

**A Simple Approach to Linehaul-Backhaul Problems:
A Guided Local Search Approach for the Vehicle Routing Problem**

Yingjie Zhong, M.S.O.R.

and

Michael H. Cole, Ph.D.
Assistant Professor

Department of Industrial Engineering
University of Arkansas
4207 Bell Engineering Center
Fayetteville AR 72701 USA

TABLE OF CONTENTS

DEDICATION	Error! Bookmark not defined.
ACKNOWLEDGMENTS.....	Error! Bookmark not defined.
TABLE OF CONTENTS	vi
LIST OF TABLES	viii
LIST OF FIGURES.....	ix
CHAPTER 1 - INTRODUCTION	1
CHAPTER 2 - LITERATURE REVIEW	4
2.1 VRP WITH TIME WINDOWS (VRPTW).....	4
2.2 VRP WITH BACKHAULS (VRPB).....	6
2.3 VRP WITH BACKHAULS AND TIME WINDOWS (VRPBTW)	8
CHAPTER 3 - SOLUTION APPROACH	10
3.1 SOLUTION FEASIBILITY CONSIDERATION	10
3.2 GUIDED LOCAL SEARCH (GLS).....	11
3.2.1 <i>Solution features</i>	11
3.2.2 <i>Utility function</i>	12
3.2.3 <i>Augmented objective function and parameter setting</i>	13
3.3 LOCAL SEARCH HEURISTICS AND NEIGHBORHOODS.....	16
3.3.1 <i>Intra-route exchange: 2-opt</i>	16
3.3.2 <i>Inter-route exchange: 1-move, 1-exchange</i>	17
3.4 DATA STRUCTURES	19
3.4.1 <i>Basic data structure</i>	19
3.4.2 <i>Special data structures</i>	19
3.5 ALGORITHM	24

3.5.1 Phase I.....	24
3.5.2 Phase II.....	26
CHAPTER 4 - EXPERIMENTS.....	29
4.1 VRPBTW WITHOUT CUSTOMER PRECEDENCE	30
4.2 VRPBTW WITH CUSTOMER PRECEDENCE.....	31
CHAPTER 5 - CONCLUSIONS AND RECOMMENDATIONS.....	35
REFERENCES.....	36

LIST OF TABLES

3.1 Pseudocode for setting initial arc time precedences	12
3.2 Pseudocode for phase II	27
3.3 Pseudo code for section planning a route	28
4.1 Results for MC2 data set	30
4.2 Results for MR2 data set	31
4.3 Results for MRC2 data set.....	31
4.4 Results for problem derived from R101	32
4.5 Results for problem derived from R102.....	32
4.6 Results for problem derived from R103	33
4.7 Results for problem derived from R104	33
4.8 Results for problem derived from R105	34

LIST OF FIGURES

3.1 Before and after 1-exchange of nodes x and y	18
3.2 Data structure for capVio[].....	22
3.3 Sample route for capVio[] discussion	22
3.4 Route with customer precedence violation = 2	23
3.5 Route A	24
3.6 Route B.....	24

CHAPTER 1 - INTRODUCTION

This report develops a heuristic algorithm and prototype software for solving the Vehicle Routing Problem with Backhauls and Time Windows (VRPBTW). The problem differs from the well-known VRP in that customers can demand either delivery (linehaul) or pickup (backhaul), and each customer has a service time window. In words, the VRPBTW is as follows.

Given:

- Single depot
- Set of customers with known demands (pickups and/or deliveries)
- Customer service time windows consisting of a ready time (earliest time allowed for truck arrival) and a due time (latest time allowed for truck arrival).
- Homogenous fleet of vehicles (trucks) of fixed capacity

Find:

- Set of vehicle routes that minimize the number of vehicles and the total distance traveled (a surrogate for total cost)

Subject To:

- Each customer's demand (pickup or delivery) must be satisfied
- A customer's demand may not be split between different vehicles
- Customer time windows are "hard" – they may not be violated
- Fleet size (number of vehicles) is unconstrained
- Customer precedence (whether or not linehauls must precede backhauls).

The mathematical model is as follows.

Parameters:

N = number of customer, indexed 1 to N

$0/(N+1)$ = index of the depot

C = fixed vehicle capacity

S = set of routes (a solution)

s = a route that belongs to S

v = vehicle speed

l_{ij} = distance between customer i and customer j , with $i, j = 0, 1, 2, \dots, N+1$

$d_{(i)}(s)$ = demand of the i th customer in route s , delivery as positive and pick up as negative

$l_{(i,i+1)}(s)$ = distance from the i th to $(i+1)$ th customer in route s

$tt_{(i,i+1)}(s) = l_{(i,i+1)}(s)/v$, travel time from the i th customer to the $(j+1)$ th customer in route s

$rt_{(i)}(s)$ = ready time of the i th customer in route s

$dt_{(i)}(s)$ = due time of the i th customer in route s

$sv_{(i)}(s)$ = service time of the i th customer in route s

$n(s)$ = number of customers, including depot (starting and ending), in route s .

Variables:

$I_{ij}(S) = 1$ if arc ij is in solution S , 0 otherwise

$at_{(i)}(s)$ = vehicle arrival time at the i th customer in route s

$lt_{(i)}(s)$ = vehicle leave time from the i th customer in route s

$vl_{(i)}(s)$ = vehicle load at the i th customer in route s

$cl(s)$ = total linehaul demand for route s

$pv(s)$ = total number of backhaul customer clusters that precede linehaul customer in route s

Formulation:

Minimize:

$$g(S) = \sum_{i \neq j} I_{ij}(S) \cdot l_{ij} \quad i, j = 0, 1, \dots, N+1 \quad (1.1)$$

Subject to:

$$lt_{(0)}(s) = 0 \quad s \in S \quad (1.2)$$

$$at_{(i)}(s) = lt_{(i-1)}(s) + t_{(i-1,i)}(s) \quad i = 1, 2, \dots, n(s) \quad s \in S \quad (1.3)$$

$$at_{(i)}(s) - dt_{(i)}(s) \leq 0 \quad i = 1, 2, \dots, n(s) \quad s \in S \quad (1.4)$$

$$lt_{(i)}(s) = \max(rt_{(i)}(s), at_{(i)}(s)) + sv_{(i)}(s) \quad i = 1, 2, \dots, n(s) \quad s \in S \quad (1.5)$$

$$vl_{(0)}(s) = cl(s) \quad s \in S \quad (1.6)$$

$$vl_{(i)}(s) = vl_{(i-1)}(s) - d_{(i)}(s) \quad i = 1, 2, \dots, n(s) \quad s \in S \quad (1.7)$$

$$vl_{(i)}(s) - C \leq 0 \quad i = 0, 1, \dots, n(s) \quad s \in S \quad (1.8)$$

$$pv(s) = 0 \quad s \in S \quad (1.9)$$

Constraints (1.2) to (1.5) state that a customer cannot be serviced after its due time. Constraints (1.6) to (1.8) indicate that the vehicle load can not exceed the fixed capacity at any customer/depot. Constraint (1.9) says that the backhaul customer can not precede any linehaul customer in each route (this constraint is only used for the VRPBTW with customer precedence).

The traditional VRP is an NP-hard problem. Adding time windows and backhauls makes does not simplify the problem. Thus, this research develops a heuristic approach to the VRPBTW. The new heuristic is a cluster-first, route-second algorithm, with most effort spent on the routing phase. The two phases are as follows.

Phase 1 – Use an adapted sweep algorithm (Fisher and Jaikumar, 1981) to generate an initial infeasible solution. Use a guided local search heuristic (based partly on Voudouris and Tsang, 1999) to guide local search heuristics such as 2-opt, 1-move, and 1-exchange (a new method) to improve the initial solution.

Phase 2 – Enhance feasibility using a new technique called “section planning.” The technique inserts new routes until feasibility is attained, and arranges customers within routes to reduce travel distance. This phase is iterative with feasibility constraints being “soft” in early iterations and “hard” in later iterations.

This research makes two main contributions to the state-of-the-art in solving the VRPBTW problem:

- 1-exchange: a new local search procedure based on a particular neighborhood structure.
- Section Planning: a new method to enhance solution feasibility.

