UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Master of Science in Computer Science

Final thesis

Design and development of Dynamic Fleet Management prototype

1^{st} Reader:
Prof. Alberto Montresor
Università degli Studi di Trento
Trento, Italy

Supervisor:
Dr. Nitin Maslekar
NEC Europe L.t.d
Heidelberg, Germany

Graduand:
Alemnew Sheferaw Asrese

Academic year 2012/2013

To my true lovely parents, Tena Yihunie & Sheferaw Asrese
who sacrified their lives to make me a better person. I am so lucky
and blessed to be your son.

Acknowledgements

First and foremost, thanks be unto God the Alpha and Omega for all His unspeakable gifts.
This thesis would have been impossible without other peoples involvement. Thus I would like to take this opportunity to express my warm gratitude for them.
My great appreciation and thank goes to my wonderful supervisor Dr. Nitin Maslekar. I would like to thank you for taking your time to correct my mistakes, answer my countless questions, and commenting the report.
I would also like to thank you Professor Alberto Montresor for your continuous guidance, suggestions and commenting the report.
I would like to thank Mr. Konstantinos Gkiotsalitis, Mrs. Maria Goleva and other ITS group members of NEC Laboratories. I have learned a lot from you.
I would like to thank Dr. Daniel Yacob for your support to start my study at University of Trento. Without your generous support I might not be here to write this thesis. Starting from our very first email I have learnt a lot from you. I will never forget it throughout my life.
I would also like to appreciate all friends in Trento and around Heidelberg. You made my stays so pleasant.
Last but not least, I would like to express my special thanks to my families my mother Tena Yihunie, my father Sheferaw Asrese, sisters, and brothers. Your love and support in all my school life is not something to express in words.

Contents

List of Figures

Abbreviations

DVRP Dynamic Vehicle Routing Problem

KPI key performance indicators

TW Time Window

VRP Vehicle Routing Problem

VRPTW Vehicle Routing Problem with Time Window

Chapter 1

Introduction

Vehicle Routing Problem is a combinatorial optimization problem which can be defined as the process of finding the least cost for a route of the vehicles to visit/service a set of geographically scattered points or customers starting from one or more points called the depot.
The routes of the vehicles are planned initially based on the available requests which are disclosed in advance. But the requests may be revealed while the planned routes are on execution. In this case dynamic routing of the vehicles is required so as to include those new requests in to the ongoing plan. This kind of variant of the VRP is called DVRP.

1.1 Statement of the problem

Nowadays, scheduling is one of the main challenges in the field of transportation and distribution networks. It is quite hard to obtain an optimal route for a fleet of vehicles with a minimum cost and without breaking operation constraints. For instance, in a postal delivery service the post office wants to deliver a set of postal mails or packages to its customers with a minimum cost (the cost includes travel distance, travel time, the number of postmen required, e.t.c). The customer also wants to receive his/her postal package within a certain time period. The post office would always ask "how can we find the optimal route that minimizes the cost?" Another possible question which the post office may ask is: "which is the

best route with a minimum cost so that the customers packages can be delivered within a specific time period and the post office maximize its profit?" These and other similar questions need to be answered.
This kind of problem gets even worse when there are an immediate requests, and in the case of dynamism in general. This dynamism might be caused, when:

- There is a missed delivery of services of customers and a re-delivery is required.
- There is a change in the location of customers in which the information is disclosed after the initial planning is done or even after the vehicle started its trip. The customer may wants to get the delivery to his/her new location.
- The demand of the customers may not be known in advance to the scheduling, and the request is revealed after the routing of the vehicles is done.
- There is a traffic jam in the road of the vehicles. This affects the vehicle plan to offer the required service to its customers.
- There is a breakdown of vehicles after leaving the depot.

When one or more of the aforementioned dynamism causes take place, it is necessary to route the fleet of vehicles dynamically so as to provide the required service to the customer.
The general problem that we consider in this thesis is: how to design and develop a dynamic fleet management system. The following are the specific questions which we are dealing with.

1. Can we design and develop an algorithm which solves the routing of vehicles in general?
2. Can we design an efficient algorithm that solves the routing of the vehicles in the case of dynamism?
3. Does the algorithm that we develop provides us the expected output? That is, a possible optimal route of vehicles with a minimum cost.
4. Is it possible to obtain an optimal route for the vehicles in a minimum computational time and resource?

1.2 Objective of the thesis

The main objective of this thesis is to design and develop a dynamic fleet management system prototype which can help to route a vehicle dynamically. This thesis focuses on the following specific tasks.

