routing problem with time windows. TranspSci;45:387–404.

22) Duhamel, C., Potvin, J., & Rousseau, J. 1997. A Tabu Search Heuristic for the Vehicle Routing Problem with Backhauls and Time Windows.*Transportation Science 41,* 49-59.

23) Dumas, Y., Desrosiers, J., Soumis, F. 1991, The pickup and delivery problem with time windows. European Journal of Operational Research 54:7–22.

24) Lu, Q., & Dessouky, M. 2006. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research 175,* 672-687.

25) Parragh, S., Doerner, K., & Hartl, R. (2008). A survey on pickup and delivery problems. *Journal Für Betriebswirtschaft,* 81-117.

26) Agra, A., Christiansen, M., Figueiredo, R., Hvattum, L., Poss, M., & Requejo, C. 2013. The robust vehicle routing problem with time windows. *Computers & Operations Research,* 856-866.

27) Agra A, Christiansen M, Figueiredo R, Magnus Hvattum L, Poss M, Requejo C. 2013. Layered formulation for the robust vehicle routing problem with time windows. In: Lecture Notes in Computer Science, vol. 7422/2012; 2012, p. 249–60.

28)Ropke, S., Cordeau, J., & Laporte, G. 2007. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks 49(4) ,* 258-272.

29) Laporte, G., Musmanno, R., & Vocaturo, F. 2010. An Adaptive Large Neighbourhood Search Heuristic for the Capacitated Arc-Routing Problem with Stochastic Demands. *Transportation Science 44(1),* 125-135.

30) Marla, L., Barnhart, C., & Biyani, V. 2013. A decomposition approach for commodity pickup and delivery with time-windows under uncertainty. *J Sched Journal of Scheduling,* 489-506.

31) Cortés, C., Matamala, M., & Contardo, C. 2010. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research,* 711-724.

32) Karger, David. 1993. Global Min-cuts in RNC and Other Ramifications of a Simple Mincut Algorithm. *Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms*.

33) Hochbaum,D.S., .1997. Approximation Algorithms for NP-Hard Problems. PWS Publishing Company, Boston.