CHAPTER 2 - LITERATURE REVIEW

This section reviews literature (mostly post-1990) representing the current state of the art in solving the VRPTW (vehicle routing problem with time windows), VRPB (vehicle routing problem with time windows), and VRPBTW (vehicle routing problem with time windows and backhauls). For the general VRP, good sources of information include Bodin et al (1983), Golden and Assad (1988), Aarts and Lenstra (1997), and Reeves (editor) (1993).

2.1 VRP with time windows (VRPTW)

Potvin et al (1996) develops a tabu search heuristic for the VRPTW. The tabu search is based on particular local search heuristics that always maintain the feasibility of the solution. Two local search algorithms are used:

1. 2-opt* exchange (based on 2-opt), is used as an inter-route exchange technique, in order to ease maintenance of solution feasibility
2. Or-opt is used for intra-route as well as inter-route exchange.

The local search exploration space for 2-opt* and Or-opt is limited so that exchanges can only be made between customers within a certain distance of each other. A fixed tabu list is designed to prohibit returning to a recent solution. In their application, a tabu list of length five provided the best result. Based on a standard set of 100-customer Euclidean VRPTW problems, Potvin's heuristic provides better results than such algorithms as Solomon's L1 heuristic, the parallel insertion heuristic PARIS, the greedy randomized adaptive search procedure GRASP, the cyclic transfer algorithm CTA, and the genetic sectoring algorithm GIDEON.

Potvin and Bengio (1996) introduce a genetic routing system (GENEROUS) based on the natural evolution paradigm. They avoid the issue of genetic representation/encoding and the design of crossover operators. They design two genetic operators to generate the offspring from the parent solutions. Instead of generating the offspring randomly, they use a heuristic to provide guidance. Sequence-Based Crossover is similar to 2-opt*; Route-Based Crossover is similar to Or-opt. Offspring are subject to two types of mutation. One-Level-Exchange moves one customer from one route. Two-Level-Exchange acquires one

customer from one route and move one customer to another route. GENEROUS is not efficient in computing time, but does generate high quality solutions compared to other VRPTW heuristics.

Russel (1995) and Chang and Russel (1997) recognize that the quality of the initial solution is very important for algorithms that seek to maintain feasibility. Russell (1995) designs a parallel construction heuristic that embeds a route improvement procedure within the route construction stage. The heuristic is adapted from Solomon's sequential insertion heuristic. To construct the initial solution, the new heuristic has three ordering rules: 1) the earliest time window; 2) the width of the time window augmented by distance from the depot; 3) the farthest distance from the depot. Instead of wasting time searching a large solution space, the heuristic searches only within restricted neighborhoods.

Chiang and Russell (1997) develop a reactive tabu search for the VRPTW. As in Russell (1995), the heuristic embeds route improvement into the route construction stage. Based on an initial feasible solution, the construction procedure is carried out to generate another solution with fewer routes. This process iterates until no more feasible solutions can be generated.

The route improvement procedure, which runs in parallel with the route construction procedure, comprises three techniques:

- (1, 0) - move a customer to another route
- (0, 1) - move a customer from another route
- (1, 1) - swap two customers between two routes.

The neighborhood structure of the procedure follows the λ -interchange mechanism of Osman (1993). The λ -interchange mechanism is an ordered search method that examines all possible combinations of pairs of routes for exchange. The authors also use Solomon's *push forward* and *push back* mechanisms to organize the time window evaluation procedure. Results show that the reactive tabu search is successful both for standard test problems as well as real-world problems. The success of the reactive tabu search heuristic is attributed not only to the search strategies but also to the neighborhood mechanism and the embedding of the tabu search improvement process during route construction.

Kontoravdis and Bard (1995) design a greedy randomized adaptive search procedure (GRASP). It seems that most efforts are carry out during the construction of the initial routes. The algorithm first searches for a number of seeds used to generate the routes. Then, the algorithm follows two phases to

construct and improve the solution. In the first phase, a feasible solution is constructed and a local optimum is reached by local search. In the second phase, each route is considered for elimination, with routes having fewer customers examined first. To improve the quality of the solution in terms of the features of a particular problem, three different lower bounding heuristics were developed. The first considers the “bin packing” aspect of the problem with regard to vehicle capacity; the second is based on the geographically distribution of the customers; and the third independently exploits the time window constraints. In practice, the simplest method, which consider only vehicle capacity constraints, outperforms the more sophisticated methods. The authors also do experiment on the VRPBTW, but they do not indicate which part of the algorithm is adjusted to include consideration of backhauls.

2.2 VRP with backhauls (VRPB)

Deif and Bodin (1984) develop a heuristic to solve the VRPB using a modified Clarke-Wright algorithm. They assume that the backhaul customers can not be visited until all the linehaul customers are served in a route. Thus the whole route can be divided into two separate parts, the linehaul set and the backhaul set. In the algorithm, linehaul routes and backhaul routes are organized separately. A penalty multiplier serves to delay insertion of the backhauls until the end of a route. One limit of this heuristic is that once a backhaul is inserted, the direction of that route is fixed. The result show that prevention of early route termination problem leads to better results, and that linehaul and backhaul customers should not be freely combined.

Golden et al. (1985) also tried the Clarke-Wright method and the cheapest insertion heuristic on a different VRPB. They assumed 1) deliveries after pickups are allowed, 2) many more deliveries than pickups, 3) about 10% of all points are backhauls. Two constraints are included, the vehicle capacity and the route length. The experiment result shows that as the penalty terms are increased, the number of deliveries after backhauls decreases to zero while the total route length increases. Generally the result will be better when the backhauls allow to be inserted as soon as vehicle capacity allowed.

Casco, Golden, and Wasil (1988) note that to solve the VRPB, several important issues need to be considered

- What percentage of stops are backhauls?

- Must backhauls always come after all deliveries are made?
- What is the size of a typical backhaul relative to vehicle capacity?
- Is the number of trucks fixed in advance?
- Can backhauls be split?
- Will a vehicle ever serve backhauls only?

They propose three rules to increase the convenience of vehicle operations in a VRPBTW environment:

- A vehicle servicing a particular set of customers should deliver sixty percent of its units before accepting any backhauls.
- If a delivery follows a backhaul on a proposed route, then accept the backhaul as long as no more than eighty percent of the vehicle is full after doing so.
- If no deliveries remain, then all backhauls are eligible to the extent permitted by vehicle capacity.

To incorporate the proposed standards, they suggest generating initial solutions using the Clarke-Wright approach, but with route capacities smaller than the actual vehicle capacities in order to increase flexibility in assigning backhauls to routes. The decision to insert a backhaul considers the delivery load that remains on the vehicle after the backhaul is served. In their heuristic, the direction of the route can be changed based on the load on the vehicle.

Goetschalckx and Jacobs (1989) propose a two-phased solution methodology to solve the VRPB with customer precedence, in which backhauls cannot be visited before linehauls. In the first phase, a high quality initial feasible solution is generated based on space-filling curves. The clustering of linehaul customers and the backhaul customers are separate and then lined up together. In the second phase, the solution is improved based on optimization of the sub-problems by using the local optimization heuristics such as greedy algorithms, k -median algorithms combined with 2-opt and 3-opt. Results show that the solution time of the space-filling curve heuristic is relatively insensitive to problem size, compared to the Clarke-Wright heuristic. However, the quality of the solution is hard to compare to those of other heuristics.

Jacobs-Blecha and Goetschalckx (1993) develop another heuristic (LHBH) to solve the VRPB. Their model decomposes the VRPB into three sub-problems. The first two sub-problems correspond to the clustering decisions for the delivery customers and the pickup customers, which are independent

Generalized Assignment Problems (GAP). A savings regret heuristic is adapted in these two phases. The third sub-problem consists of K independent, single route TSPs with backhauls, and is solved by using cheapest insertion heuristics. The result shows that the solution quality of LHBH is better than the Clarke-Wright heuristic, although LHBH has longer solution times for the problems considered. The execution time of the Clarke-Wright heuristic increases much faster with problem size than does that of LHBH.

Toth and Vigo (1996) proposed a cluster-first-route-second heuristic algorithm for VRPB called TV. It starts from an initial, possibly infeasible, solution obtained by solving a relaxation of VRPB, and tries to build a final set of feasible routes through vertex movements and arc exchanges. In the first step, the Lagrangian lower bound is obtained by including into the objective function the degree constraints for each vertex, and a subset of the so-called capacity-cut constraints, imposing both the connectivity of the solution and the capacity requirement. The result is the collection of linehaul and backhaul vertices respectively. Then use a matching heuristic (TSPB) to match the linehaul and backhaul clusters. The second step includes both intra-exchange and inter-exchange by using 2-opt and 3-opt heuristic under the feasibility condition. The TV heuristic gets better solutions than SF and LHBH heuristics. Toth and Vigo (1999) extend the heuristic to solve asymmetric VRPB problems.