- Review the existing works on VRP in general and more specifically on DVRP.
- Select one among the available algorithms and implement a prototype for solving the static-VRP problem.
- Extend the implemented prototype of the static-VRP to tackle the dynamic variant of the problem. More specifically we focus on the first two dynamism causes mentioned under the problem definition in section 1.1.
- Perform an experimental tests on the prototype and measure the performance and scalability.
- Forward a conclusion and insights for a future research direction.

1.3 Outline of the thesis

The rest part of this thesis is organized as follows. Chapter 2 will introduce the VRP in general. It will describe the applications of VRP, the different variants of VRP, the methods and algorithms employed to solve VRP. It will also explain some related works to DVRP.
Chapter 3 will illustrate the methods and algorithms that will be used to solve the DVRP. Our proposed algorithm and dynamic scheduling will be presented in this chapter.
Chapter 4 will present the experimental results performed in our prototype. Also it will describe the implementation of our prototype, the testing data and the key performance indicators (KPI) used to evaluate our algorithm.
Chapter 5 will give a conclusion and an insight to future research direction in the area, more specifically to the continuation of this work.

Chapter 2

State of the Art

The Vehicle Routing Problem (VRP) is a combinatorial optimization problem which was first introduced by Dantzig and Ramser in 1959 as a generalization of Travel Salesmen Problem (TSP) which was defined in the early 19^{th} century. Vehicle Routing Problem can be defined as a process to compute a least cost routes for a fleet of vehicles to visit geographically scattered points, cities or customers starting from one or more depots [1].

In the past five decades a great attention has been given for VRP and thousands of papers have been published about it. Specially in the area of Operational Research, Transportation, Logistics, Computer Science and mathematics an extensive research has been done to solve the problem. Several algorithms ranging from exact methods to meta-heuristic algorithms have been developed and used. The Vehicle Routing Problem is very essential in every day activities of both private and public sectors especially in distribution networks and transportation domain. VRP application is not limited to but also includes distribution plan for wholesaler order, garbage disposal, mail delivery, mailbox collection, repairmen scheduling, school bus routing, bank deliveries, snow ploughing, e.t.c.

The remaining part of this chapter is organized as follows. The first section formally describes the Vehicle Routing Problem . The variants of the Vehicle Routing Problem is described in section two. The third section is about the algorithms used in Vehicle Routing Problem . A Dynamic variant of the problem is described in the forth section.

2.1 Overview of the Vehicle Routing Problem

The Vehicle Routing Problem is of a class of NP-hard problems [2]. VRP in general can be defined on a graph $G = (V, A, C)$, where $V = \{v_0, ..., v_n\}$ are the set of vertices [1]; $A = \{(v_i, v_j) \mid (v_i, v_j) \in V^2, i \neq j\}$ is the set of arcs; and $C = (C_{ij})_{(v_i,v_j)\in A}$ is a cost matrix defined over A. In convention, vertex v_0 represents the depot and the remaining part of the set V represents the customers that need a service. The cost matrix C represents the travel cost, the distance between pairs of vertices, or the travel time. VRP is then finding an optimal route for K identical vehicles that starts from the depot and visits all the other vertices exactly once by exactly one vehicle with a minimum cost and returns back to the depot [3].

VRP can be also defined as designing an optimal delivery route with a least cost for a given $N + 1$ geographically scattered locations. K identical vehicles with a capacity Q visits N customers with a known non negative demand $(d_i, i = 1, ..., n)$, without breaking a set of operational constraints. In most of the cases

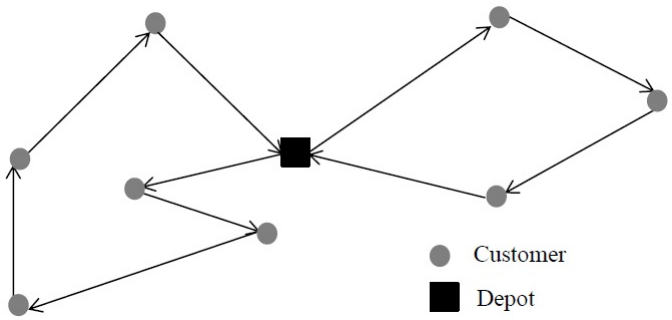

Figure 2.1: A vehicle routing scenario with 8 customers.

the vehicles start at the depot and are required to return back to the depot after visiting the customers. Some of the operational constraints are:

- The total sum of the demand of the customers in a given single route should not exceed the capacity of the vehicle assigned.
- Every customer should be visited exactly once by exactly one vehicle.

[1] unless it is specified explicitly, throughout this document vertice, customer and location are used interchangeably.

- Each vehicle in a route should return back to the depot after finishing serving the customers.
- The length of a route may not exceed a prescribed time bound L. This length made up of the intercity travel time and of the stopping time at each customer in the route.
- A customer i may need to be visited in a specified time interval [4].