2.3 VRP with backhauls and time windows (VRPBTW)

Gelinas et al. (1995) propose a new branching strategy for branch-and-bound approaches based on column generation for the VRPBTW. This algorithm finds optimal solutions for test problems with up to 100 customers.

Kontoravdis and Bard (1995) design a greedy randomized adaptive search procedure (GRASP) for VRPBTW. This algorithm is also capable of solving VRPBTW without customer precedence. Experiments on problems with clustering distributed customer were reported.

Potvin, Duhamel, and Guertin (1996) design a genetic algorithm to identify an ordering that produces good routes. Duhamel, Potvin, and Rousseau (1997) design a tabu search heuristic for the VRPBTW. They also assume that the backhaul customers in a route can not be served until all the linehaul customers have been served.. Their heuristic has two steps. In the first step, a feasible solution is constructed by the adapted Solomon (1987) insertion heuristic called I1, which is basically a sequential

construction procedure. During the second step, tabu search together with local search and improvement algorithms are used to improve the solution, maintaining the feasibility all the time. The local search heuristics are 2-opt* exchange, Or-opt exchange, and swap.

CHAPTER 3 - SOLUTION APPROACH

This chapter details a new solution approach for the VRPBTW. Section 3.1 discusses the advantages of delaying the development of feasible solutions until the latter part of the algorithm. Section 3.2 relates the concept of Guided Local Search to the VRPBTW. Section 3.3 describes local search heuristic methods, including the newly developed 1-exchange method. Section 3.4 describes relevant data structures that simplify the programming of the algorithm. Section 3.5 details the two phases of the algorithm, and explains Section Planning, a new feasibility enhancement method.

3.1 Solution feasibility consideration

Whether to maintain feasibility from the very beginning of an algorithm or let the solution be infeasible for awhile is difficult to decide. Both approaches have advantages and disadvantage. To tolerate the infeasible solutions during the early steps of a routing algorithm allows great flexibility to exchange the customers between routes. Eventually, however, a feasible solution must be developed; this may be difficult. On the other hand, maintaining solution feasibility at all times might constrain the solution to a local optimum that is not very good overall.

The problem is that we need a flexible neighborhood for the local search heuristics, but we also want to guarantee that the algorithm gets a feasible solution with the least or close to the least number of routes. Two questions arise in developing the algorithm and attendant data structures:

- 1) Is the information of the solution before it reaches a feasible state helpful? This depends on the procedure that makes it feasible. One good example is annealing. The good quality of metal is a consequence of the whole annealing procedure before it. Temperature control is extremely important to the quality of the metal. Similarly, the penalty parameter control is also crucial to the algorithm.
- 2) Will the feasibility enhancement change much of the route structure? This also depends on the techniques used.

The algorithm developed here allows infeasibility, but pays special attention to feasibility enhancement.

During recent years, other researchers in VRP with additional constraints such as time windows or backhauls have considered postponing solution feasibility (e.g., Chiang and Russell, 1997). The method developed in this report considers solution infeasibility differently from Chiang and Russel. In their

heuristic, infeasibility means that there exist unrouted customers, but the routes that exist are feasible. In the heuristic developed in this report, the routes themselves may be infeasible.

3.2 Guided local search (GLS)

Guided local search (GLS) is a metaheuristic designed to guide local search heuristics.

Voudouris and Tsang (1999) developed guided local search and applied it to the traveling salesman problem. GLS penalizes solution features during each iteration, based on the associated costs (e.g., length, time precedence, and arc penalty). The augmented objective value, which guides the evaluation of the local search, is adjusted during each iteration of the algorithm. Thus the chance that the objective value gets stuck in a local optimum is significantly decreased.

3.2.1 Solution features

Voudouris and Tsang (1999) note that “A solution feature can be any solution property that satisfies the simple constraint that is a non-trivial one”. The cost of feature represents the direct or indirect impact of the corresponding solution properties on the objective function. A solution feature is represented by an indicator function in the following way:

$$I_{ij}(s) = \begin{cases} 1 & \text{solution } s \text{ has property } i \\ 0 & \text{otherwise} \end{cases} \quad s \in S \quad (3.1)$$

A solution feature can have more than one feature cost pertaining to it. In this report, the arcs connecting pairs of the customers are solution features; the arc length and arc penalty are the feature costs. The arc penalty is updated during each iteration of the algorithm. Once the penalty for an arc exceeds a pre-decided limit (in this report the limit is set as 60), its solution feature is set to 0 for the rest of the algorithm. The initial value of arc penalty is set equal to the arc time-precedence.

$$\text{arcLength} = [l_{ij}] \quad (3.2)$$

$$\text{arcTime} = [t_{ij}] \quad (3.3)$$

$$\text{arcPenalty} = [p_{ij}] \quad (3.4)$$

Table 3.1. Pseudocode for setting initial arc time precedences

<p>Set i, j any pair of customers or depot</p> <p>Set $t = l_{ij}/\text{vehicle speed}$</p> <p>If ($t + \text{customer } i\text{'s ready time} > \text{customer } j\text{'s due time}$)</p> <p style="padding-left: 40px;">$t_{ij} = 200$</p> <p>Else if ($t + \text{customer } i\text{'s ready time} > \text{customer } j\text{'s ready time}$)</p> <p style="padding-left: 40px;">$t_{ij} = 0$</p> <p>Else</p> <p style="padding-left: 40px;">$t_{ij} = 20;$</p> <p>End if</p> <p>If ($t + \text{customer } i\text{'s due time} < \text{customer } j\text{'s due time}$)</p> <p style="padding-left: 40px;">$t_{ij} += 20$</p> <p>Else if ($t + \text{customer } i\text{'s due time} > \text{customer } j\text{'s due time}$)</p> <p style="padding-left: 40px;">$t_{ij} += 20$</p> <p>End if</p>
--

3.2.2 Utility function

A utility function is used to guide the incrementing of the penalty index during the local search. The costs of a solution feature and the previous/initial arc penalty are used to update the arc penalty. Based on the value of the utility function of each arc, a certain percentage of arcs are penalized during each iteration. The utility function is expressed in equation (3.5) and (3.6).

$$util(s, f_{ij}) = I_{ij}(s) \cdot f(l_{ij}, t_{ij}, p_{ij}) \quad s \in \mathcal{S} \quad (3.5)$$

$$f(l_{ij}, t_{ij}, p_{ij}) = \frac{l_{ij} \cdot t_{ij}}{1 + p_{ij}} \quad (3.6)$$

3.2.3 Augmented objective function and parameter setting

By using Lagrangean-relaxation method to solve the VRPBTW, the inequality constraints can be included to the original objective function (Taha, 1997) to create a new objective function as shown in equation 3.7.

$$L(S) = g(S) + \bar{\lambda}_t \cdot \bar{\mathbf{v}}_t + \bar{\lambda}_c \cdot \bar{\mathbf{v}}_c \quad (3.7)$$

$$\bar{\lambda}_t = \bar{\lambda}_t(1) \cup \bar{\lambda}_t(2) \cup \dots \cup \bar{\lambda}_t(n(S)) \quad (3.8)$$

$$\bar{\lambda}_t(s) = (\lambda_{t_1}(s), \lambda_{t_2}(s), \dots, \lambda_{t_{n(s)}}(s))^T \quad (3.9)$$

$$\bar{\mathbf{v}}_t = \bar{\mathbf{v}}_t(1) \cup \bar{\mathbf{v}}_t(2) \cup \dots \cup \bar{\mathbf{v}}_t(n(S)) \quad (3.10)$$

$$\bar{\mathbf{v}}_t(s) = (at_{(1)}(s) - dt_{(1)}(s) + st_{s_1}^2, \dots, at_{(n(s))}(s) - dt_{(n(s))}(s) + st_{s_{n(s)}}^2)^T \quad (3.11)$$

$$\bar{\lambda}_c = \bar{\lambda}_c(1) \cup \bar{\lambda}_c(2) \cup \dots \cup \bar{\lambda}_c(n(S)) \quad (3.12)$$

$$\bar{\lambda}_c(s) = (\lambda_{c_1}(s), \lambda_{c_2}(s), \dots, \lambda_{c_{n(s)}}(s))^T \quad (3.13)$$

$$\bar{\mathbf{v}}_c = \bar{\mathbf{v}}_c(1) \cup \bar{\mathbf{v}}_c(2) \cup \dots \cup \bar{\mathbf{v}}_c(n(S)) \quad (3.14)$$

$$\bar{\mathbf{v}}_c(s) = (vl_{(1)}(s) - C + sc_{s_1}^2, \dots, vl_{(n(s))}(s) - C + sc_{s_{n(s)}}^2)^T \quad (3.15)$$

$n(s)$: total number of customers in route s

$n(S)$: total number of routes in solution S

st_i^2, sc_i^2 : slack variables

$\bar{\lambda}_t$ and $\bar{\lambda}_c$ are Lagrangean multipliers, acting as dual variables with their values corresponding

to the price of the related resources. The slack values $st_{s_i}^2$ and $sc_{s_i}^2$ need to be larger than or equal to 0.

In the heuristic, infeasible solutions are allowed, that is,

$$at_{(i)}(s) - dt_{(i)}(s) \geq 0 \quad \text{for some } i, s \quad (3.16)$$

$$vl_{(i)}(s) - C \geq 0 \quad \text{for some } i, s \quad (3.17)$$

For the violated constraints, the slack variables are set as follows:

$$at_{(i)}(s) - dt_{(i)}(s) - st'_{s_i} = 0 \quad (3.18)$$

$$vl_{(i)}(s) - C - sc'_{s_i} = 0 \quad (3.19)$$

To obtain the feasible solution, st'_{s_i} and sc'_{s_i} need to be zero. Our idea is derived from the Lagrangean method, with the Lagrangean multipliers penalizing the violated constraints' slack variables. A disturbance item derived from the GLS is also included. Equation (3.20) is the augmented objective function for the VRPBTW without customer precedence.

$$h(S) = g(S) + \lambda \cdot \sum_{ij \in S} p_{ij} \cdot I_{ij}(S) + \lambda_t \cdot v_t + \lambda_c \cdot v_c \quad (3.20)$$

$$v_t = \sum_{s \in S} \sum_{i=1}^{n(s)} \max(at_{(i)}(s) - dt_{(i)}(s), 0) \quad (3.21)$$

$$v_c = \sum_{s \in S} \sum_{i=0}^{n(s)} \max(vl_{(i)}(s) - C, 0) \quad (3.22)$$

v_t : Violation of due time constraints

v_c : Violation of capacity constraints

λ , λ_t and λ_c : Parameters associated manipulating the magnitude of the corresponding term

The p_{ij} terms act as disturbances to the original objective function and prevent getting stuck in a local optimum. The influence of p_{ij} in the augmented objective function should be eventually be zero. Note that the exact v_t and v_c for each solution are not calculated, instead the differences of v_t and v_c are calculated when evaluating local search moves. This reduces computation time.

Parameter λ decreases after each iteration, while parameter λ_t and λ_c increase after each iteration. Equations (3.23) to (3.28) give the method to update these parameters.

$$\lambda_{t+1} = \lambda_t / \theta \quad (3.23)$$

$$\theta = \exp(\ln(\lambda_{t_{initial}}) - \ln(\lambda_{t_{final}}) / n) \quad (3.24)$$

$$\lambda_{c_{i+1}} = \lambda_{c_i} \cdot \theta_c \quad (3.25)$$

$$\theta_c = \exp(\ln(\lambda_{c_{final}}) - \ln(\lambda_{c_{initial}}) / n) \quad (3.26)$$

$$\lambda_{t_{i+1}} = \lambda_{t_i} \cdot \theta_t \quad (3.27)$$

$$\theta_t = \exp(\ln(\lambda_{t_{final}}) - \ln(\lambda_{t_{initial}}) / n) \quad (3.28)$$

n : Total number of iterations,

$i = \{1, 2, \dots, n - 1\}$: Iteration,

$\lambda_{initial}, \lambda_{t_{initial}}, \lambda_{c_{initial}}$: Initial value of $\lambda, \lambda_t, \lambda_c$.

$\lambda_{final}, \lambda_{t_{final}}, \lambda_{c_{final}}$: The value of $\lambda, \lambda_t, \lambda_c$ in the last iteration.

Equation (3.29) is the augmented objective function for the VRPBTW with customer precedence (the backhaul customers can only be allocated after all the linehaul customers in each route). A multiplier, λ_p , is added to evaluate the precedence of customers in each route.

$$h(S) = g(S) + \lambda \cdot \sum_{ij \in S} p_{ij} \cdot I_{ij}(s) + \lambda_t \cdot v_t + \lambda_c \cdot v_c + \lambda_p \cdot v_p \quad (3.29)$$

$$v_p = \sum_{s \in S} p v(s) \quad (3.30)$$

λ_p increases after each iteration as expressed in equation (3.31) and (3.32).

$$\lambda_{p_{i+1}} = \lambda_{p_i} \cdot \theta_p \quad (3.31)$$

$$\theta_p = \exp(\ln(\lambda_{p_{final}}) - \ln(\lambda_{p_{initial}}) / n) \quad (3.32)$$

v_p : Index of customer precedence violation

λ_p : Parameters associated manipulating the magnitude of the corresponding term

$\lambda_{p_{initial}}$: Initial value of λ_p .

$\lambda_{p_{final}}$: The value of λ_p in the last iteration.

3.3 Local search heuristics and neighborhoods

Local search heuristics are designed to find high quality solutions within a shrunk solution space (a neighborhood). Simple local search heuristics followed by more sophisticated local search heuristics often can yield good solution quality within reasonable running time. In solving a complicated problem, a set of local search heuristics based on different neighborhoods is necessary to increase the search efficiency and effectiveness. The selection of a neighborhood for each local search heuristic is quite flexible. However, if the neighborhood is too large, the running time may be extremely long; if the neighborhood is too small, the solution quality may be not good. The approach used in the algorithm developed here is to define a large neighborhood and to discard obviously bad portions before conducting a detailed search.

This section introduces three local search heuristics that can be categorized into two classes, intra-route exchange and inter-route exchange. 2-opt and 1-move are well-known arc exchange heuristics that are frequently used in the VRP literature, while 1-exchange is a new approach designed particularly for the VRPBTW. The following subsections detail each local search heuristic and the associated neighborhoods.

3.3.1 Intra-route exchange: 2-opt

2-opt (Croes, 1958) is a well-known local search heuristic used to solve travelling salesman problem (TSP) and other combinatorial problems. 2-opt is normally used in the VRP without time windows. Potvin et al (1996) developed 2-opt*, a modification of 2-opt, that maintains solution feasibility when solving the VRPTW. The algorithm developed in this report uses the original 2-opt heuristic since maintaining solution feasibility is not an issue.

In the case of VRPBTW without customer precedence, 2-opt considers two items when considering swapping two arcs: the length and the time precedence of the arcs and their corresponding parameters. For the VRPBTW with customer precedence, 2-opt must also consider customer precedence. For both cases of the VRPBTW, the neighborhood of 2-opt for each route is all pairs of arcs within that route.

3.3.2 Inter-route exchange: 1-move, 1-exchange

1-move:

The 1-move heuristic considers deleting one node from a route and inserting it into another route. The evaluation of 1-move is decided by the length and time precedence of the arc, and the time and capacity violation change caused by 1-move. 1-move is the same heuristic as operator (1,0) and (0,1) proposed by Chiang and Russel (1997). It is effective in improving solution quality and feasibility when the solution quality is relatively low.

The neighborhood of 1-move comprises all the customers and each customer's potential positions in any pair of routes in the current solution.

$$oneMove(R_s, R_t) \quad s, t \in S, s \neq t \quad (3.33)$$

Note that a customer cannot necessarily be moved to just any position in the current set of routes. The set of potential positions is a more limited neighborhood defined by the penalty of the solution features (arcs) associated with swapping the customer. Since arc penalties are updated dynamically, when they exceed the pre-decided limit, they preclude certain customer positions from being allowable.