This definition and the constraints are somehow relaxed in some variants of the problem.

2.2 Variants of Vehicle Routing Problem

The Vehicle Routing Problem is extended by considering different operation constraints. As a result different variants of the problem are available. In this section we describe briefly some of the most commonly used variants.

- **Capacitated Vehicle Routing Problem (CVRP)** : this variant of the problem is concerned with the vehicle capacity constraint. The total sum of the the customer's demand in a route should not exceed the total capacity of the assigned vehicle.
  Mathematically: $\sum_{i=1}^{n} d_i \leq Q_k$; where Q_k is capacity of vehicle k, and d_i is the demand of customer i .
- **Distance constrained Vehicle Routing Problem (DCVRP)**: in this class of Vehicle Routing Problem the total length of the route that a vehicle can travel may not exceed a fixed limit L.
- **Vehicle Routing Problem with time window (VRPTW)**: when the delivery at all or some of the customers is within a pre-specified time interval, the VRP becomes VRPTW. Vehicles may arrive earlier to the service starting time. In case the vehicle waits until the starting time reaches. If the vehicle arrives out of the specified time window, it can be accepted with cost in soft time windows. While in hard time windows the vehicles are not allowed to arrive out of the time window [2].

- **Vehicle Routing Problem with backhauls (VRPB)**: when a route of the vehicle contains both deliveries (called linehauls) and then collections (called backhauls) the VRP problem called VRP with Backhauls [2]. The linehauls picked up at the depot and delivered to the customers, while the backhauls collected from the customers and returned to the depot.

- **Heterogeneous fleet Vehicle Routing Problem (HVRP)**: when the fleet of vehicles is composed of different types of vehicles the class of the problem becomes HVRP.

- **Vehicle Routing Problem with Pick and delivery (VRPPD)**: suppose that i and j are locations in the route of a vehicle and when a pickup at location i should be done before the delivery at location j the variant of the problem is called VRP with pickup and delivery.

- **Vehicle Routing Problem with simultaneous pickup & delivery (VRPSPD)**: when the same customer have both delivery and pickup demands the variant of the problem is VRP with simultaneous pickup & delivery. This is a variation of VRPPD in which both are done at the same customer. For instance, in beverage distribution the vehicle delivers the beverages at the customers and collect the bottles of the previous deliveries from the same customer.

- **Vehicle Routing Problem with stochastic demand (VRPSD)**: in this variant the problem the customers demand are assumed to be known as stochastic variables during the time of planning. A plan failure may occur when the total demand in a route exceeds the vehicle capacity and rerouting may be required. VRPSD becomes a DVRP when the demands revealed during the plan execution.

- **Dynamic Vehicle Routing Problem (DVRP)**: in dynamic variant of the VRP some of the customers demands are known in advance before the start of the working day and other new requests arrive as the working day progresses. The system should incorporate these new requests into the schedule. This variant of the problem is described in more detail in the next section.

- **Open Vehicle Routing Problem (OVRP)** : in the other variants of the problem the vehicle are required to return back to the depot after servicing all the customers. While in Open VRP the vehicles are required only to start at the depot and can stop at any customer visited.
- **Close-open mixed Vehicle Routing Problem (COMVRP)**: this variant is the combination of both the CVRP and OVRP variants. It is introduced by Liu and Jiang in 2012 [5]. The characteristics of COMVRP is similar with the CVRP and OVRP. All vehicles starts at the depot. After completing the delivery for CVRP all vehicles are required to return back the depot, while the OVRP vehicles can stop at any customer served at the end [5].
- **Multidepot Vehicle Routing Problem (MDVRP)**: in MDVRP, the vehicles start from multiple depots and should return back to their depot of origin after visiting all customers in the route assigned to.

2.3 Algorithms for Vehicle Routing Problem

Several algorithms ranging from the exact methods to a meta-heuristic have been developed and employed to solve the Vehicle Routing Problem . In this section we briefly describe some of these algorithms.

2.3.1 Exact methods

All known exact algorithms for VRP can solve only small instances of the problem. The exact algorithms can be either in direct search algorithms, dynamic programming, or integer linear programming categories [1]. Since the focus of the thesis is on dynamic part of the problem we do not need to dig into deep on this type of algorithms.

2.3.2 Heuristic algorithms

The heuristic algorithms for the VRP are derived from the procedures employed for the TSP [4]. Heuristic algorithms gives a solution but it is not guaranteed to

be the optimal solution. Some of the heuristic algorithms used to solve VRP are described in this subsection.