1-exchange:

The 2-opt and 1-move local search heuristics are not sufficient for the VRPBTW. The difficulty of the VRPBTW is that it has more local optimal obstacles than the VRPB or VRPTW. 1-exchange is related to the 1-move heuristic, but does not simply swap two customers between two routes. It also restructures each route to account for the related customer deletion and insertion. Figure 3.1 shows two routes before and after a pair of nodes undergoes 1-exchange. In the beginning, customer x is in route 1 connected by arcs (x_1, x) and (x, x_2) , and customer y is in route 2 connected by arcs (y_1, y) and (y, y_2) . The 1-exchange procedure first deletes the two customers from the routes. Then, customer x searches for a candidate position in route 2, and simultaneously customer y searches for a candidate position in route 1. The best combination of these two insertions (if the evaluation result is better than the original customer

locations) is the result of 1-exchange. Customer x finds its best insertion location in route 2 is after customer y_2 ; the best insertion location for customer y in route 1 is before customer x_1 . Note that this differs from simply swapping customer x and y without revising the routes to account for the deletions and insertions.

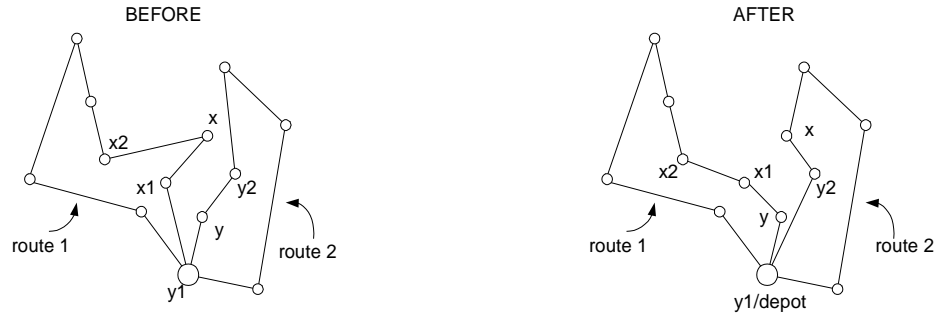


Figure 3.1 Before and after 1-exchange of nodes x and y

The 1-exchange heuristic is designed to intensify the local search. The neighborhood of 1-exchange

$$oneExchange(R_s, R_t) \quad s, t \in S, s \neq t \quad (3.34)$$

is larger than that of 1-move since both of the customers need to find their best position in the other route.

To reduce the computational burden:

1. The GLS arc penalty updating will automatically shrink the local search space. If the penalty of an arc exceeds a pre-decided limit, discard this arc from the search space
2. If the length of the new arcs is more than two times that of the deleted arcs, cancel this candidate exchange and avoid extensive evaluation of the time windows and capacity violations
3. Perform 1-exchange once in every two or three iterations of the algorithm, rather than every single iteration.

3.4 Data structures

3.4.1 Basic data structure

Matrixes: arcLength, timePrecedence and arcPenalty

The data that will be most frequently used in the algorithm is the lengths and time precedence of the arcs. These data are stored in two static matrixes, *arcLength* and *timePrecedence*. The arc length l_{ij} is the Euclidean distance between customer (or depot) i and customer j . The time precedence t_{ij} is the index of time feasibility from customer i to customer j . The time precedence is set to 0 if it is most likely that the arc will not lead to infeasible route; the time precedence is set very high if the arc is definitely infeasible

ArcPenalty is dynamic matrix since some of arcs' penalties should be updated after each big iteration. The initial value of the *arcPenalty* is set as the value of the *timePrecedence*.

List: routes

Such customer information as customer ID number, time windows, the location, the polar angle, and the type of service is used during route construction and route improvement. Dynamic information, such as the vehicle load, the reach time, and the leave time at the customers is also used. The following procedure organizes the information:

1. Create a storage class named *Customers* to organize that private information related to individual customer;
2. Define an array of the *customers* to store the customer-related information (include the depot);
3. Create an object of *CPtrList* class named *route* to connect customers (include the depot) in the same route by pointing the pointer to the array of customers;

All the procedures related to route construction and route improvement, such as the evaluation of objective function and arc exchange, are based on the *route* object.

3.4.2 Special data structures

The evaluation of time window violations, capacity violations, and customer precedence violations (for VRPBTW with customer precedence) is of primary importance since this procedure is used every time a local exchange is evaluated, and occupies a large fraction of the total run time. Special data structures are

necessary to minimize the calculation time and significantly reduce total run time. In the remainder of this section, data structure for evaluation of time window violation, the capacity violation and the customer precedence violation are discussed.

Time-window-related

Two arrays, $timeVio[]$ and $timeVioE[]$, are used to store the dynamic information of the due time violations and the ready time violation respectively.

$$timeVio[] = \{tVio_i \mid i \in S\} \quad (3.35)$$

$$timeVioE[] = \{tVioE_i \mid i \in S\} \quad (3.36)$$

The elements in these two arrays are structures which comprise:

- 1) the order of the customer in the route,
- 2) the volume of the violation,
- 3) the pointer to the customer.

The information in these two arrays needs to be updated after each successful local move. There are two advantages of this particular data structure. First, it eliminates the necessity of checking the feasibility of time windows under certain circumstances. Suppose when doing 1-move, customer x is deleted from route I . Customer x is in the δ^{th} position in the route, and the σ^{th} customer is the last customer violating the due time violation. Under this condition, no evaluation is necessary. Second, it can give necessary information for use in feasibility enhancement. For instance, some customers are obviously infeasible for some routes. They can be directly deleted from the routes. In section planning (introduced later), this data structure is particularly useful and very efficient.

The data structure incurs no additional run time, since all the information is already calculated during the calculation of the objective value.

Capacity-related

In the VRPBTW with customer precedence, all the linehaul customers should be arranged in front of the backhaul customers. Two arrays, $lineDmd[]$ and $backDmd[]$, store the total linehaul demand and the total backhaul demand respectively.

$$lineDmd[] = \{lD_i \mid i \in S\} \quad (3.37)$$

$$backDmd[] = \{bD_i \mid i \in S\} \quad (3.38)$$

The largest vehicle load is equal to $\max(ld, bd)$. Although vehicle capacity will be violated before reaching the final solution, this violation is negligible during the algorithm because 1) the customer precedence will converge to feasibility as corresponding Lagrangean multiplier is increased; and 2) the capacity violation will also converge to feasibility as long as the customer precedence is feasible.

An array, *capVio*[], is defined to represent the customers that violate the capacity constraints during each iteration. Each element in the array is a structure that has three variables:

- 1) the order in the route,
- 2) the volume of violation,
- 3) the pointer to the customer.

There are two advantages of this data structure. First when evaluating the local search, we need not search the customers in the route one after another, but simply allocate the corresponding active capacity-violating-customer and search for only a few customers. For instance, in figure 3.5, suppose we need to move customer 8 out of the route. Knowing that there's three elements in *capVio*[] array (see figure 3.2), and customer 8 is a backhaul customer, we can easily find that there is one capacity-violating-customer (depot) active, whose order in the route is 10, and the violation volume is 60. We also find that customer 8's demand exceeds 60, thus the decreased capacity violation for this route is 60. The operation is simple if we want to add a customer into the route.

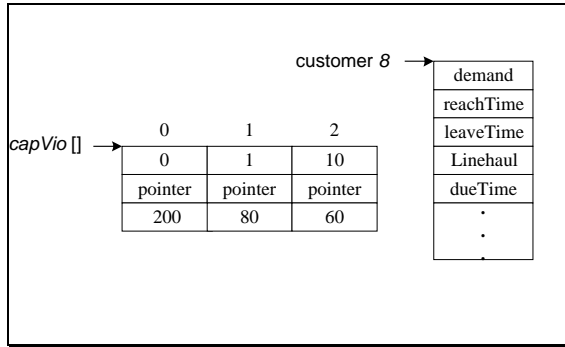


Figure 3.2 – Data structure for `capViol[]`

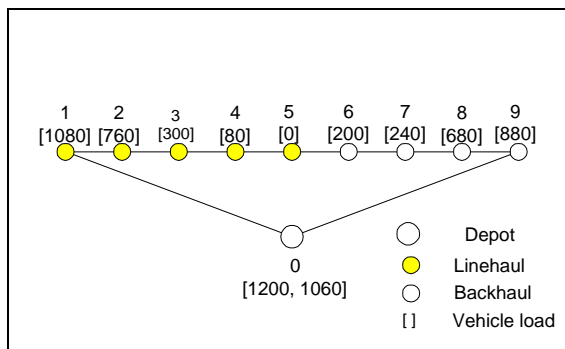


Figure 3.3 – Sample route for `capViol[]` discussion

The second advantage of this data structure is that it can increase the efficiency of feasibility enhancement. To simplify the discussion, the consideration of time window is ignored here (a more sophisticated technique to do feasibility enhancement with time windows will be discussed later). Note that the infeasibility will be decreased gradually. By surveying the capacity violating customers in this route (in figure 3.3), we identify the most capacity-violated node to be the depot at the beginning of the route. Then the strategy is to delete one linehaul customer from this route. The total number of search nodes is 5.

In the VRPBTW without customer precedence, backhaul customers can be inserted into wherever solution is feasible, that is, the vehicle load is less than or equal to vehicle capacity, and the time window constraint is not violated. In this case, the vehicle load is often larger than $\max(ID_i, bD_i)$. Each time when evaluating the local move, capacity violation must be checked in detail. The data structure `lineDmd[]` is still useful, since it indicates the vehicle load when the vehicle first leaves the depot. The data structure `capViol[]` is also used since,

- 1) When deleting a customer from a route, the information in *capVio*[] is enough to evaluate the capacity violation change;
- 2) It also supplies quick information to do feasibility enhancement later.

Customer-precedence-related

The data structure in this section is specifically for the VRPBTW with customer precedence. Before the solution reaches feasibility, backhaul customers can be allocated before linehaul customers. To evaluate the precedence violation of each route, the array *custPrecedence*[] is used. The rule to evaluate the customer precedence violation is as follows. Search from the beginning of the route forward, and find how many groups of linehaul customers are within backhaul customers. The number of groups is the index of precedence violation for the route. For example, in Figure 3.4 there are two groups of linehaul customers {(4, 5), (7)} bracketed by backhaul customers; the customer precedence violation is thus 2.

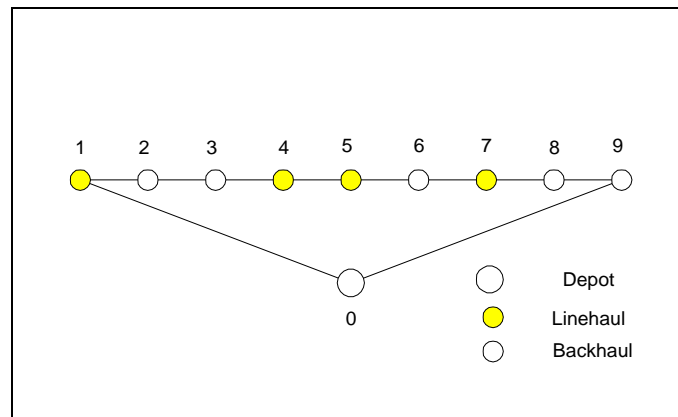


Figure 3.4 – Route with customer precedence violation = 2

Two topics of concern include 1) is this definition a good guide for the local search? 2) Is this efficient? To the first question, consider two situations in figure 3.5 (A) and (B). Case B is in a much more ordered situation than case A, although total linehaul customers after backhaul customers is four for both case. Following the rule proposed above, the precedence violation index is 4 for case A, but only 1 for case B. To answer the second question, this is efficient since we need not search the whole route for the information of precedence violation during each local move. For instance, suppose we need to evaluate the

moving *customer 3* in *route A* (in Figure 3.5) to the position between *customer 3* and *customer 4* in *route B* (in Figure 3.6). According to the data structure, we know that *customer 2* and *customer 4* in *route A* are backhaul customers, while *customer 3* and *customer 4* in *route B* are linehaul and backhaul customers, respectively. It is simple to calculate that the precedence violation in *route A* decreases by 1, while the precedence violation in *route B* remains the same.

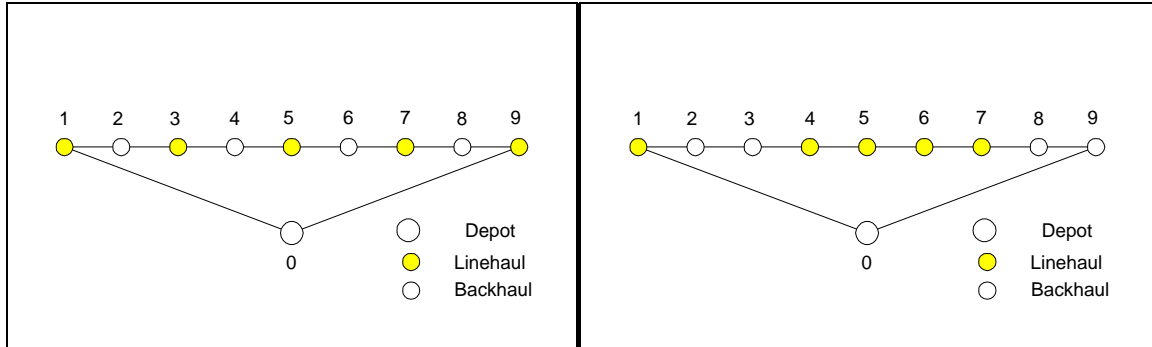


Figure 3.5 – Route A

Figure 3.6 – Route B

3.5 Algorithm

This section introduces the overall algorithm for both the VRPBTW with precedence and the VRPBTW without precedence. Since the main structure of algorithm for these two problems is similar, they are introduced together, with differences noted.

3.5.1 Phase I

Phase I has four stages:

1. Clustering and initial route construction
2. Route improvement: 2-opt (within each route)
3. Route improvement: 1-move (between pairs of routes)
4. Route improvement: 1-exchange (between pairs of routes)

Clustering and initial route construction

Customers cluster are developed based on the adapted sweep algorithm, which assign customers to clusters according to their polar angles. The start angle is a randomly generated number between 0 and $\pi/2$. The tightness of the route capacity is a random number as well, which ranges from 0.6 to 1.0 times the vehicle capacity. The purpose of designing the start angle and tightness of route capacity is to diversify the initial route construction. The sweep procedure is as follows. From the start angle, scan counterclockwise to include customers into the route, once the route capacity exceeds the defined tightness (whether linehaul demand or backhaul demand), assign the set of customers to the same cluster. Continue to scan until all the customers are clustered. The initial routes are generated from the result of clustering. The algorithm divides the customers in each cluster into two sub-zones with equal polar angle. Each customers in each sub-zone are added nto route according to polar angle.

According to the geographical dispersion of the customers and their various properties, the quality of the initial solution based on different start angles and different capacity tightness varies greatly. The algorithm allow the user to construct up to 50 different initial solutions, then select one for further operations. The general rules to select the initial solution are:

1. Always select an initial solution in which the number of routes less or equal than the number necessary if the minimum number of routes is required;
2. From the violation index provided, select an initial solution with small violations of the due time, the capacity, and the customer precedence;
3. If there are still several good initial candidates, choose randomly.

Route improvement

This section describes how the GLS guides the local search heuristics to improve the initial infeasible solution into a much more ordered structure.

2-opt exchange is designed for decreasing time precedence violations (and customer precedence violations for the VRPBTW with customer precedence) in each route. Once the 2-opt heuristic is done, the order of customer according to their time precedence (and the customer precedence), reach its lower bound

level. So, 2-opt is only used at the beginning of the algorithm. 1-move exchange is the second step to carry out when 2-opt can not improve the augmented objective value, and it stops when it can not significantly improve the augmented objective value. Finally, 1-exchange is used.

The GLS first set the initial value of the penalty parameters then does the 1-move and 1-exchange with a predetermined number of iterations. The parameters will automatically increase to decrease the solution infeasibility after each iteration.

3.5.2 Phase II

Phase II is an iterative procedure that adds a user-specified number of new routes in an effort to achieve feasibility. Each iteration has two main steps: route feasibility enhancement and route improvement (see Table 3.1). For each iteration of the feasibility enhancement step, one new route (consisting of customer deleted from existing routes) is added to the solution. Some tips for feasibility enhancement:

- For the VRPBTW with precedence, always delete customer-precedence-violating customers first
- For both case, delete obviously unsuitable customers from each route before doing any other operations;
- When approaching a feasible solution, try to enhance one or two routes' feasibility at each iteration (using section planning).

Instead of changing the solution to be feasible in one step, the algorithm gradually adds additional routes, and does the route improvement at the same time.