1. Constructive heuristic
   This algorithm creates the vehicle routes by merging two or more existing routes or inserting nodes (cities or customers) into an existing route. The most widely used algorithm of this class is the Savings methods which was formulated by Clarke and Wright in 1964 [6]. The savings method can be described as follows.
   The savings method considers the set of customers and calculates the "savings" for every pair of customers. Depending on the savings result obtained, the customers can be served by linking them into a route. When the cost of serving by linking the customers into a route is high, they served separately [2]. This algorithm is described in detail on chapter three.

2. Two-phase heuristic
   As its name indicates the algorithm works in two steps. This algorithm can be seen in two different types.

   - Cluster-first, route-second algorithms: this method cluster the demand first and determine the vehicle the route for each cluster. Fisher and Kaikumar, the Petal algorithm, and the Sweep algorithm are of this type algorithms [7].
   - Route-first, cluster-second Algorithms: these algorithms construct a tour in the first phase without taking into account the side constraints and later on divide the tour in to feasible route of vehicles [8].

2.3.3 Meta-heuristic algorithms

Starting from the 90's different meta-heuristic algorithms have been developed and used to solve the VRP. Most of these algorithms are inspired by natural and environmental processes, learning, and searching mechanisms. We mentioned some of these algorithms in this subsection.

1. Ant colony algorithm:
   This optimization algorithm inspired by the ants behavior of foraging for

food. When ants find a food in a random exploration of the surrounding they return to the nest by depositing a substance called pheromone trial. When other ants smells this pheromone they stat following the same path to hunt for food [9]. Ant colony algorithm is a technique to solve a combinatorial optimization problems by the use of artificial ants based on the behavior of the real ants. They have an ability to remember the past action and the knowledge about the distance to the other location. In the routes for the ants (represent vehicles) constructed incidentally by selecting the next customers until all customers are visited or the capacity of the vehicle is full. The ants pheromone trials is updated to improve the future solution [10].

2. Genetic algorithm (GA):
   The researches for genetic algorithm for Vehicle Routing Problem was inspired by the Darwin's evolutionary theory, the survival of the fittest. In GA the customers are considered as a gene, and the string of customers form a chromosome which is the route to be visited by a single vehicle. Initially the vehicles start from the depot and the set of customers in the route is empty. Then the vehicles select adjacent customers at random to form a chromosomes (route) and start serving the customers in the route. When all the adjacent customers are visited or when the vehicle capacity is empty the vehicle returns to the depot in the shortest path. The fitness of a chromosome is evaluated based on the total travel distance, the level of any constraint violation, the number of customers visited, and the total quantity dispatched by the vehicle [11]. A parent solutions are selected at random from the population and by using crossover procedure the offspring are produced from the parents. The one with the better fitness value is selected as the best solution [12]. If the offspring is better the parent will be replaced by the offspring and the process iterates until the best solution found.

3. Tabu search algorithm:
   Tabu search algorithm is a neighbor search mechanism which iterates to find the best solution based on the initial solution produce at random. A

solution is a set of routes in which a route R_r starts from the depot and visits a set of customers and finally return to a depot. The are many variants of this algorithm such as Osman's algorithm, Taillard's algorithm, the Xu and Kelly algorithm, the Rego and Roucairol algorithm, e.t.c. [13].

There are also other types of heuristic algorithms such as Deterministic annealing, Neural Networks, Simulated annealing. However, it is beyond the scope of this study to mention and describe every algorithm.

2.4 Dynamic Vehicle Routing Problem

Nowadays dynamism is everywhere. So does in transportation and distribution networks. Traffic jams, customers location change, missed deliveries, online order request, environmental factors such as road construction, and vehicle availability and breakdown are some of the causes for dynamism in transportation and distribution networks. Dynamic rerouteing of the fleet of vehicles is required to satisfy customers need and reduce operational costs. As a result of the recent advances in communication and information technology, information can be obtained and processed at real time. This advancement enables to solve the dynamism problems in the transportation and distribution networks.

The Dynamic Vehicle Routing Problem is one variant of the VRP in which the requests or the demands of the customer are not known in advance to the route planning. Instead the requests are revealed while the schedule is on execution. The system has to decide whether to accept the request or not. An accepted request must be served. When companies get requests dynamically and if they cannot fulfil the request because the request is too costly or for other reason, they may pass the request to their competitor.