Table 3.2 - Pseudocode for phase II

```
For k=1 to NumberOfNewRoutes (a parameter)
  For i = 1 to NumberOfRoutes
    Delete precedence violating customers
    Delete "big" violators of capacity and time windows
    Do Section Planning
  Next I
  Add one new route comprising deleted customers
  Update info
  For j = 1 to NumberOfIterations (a parameter)
    Do route improvement
    Update info
  Next j
Next k
```

Section Planning for route feasibility enhancement

The goal of the route feasibility enhancement is, first, to increase the feasibility of the solution, which is to decrease the time violation and capacity violation of each route; second, to maintain the flexibility for route improvement procedures, which is dominated by the parameter updating mechanism. The contents discussed previously in the "special data structure" section showed the usefulness of the data structure. This section introduces Section Planning, a new feasibility enhancement technique which uses these special data structures.

The idea of Section Planning is to divide each route into several sections, and to try to delete the customer in each section that most contributes to the violation of the constraints. According to the adapted sweep algorithm used, the number of capacity-violating customer may not as much as the number of time-window-violation customers. So the number of capacity-violating customers may be used as a good criterion for the number of sections in a route. When there is no capacity-violating customer in a route, the number of ready-time-violation customers may be a good criterion for the number of the sections in a route,

because the ready-time-violation-customers will stop the benefits of deleting one customer from the section. When only due-time-violation customers exist, the whole route is regarded as one section.

Table 3.3 - Pseudo code for section planning a route

```
Define the sections in the route
If a capacity-violating customer (capVio) exists then
  For each section
    Let n = customer with largest due time violation (timeVio) in this section
    If (find backhaul customer before customer n that best reduces capacity violation)
      Delete the backhaul customer
      Update info
    Else
      Let n = customer with largest due time violation (timeVio) in next section
      Delete linehaul customer before customer n that best reduces capacity violation
      Update info
    Endif
  Next
Else If a ready time violating customer (timeVioE) exists then
  For each section
    Let n = customer with largest due time violation (timeVio) in this section
    Delete the customer before customer n that best reduces the due time violation
    Update info
  Next
Else If a due time violating customer (timeVio) exists then
  For each section
    Let n = customer with largest due time violation (timeVio) in the entire route
    Delete the customer before customer n that best reduces the due time violation
    Update info
  Next
Else
  This route is feasible
End If
```

CHAPTER 4 - EXPERIMENTS

The two versions of the heuristics (for VRPBTW with and without customer precedence) were tested on problems sets derived from Solomon's data sets R1, R2, C2, and RC2. In Solomon's original data sets, all the problems are Euclidean with a single service type. Each of the data sets contains between eight and twelve 100-node problems over a service area defined on a 100 by 100 grid. For R1 and R2, the customer locations were generated uniformly over the service area. Set C2 has clustered customers and set RC2 has a combination of clustered and randomly placed customer. In addition, R1 has a short scheduling horizon and a vehicle capacity of 200 units; R2, C2 and RC2 have long scheduling horizons and vehicle capacity of 1000, 700, and 1000 units, respectively. The time window constraints in problem set R1 allow only a small number of customers to be served by each vehicle. The opposite is true for R2, C2, and RC2. In all cases the vehicle capacity is fairly loose.

The heuristic for VRPBTW without customer precedence was tested on three data sets created by .Kontoravdis, et al. (1995). They constructed data sets MR2, MC2, and MRC2 based on Solomon's R2, C2, and RC2 data sets, respectively, by designating customers as either linehaul or backhaul with equal probability. Their restriction to R2, C2, and RC2 "was motivated by the fact that the time window constraints are dominant for R1, C1 and RC1, and thus the solution state space would not be greatly affected after changing half of the customers to backhaul". Nevertheless, the vehicle capacity was reduced from 1000 to 250 units to assure that the capacity constraint had the primary influence on feasibility.

The heuristic for VRPBTW with customer precedence was tested on 45 data sets created by Gelinas et al. (1995). They used the first five problems of the R1 data set to construct test problems by randomly choosing 10%, 30%, and 50% of the 100 customers to be backhaul customers, leaving other attributes unchanged. They generated additional test problems by considering only the first 25 and first 50 nodes.

Both of the heuristics (for VRPBTW with and without customer precedence) were programmed in Microsoft Visual C++ 6.00 and executed on a 450 MHz PC.

4.1 VRPBTW without customer precedence

The results for Kontoravdis' problem set are summarized in Table 4.1 to 4.3. Two different runs were performed on each problem and the best solution was selected at the end. Kontoravdis' solution (GRASP) for individual problem is not available. The GRASP solution showed here is the average of each data set. The solutions of our heuristics, including the number of routes, distance, and computation time (in seconds) are shown for each problem. GRASP was implemented in C and run on a SUN SPARC-10 workstation.

The results show that on data set MC2, the new heuristic did not get the minimal number of routes as GRASP did, but the total distance is around 19% shorter than obtained by GRASP. On set MR2, the new heuristic got the same number of routes as GRASP, and saved around 15% of the total length. On set MRC2, the new heuristic obtains a solution with fewer routes and 10% less travel distance.

Table 4.1 Results for MC2 data set

<i>Problems</i>	<i>GRASP</i>			<i>GLS</i>		
	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
MC201	-	-	-	5	763.88	57
MC202	-	-	-	4	1186.24	347
MC203	-	-	-	4	1096.31	323
MC204	-	-	-	4	885.73	431
MC205	-	-	-	5	781.70	169
MC206	-	-	-	5	860.74	352
MC207	-	-	-	5	792.96	476
MC208	-	-	-	5	859.92	393
Average	4	1094.94	130.1	4.625	903.56	318.5

Table 4.2 Results for MR2 data set

<i>Problems</i>	<i>GRASP</i>			<i>GLS</i>		
	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
MR201	-	-	-	4	1388.73	108
MR202	-	-	-	4	1198.99	422
MR203	-	-	-	4	988.92	544
MR204	-	-	-	4	858.32	450
MR205	-	-	-	4	1172.53	253
MR206	-	-	-	4	979.50	406
MR207	-	-	-	4	912.69	451
MR208	-	-	-	4	764.52	408
MR209	-	-	-	4	978.82	516
MR210	-	-	-	4	1061.36	557
MR211	-	-	-	4	878.81	692
Average	4	1168.53	122.3	4	1016.66	437

Table 4.3 Results for MRC2 data set

<i>Problems</i>	<i>GRASP</i>			<i>GLS</i>		
	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
MRC201	-	-	-	5	1498.90	73
MRC202	-	-	-	4	1539.41	493
MRC203	-	-	-	4	1303.48	713
MRC204	-	-	-	4	932.48	472
MRC205	-	-	-	4	1632.04	362
MRC206	-	-	-	4	1433.43	208
MRC207	-	-	-	4	1217.2	599
MRC208	-	-	-	4	1085.57	1134
Average	4.5	1496.91	135.7	4.125	1330.31	506.8

4.2 VRPBTW with customer precedence

The test results for the heuristic of VRPBTW with customer precedence are reported in Table 4.4 to 4.8. Five different runs were performed on each problem and the best solution was selected at the end. Results from Potvin's genetic algorithm (run on a Sun SPARC10 workstation) are also reported. On

average, our results are worse than Potvin's solutions, but still within 5% of the optimal solution. One good point is that the new heuristic executes very quickly.