To the best of our knowledge, the common causes of dynamism for vehicle routing studied so far in the literature are the online arrival of the customer request, the travel time and vehicle availability and breakdown. Moreover, the degree of dynamism varies between problems or even within the same instances of a problem. The dynamism of a problem can be measured by two factors. The first one is the frequency of changes, that is, the arrival of the new information. The second

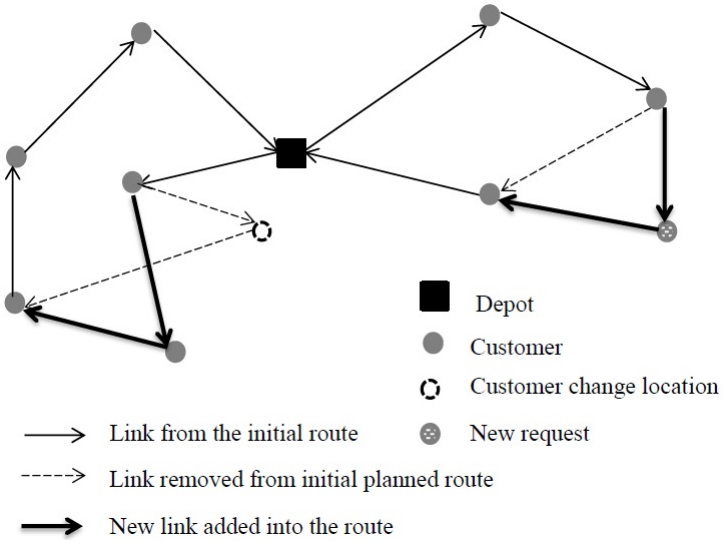

Figure 2.2: A dynamic vehicle routing scenario with 8 advance, 1 immediate request and 1 location changed customers.

one is the urgency of the requests, that is, the time interval between the arrival of the request and the expected service time [14].

Lund et al. defined the degree of dynamism δ as a ratio of the number dynamic requests n_d to the total number of request n_{tot} [14].

Mathematically:

$$\delta = \frac{n_d}{n_{tot}} \tag{2.1}$$

in which the value of δ may vary between 0 and 1. For instance, if δ is equal to 0.5 means that half of the customer requests are dynamic.

[15] defined the effective degree of dynamism δ^e, which represents the average of the disclosure time of the request. The measure of the effective degree of dynamism is defined as:

$$\delta^e = \frac{1}{n_{tot}} \sum_{i=1}^{n_{imm}} (\frac{t_i}{T}) \tag{2.2}$$

where :

- n_{imm} is the number of immediate requests;
- t_i is the time in which the i^{th} request is received, that is, $0 < t_i \leq T$;
- T is the end of the planning horizon that starts at 0;

- n_{tot} is the sum of the number of advance requests and immediate requests;

Larsen [15] also extended the effective degree of dynamism with time window δ^e_{TW} by considering the reaction time. Reaction time can be defined as the difference between the request disclosure time and the latest possible time at which the service of the request should begin. The reaction time for i^{th} immediate request is denoted by r_i and is computed as $r_i = l_i - t_i$, where l_i is the latest possible time the service should begin, and t_i is the time of the i^{th} immediate request received. The higher the reaction time means the more flexibility to insert the request into the current route.

$$\delta^e_{TW} = \frac{1}{n_{tot}} \sum_{i=1}^{n_{tot}} (1 - \frac{r_i}{T}) \tag{2.3}$$

The value of the effective degree of dynamism is in the interval [0, 1]. $\delta^e = 0$ means that the system is pure dynamic, and $\delta^e = 1$ means that the system is pure static. In Dynamic system the requests are received at time 0, and in static systems the requests revealed at time T. [15] used these three parameters to design a framework that classifies dynamic routing problem in to weakly dynamic, moderately dynamic, and strongly dynamic problem.

2.4.1 Related works on DVRP methods

DVRP has been studied since 1977 firstly by Wilson and Colvin [14]. However, it has got more attention in the scientific research community within the last two decades. In this study we consider only some of the recent works done in the field.

Bent et al. [16] used a multiple plan approach (MPA) to solve a dynamic vehicle routing with stochastic request. MPA generates plans continuously based on the current information and remove those plans which does not have the current information. MPA handle four types of events. That are customer request, vehicle departures, plan generation and timeouts. Customer request which update the set of plans to incorporate the new requests to the schedule. [16] used a ranking function to maintain a list of distinguished plans.

Zeimpekis et al. [17] proposed a dynamic fleet management system that comprises

six different modules. The modules are Geographical Information module, decision support module, data management module, vehicle on-board system, control center user interface and vehicle user interface. They mainly emphasized on the system model for dynamically routing the moving vehicles. They did not give any experimental results.

Cheung et al. [18] proposed a framework to solve the dynamic VRP. Their framework first determine the route plan by solving the static VRP before the start of the daily operation and then when a new information arrives then it makes the dynamic routing. To plan the initial routes for the static routing [18] used "seed selection" technique and this solution is refined latter by using genetic algorithm and the final solution is selected. While for making the dynamic routeing plan the framework considers two dynamism causes. The first one is new customer request arrival and the second one is new travel time data arrival. In the former case, it searches the best possible insertion point for the new request in all the existing routes, check the feasibility of the insertion and evaluate the impact of insertion. At this step the refinement procedure used in the static case is applied. In the later case, if the travel time does not affect the feasibility of the current route plan simply the refinement process continues to improve the solution. Otherwise remove those infeasible order requests and consider them as a newly arrived order.

Montemanni et al. [19] designed a new algorithm based on ant colony system for solving DVRP. The dynamism cause they considered in their work is only the arrival of request as the execution of the initial plan progresses. The algorithm they proposed consists of three main elements. An event manage, which collects new orders and keep trace of already served orders and keep the current location of the vehicle. The event manager uses this information to construct a sequence of static VRP like instances which can be solved by the second element of their proposed architecture, that is, the Ant Colony System. The third component, the pheromone construction procedure is used to pass information about the characteristics of a good solution from the static VRP to the following one. [19] sets a cut-off time, and the order requests that arrive after this cut-off time are postponed to the next working day. The static problem considers only unversed customers from the previous day. The working day is divided into n_{ts} time slots,

each one longs $T_{ts} = \frac{T}{n_{ts}}$ seconds, where T is the total length of the working day in seconds. New incoming requests with in the the time slot are considered only at the end of that time slot. They use the time slot concept to limit the time dedicated to each static problem.

Chapter 3

Solving the DVRP: methods and algorithms

In this chapter we describe the algorithm that we proposed to solve the DVRP. Before we present the proposed algorithm we discuss the algorithms that was used as a base algorithm. We use the savings method as a base for solving the static-VRP and add time windows for each route that we get. In order to solve the dynamic routeing of the fleets of vehicle we extend the time window by adding an extra time called slack time. Finally by using this slack time we show the dynamic routeing of the missed customers and the location change of customers.
This chapter is organized as follows. The first section describes the savings method. The second section is about Time Window (TW) concept of VRP. The third section presents our proposed method and algorithm. Section four describes the dynamic routeing algorithm.

3.1 The savings method

As it is already stated in chapter 2 the savings method was formulated by Clarke and Wright. The basic idea behind this algorithm is computing the cost of visiting a pair of customers to decide whether or not to link the customers into a single route and service them with a single vehicle.
Suppose that there are customer i and j located at a symmetric distance of C_{i0}

and C_{0j} respectively from the depot, and C_{ij} between them. If these customers are to be serviced in a separate route the total cost is $2C_{i0} + 2C_{0j}$. While if the routes of the two customers are merged into a singe route these customers can be served with a total cost of $C_{i0} + C_{0j} + C_{ij}$. Hence, the savings S_{ij} of this tour is computed as follow:

$$S_{ij} = C_{i0} + C_{0j} - C_{ij} \tag{3.1}$$

In savings algorithm initially each customer is visited within a separate route. However, it is possible to merge two or more customers route into one and service these customers together so as to reduce the cost. The merging is a repetitive process to get the best cost reduction. It stops when there is no more feasible combination of routes.

Let us consider the following scenario to illustrate how savings method works. Suppose there are ten customers which have a non negative demand. And a vehicle which has a capacity of 70 units should service these customer from a depot "A" located at a location of (35, 35).

Table 3.1: Illustrative example: customers with their demand.

Customer	Location	Demand	Customer	Location	Demand
B	41 , 49	10	G	25 , 30	3
C	35 , 17	7	H	20 , 50	15
D	55 , 45	13	I	10 , 43	9
E	49 , 20	19	J	55 , 60	16
F	15 , 30	26	K	30 , 60	16

By using the euclidean distance formula we can calculate the cost or distance between each pair of customers. Table 3.2 depicts the cost between a pair of customers. Table 3.3 shows the savings of merging a pair of customers into a single route. The savings method starts merging the customers with a highest saving, *i.e.*, customer D and J, only if the sum of their demand do not exceed the vehicle capacity. The merging process continues to the next highest saving value and repeats until all the customers are routed into a route.

Let us see how the routes are being constructed. Customer D and J gives the highest saving and their total demand (*i.e.*, $13 + 16 = 29$) is less than the vehicle capacity. Hence, D and J can be merged into a single route. The resulting

Table 3.2: Cost matrix between pair of customers for the illustrative example

	B	C	D	E	F	G	H	I	J	K
A	15.23	18	22.36	20.52	20.62	11.18	21.49	26.25	32.02	25.49
B		32.56	14.56	30.18	32.20	24.84	21.02	31.58	17.8	15.56
C			34.40	14.32	23.85	16.40	36.24	36.06	47.42	43.29
D				25.70	42.72	33.54	35.57	45.04	15	29.15
E					35.44	26	41.73	45.28	40.45	44.28
F						10	20.62	13.93	50	33.54
G							20.62	19.85	42.43	30.41
H								12.20	36.40	14.14
I									48.10	26.24
J										25

Table 3.3: Savings matrix of merging of a pair of customers for the illustrative example.