Table 4.4 Results for problem derived from R101

<i>Number of Customers</i>	<i>Back-haul %</i>	<i>Optimal solution</i>	<i>Potvin's GA</i>			<i>GLS</i>		
			<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
25	10	643.4	9	643.4	5.9	9	665.24	2
	30	711.1	10	721.8	5.6	10	723.19	4
	50	674.5	10	682.3	5.6	10	684.52	3
50	10	1122.3	14	1138.1	16.2	14	1178.73	6
	30	1191.5	16	1192.7	16.9	16	1332.44	8
	50	1168.6	16	1183.9	17.5	16	1237.65	6
100	10	1767.9	23	1815.0	169.6	24	1848.04	25
	30	1877.6	23	1896.6	163.3	24	2034.61	27
	50	1895.1	24	1905.9	211.6	25	2057.05	27

Table 4.5 Results for problem derived from R102

<i>Number of Customers</i>	<i>Back-haul %</i>	<i>Optimal solution</i>	<i>Potvin's GA</i>			<i>GLS</i>		
			<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
25	10	563.5	7	563.5	6.3	7	566.66	6
	30	622.3	9	622.3	5.9	9	623.52	5
	50	584.4	8	584.4	6.0	8	593.91	4
50	10	974.7	12	976.8	17.9	12	1017.40	11
	30	1024.8	13	1029.2	18.1	14	1125.38	29
	50	1057.2	14	1059.7	17.9	14	1107.78	27
100	10	1600.5	20	1622.9	206.5	20	1680.90	92
	30	1639.2	20	1688.1	212.9	21	1766.35	179
	50	1721.3	21	1735.7	250.8	21	1806.52	171

Table 4.6 Results for problem derived from R103

<i>Number of Customers</i>	<i>Back-haul %</i>	<i>Optimal solution</i>	<i>Potvin's GA</i>			<i>GLS</i>		
			<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
25	10	476.6	5	476.6	6.2	5	491.85	4
	30	507.0	7	507.0	5.9	7	508.31	5
	50	475.6	6	483.3	5.9	6	494.20	3
50	10	811.4	9	813.3	18.2	9	877.50	16
	30	882.8	10	892.7	18.0	10	919.74	30
	50	882.1	10	885.5	16.8	10	916.56	31
100	10	-	16	1343.7	220.1	17	1384.88	228
	30	-	15	1381.6	197.6	17	1425.83	320
	50	-	17	1456.6	210.6	16	1486.56	150

Table 4.7 Results for problem derived from R104

<i>Number of Customers</i>	<i>Back-haul %</i>	<i>Optimal solution</i>	<i>Potvin's GA</i>			<i>GLS</i>		
			<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
25	10	452.5	5	452.8	6.5	5	457.35	5
	30	467.6	6	468.5	6.2	5	487.85	5
	50	446.8	5	446.8	6.4	5	492.13	5
50	10	-	6	689.2	20.2	7	710.38	41
	30	-	7	751.5	19.0	7	796.91	19
	50	733.6	7	741.4	16.6	7	768.91	18
100	10	-	12	1117.7	244.5	12	1190.26	431
	30	-	12	1169.1	195.3	13	1213.92	215
	50	-	13	1203.7	185.4	13	1209.38	490

Table 4.8 Results for problem derived from R105

<i>Number of Customers</i>	<i>Back-haul %</i>	<i>Optimal solution</i>	<i>Potvin's GA</i>			<i>GLS</i>		
			<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>	<i>Routes</i>	<i>Distance</i>	<i>CPU Time</i>
25	10	565.1	7	565.1	5.6	7	577.85	2
	30	623.5	8	630.2	6.8	8	635.62	6
	50	591.1	7	592.1	5.4	7	618.20	3
50	10	970.6	10	1002.5	14.9	10	1041.89	2
	30	1007.5	11	1047.8	15.2	10	1089.00	8
	50	993.4	11	1018.0	15.4	11	1048.87	3
100	10	-	17	1621.0	191.4	17	1590.54	34
	30	-	16	1652.8	210.6	17	1667.92	55
	50	-	18	1706.7	160.4	19	1699.88	64

CHAPTER 5 - CONCLUSIONS AND RECOMMENDATIONS

This research developed simple but efficient heuristics to solve vehicle routing problems with backhauls and time windows (VRPBTW). This research introduces guided local search (GLS), combined with the Lagrangean method, into the area of VRPBTW. We also create a new way to get an effective feasible solution from initial infeasible solution. Initially, the heuristic was designed to solve the VRPBTW without customer precedence; this heuristic was modified to solve the VRPBTW with customer precedence. The initial heuristic outperforms existing heuristics in finding quality solutions for the VRPBTW without customer precedence. The modified heuristic, adapted to solve the VRPBTW with customer precedence, does not provide as good of solutions as existing heuristics, but does get within 5% of optimality in a short period of time.

In the course of the research, the new heuristic was found to be relatively inefficient in solving VRPBTW problems with loose time windows. This is a common weakness of existing heuristics. Unfortunately, loose time windows are a common feature of real-life industrial problems (Cheng, 2000). The primary research recommendation of this report is to adapt Potvin's GA and the newly developed heuristic to solve VRPBTW problems with loose time windows.

REFERENCES

- Aarts C. and J. K. Lenstra (Editors), *Local Search in Combinatorial Optimization*, John Wiley & Sons Ltd., 1997
- Bodin L. et al, "Routing and scheduling of vehicles and crews", *Computers and Operations Research*, Vol. 10, No.2, 1983.
- Casco D., B. Golden, and E. Wasil. "Vehicle routing with backhauls: models, algorithms and case studies", in: B. L. Golden and A. A. Assad (eds.) *Vehicle routing: Methods and studies*, North-Holland, Amsterdam, pp. 127-147. 1988.
- Cheng, T. Personal communication, J. B. Hunt, 2000.
- Chiang, W. C. and R. A. Russell, "A reactive tabu search metaheuristic for the vehicle routing problem with time windows", *INFORMS Journal on Computing*, Vol. 9, No. 4, Fall 1997.
- Croes A., "A method for solving traveling salesman problems," *Operations Research*, Vol. 5, 1958.
- Deif, I., Bodin, L., "Extension of the Clarke and Wright algorithm for solving the vehicle routing problem with Backhauling". In: Didder, A. (Ed.), *Proceedings of the Babson Conference on Software Uses in Transportation and Logistic Management*. Babson Park, pp. 75-96. 1984.
- Duhamel, C., J. Y., Potvin, and, J. M. Rousseau, "A tabu search heuristic for the vehicle routing problem with backhauls and time windows", *Transportation Science*, Vol. 31, No. 1, Feb. 1997.
- Fisher, M. L., and R. Jaikumar, "A generalized assignment heuristic for vehicle routing", *NETWORKS*, Vol. 11, 1981.
- Gelinas, S., Desrochers, M., Desrosiers, J., Solomon, M., "A new branching strategy for time constrained routing problems with application to backhauling", *Annals of Operations Research*, Vol. 61, 91-109, 1995.
- Goetschalckx, M., Jacobs-Blecha, C., "The vehicle routing problem with backhauls", *European Journal of Operational Research* 42, 39-51. 1989.
- Golden B. L. and A. A. Assad (Editors), *Vehicle Routing: Methods and Studies*, Elsevier Science Publishers B. V. (North-Holland), 1988.
- Golden B, E. Baker, J. Alfaro, and J. Schaffer, "The vehicle routing problem with backhauling: two approaches", *Proceedings of the Twenty-First Annual Meeting of S. E. TIMES* (R. D. Hammesfahr, editor), Myrtle Beach, South Carolina, pp. 90-92. 1985.
- Jacobs-Blecha, C., Goetschalckx, M., "The vehicle routing problem with backhauls: Properties and solution algorithms", Technical report MHRC-TR-88-13, Georgia Institute of Technology, 1993.
- Kontoravdis G. and J. F. Bard, "A GRASP for the vehicle routing problem with time windows", *ORSA Journal on Computing*, Vol. 7, No. 1, Winter 1995.
- Osman, I. H., "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem", *Annals of Operations Research*, Vol. 40, No. 1, 1993.
- Potvin, J. Y., T. Kervahut, B. L. Garcia, and J. M. Rousseau. "The vehicle routing problem with time windows, part I: tabu search", *INFORMS Journal on Computing*, Vol.8, No.2, Spring 1996, pp. 158-164. 1996.

Potvin, J. Y. and S. Bengio, "The vehicle routing problem with time windows, part II: genetic search", *INFORMS Journal on Computing*, Vol. 8, No. 2, Spring 1996.

Reeves C. R., Genetic algorithms, in: C. R. Reeves (Ed.), *Modern Heuristic Techniques for Combinatorial Problems*, Blackwell Scientific Publications, Oxford, 1993.

Russell R. A., "Hybrid heuristics for the vehicle routing problem with time windows", *Transportation Science*, Vol. 29, No. 2, May 1995.

Solomon, M. M., "Algorithm for the vehicle routing and scheduling problem with time window constraints", *Operations Research*, Vol. 35, 1987.

Taha, H., *Operations Research*, sixth edition, Prentice-Hall, Inc, Upper Saddle River, New Jersey, 1997.

Toth, P., Vigo, D., "A heuristic algorithm for the vehicle routing problem with backhauls". In: Bianco,

Toth, P. (Eds.), *Advanced Methods in Transportation Analysis: Proc. Of the 2nd TRISTAN Conference*, Springer, Berlin, 1996.

Voudouris C. and E. Tsang, "Guided local search and its application to the traveling salesman problem", *European Journal of Operations Research*, Vol. 113, 1999.