	B	C	D	E	F	G	H	I	J
C	0.67								
D	23.03	5.95							
E	5.66	24.2	17.16						
F	3.644	14.76	0.25	5.69					
G	1.57	12.77	7.10	5.69	21.79				
H	15.42	2.96	8.21	0.01	21.21	11.78			
I	9.9	8.17	3.56	1.4	32.93	17.57	35.25		
J	29.44	2.59	39.37	12.08	2.63	0.76	16.82	10.16	
k	25.17	0.20	18.7	1.73	12.57	6.26	32.57	25.49	32.51

route (let us call it route 1) is: `route 1: A - D - J - A`. In the next iteration customer H and I gives a higher saving. And either of them are not merged into any other existing route. So they can be merged if their demand (*i.e.*, 15 + 9 = 24) is less than the vehicle capacity. Of course, it is less than and another new route (let us call it route 2) is constructed in parallel by merging customer H and I. At this step the resulting routes are: `route 1: A - D - J - A` and `route 2: A - H - I - A`. In the third iteration customer F and I give the next higher saving. Meantime customer I is already included in route 2 and it is not an interior point to its route. Since the total demand of customers in route 2 is 24, customer F can be added. Route 2 becomes `A - H - I - F - A` with

a total demand of 50. After elapsing several iterations all the customers will be added to a route and the final routes become:

```
route 1:  A - C - D - J - B - A        Cost = 86.95
route 2:  A - K - H - I - F - H - A    Cost =106.06
```

The savings method uses two approaches to construct the routes. That is, the parallel and the sequential ways. In the parallel approach, a link (so to say a route) is started by merging customers in to a single route. When a link cannot be added to the existing one (*i.e.*, when there is a constraint violation) a new link is immediately created in parallel. The process of merging customers and links continue until all customers are assigned to a link. For instance, the illustrative example presented above is constructed using the parallel approach. While in the sequential approach only one route is created at a time. There is no guarantee to say that one approach performs better than the other. Both approaches gives good results depending on the instances of the problem. For instance, [2] has obtained better result in the sequential approach. Also [2] indicated that previous results shown that the parallel approach had performed better.

A pseudo code of the savings method is shown in algorithm 3.1. In the pseudo code a point (so to say a customer) is said to be an interior (see line 12) to its route if it is not found adjacent to the depot of the route. In line 16 of algorithm 3.1 "at the same position" is to mean that the points found either both at the beginning or both at the end of their respective route. If both points are found either at the beginning or at the end of their respective route, it is not possible to merge these two routes. In the other cases(*i.e.*, either point i is found at the beginning, and point j at the end on their respective routes or vice versa) these routes can be merged unless there is no violation of operation constraints.

Algorithm 1: The saving methods algorithm pseudo code

input : The cost between each pair of n customers and the cost between n customers and the depot

output: A set of routes

```
1  begin
2      Initialize routes
3      List savings ← ∅
4      for i ← 0 to n − 1 do
5          for j ← i + 1 to n do
6              S_ij ← computeSavings(i, j)
7              savings.add(S_ij)
8      savings.sort(descending)
9      foreach S_ij in savings do
10         if i and j not isRouted() then
11             Initialize newRoute
12             newRoute.add(i, j)
13         else if i or j isRouted() and not isInterior() and not
           constraintsViolated() then
14             link i and j
15         else if i and j isRouted() and not sameRoute() then
16             if i and j not isInterior() and not samePosition()and not
               constraintsViolated() then
17                 merge the two routes
```

There are different operation constraints that need to be respected in VRP. Among these vehicle capacity is the major constraints to be considered in using the savings method. In addition truck availability (number of vehicles available for servicing), customer service time, maximum distance that the vehicle can travel, driver working time, etc are also some of the operation constraints to be considered in the different variants of VRP.

The complexity of the savings method is $\mathcal{O}(n^3)$. The savings algorithm starts

by defining one route for each customer, which costs $\mathcal{O}(n)$, where n is the total number of customers. When computing the savings for each customer form line 3 to line 7 of algorithm 3.1 it iterates $n-1$ times and the complexity is $\mathcal{O}(n^2)$. The feasibility checking is done in a constant time. For searching the best combination of routes, that is, from line 8 to line 17 of algorithm 3.1 the complexity is $\mathcal{O}(n)$.

3.2 The time window concept for VRP

As we saw in the previous chapter there are different variants of VRP. For instance, let us consider a case of a postal delivery service. The post office divides the office hour and can assign a specific time period (say, $[e_i, l_i]$) to each customer when they will get their deliveries. That is, the customers should receive their deliveries within the specified time period. As it is already described in the previous chapter, this kind of VRP variation is called Vehicle Routing Problem with Time Window (VRPTW). This time window add a complexity to the problem, and hence VRPTW is also a type of NP-hard problems.

In VRPTW the service of a customer i (for $i = 1, ..., n$) begins at time b_i of the time window specified with an earliest time e_i and a latest time l_i. If a vehicle travels directly from customer i to customer j and arrive too early at j, it has to wait until the service start time of customer j reach. That is, the begin time $b_j = \max(e_j, b_i + s_i + t_{ij})$, where s_i is the service time of customer i and t_{ij} is the travel time between customer i and j [20].

In the following part of this section we show how to assign customers into a time window and latter on we will use it to solve the dynamic variant of the problem. In section 3.1 we used the savings method to solve the static VRP, and we already have got the routes. The route can be divided into a set of time windows and each customer is to be assigned into a time window. The time window assignment for customers in a route works as follows. A working day has OT hour length of operational time. Within this operational time a vehicle v can service m customers in a route k. Therefore we divided the operational time in to equal length of time period tp and assign to customers which are already routed by using the savings method. The earliest service time and latest service time for customers is the beginning and end of the time window respectively. It is clear that anyone

can use any algorithm to construct the initial route for the customers.

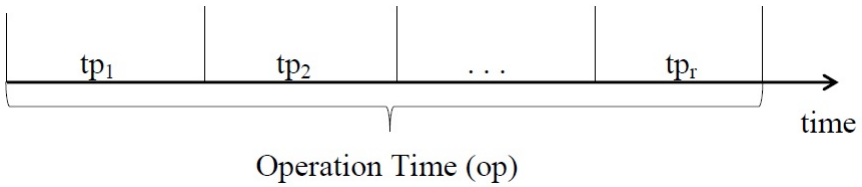

Figure 3.1: Time windows for a give operation time

In order to assign the customers to a time window we need to know the distance between each customers or the travel time it takes from one customer to another one. Once we have the travel time t_{ij} from customer i to customer j, then we can easily decide how many customers is to be assigned to a specific time window. The following pseudo code shows the assignment of customers to a time window.

Algorithm 2: The time window assignment pseudo code

input : A route which has m customers, operation time length OT and the a time window length tp

output: A route divided into time windows

```
1  begin
2      integer s ← ⌊OT/tp⌋
3      timeWindow[s] ← OT.divideIntoTimeWindow(tp)
4      currentTW ← timeWindow[0]
5      integer counter ← 1
6      for i ← 1 to m do
7          if currentTW.full() then
8              currentTW← timeWindow[counter]
9              counter ← counter + 1
10         j ← i − 1
11         if s_i + t_ji + currentTW.time() ≤ tp and j ≠ 0 then
12             currentTW.add(i)
13         else
14             currentTW← timeWindow[counter]
15             currentTW.add(i)
```

In algorithm 3.2, i where $i = 1, ..., m$ refers the customer in a given route k. If the customer is in the first position of the route, the customer is going to be assigned at the first time window. While if the customer is not at the first position of the route the customer is going to be either added to the current time window or assigned to the next time windows depending on the length of its service time and the travel time from its predecessor customer j in the route. That is, the sum of its service time s_i, travel time t_{ji}, and the time length that is required to service the customers already in the time window should be less than or equal to the total length of the time window tp.

3.3 Our proposed algorithm

We have proposed an algorithm that solves the dynamic variant of VRP. The algorithm that we proposed is based on the savings method described in section 3.1 and the time windowing concept described in section 3.2. The algorithm can efficiently solve the dynamic VRP, especially the dynamism which are caused by missed deliveries and customers location change. Let us consider a case of postal delivery service and see the two dynamism causes in more detail.

1. Missed deliveries:
   A postman started to distribute the postal package for customer a, b, c, d, e, and f. Assume the delivery sequence is in alphabetic order. When postmen has arrived at location d he noticed (or be informed) that customer $b's$ package is not delivered. Customer b should get his/her package within that date. Therefore, a dynamic rescheduling is required to service the missed customer b, by finding the best possible insertion point starting from the current location of the postman , that is, the location of customer d. It may be highly expensive to insert customer b from the location of the postman on wards. In this case customer b will get his/her package with another route after the postman returned back to the post office (*i.e.*, the depot).

2. Customer location change:
   The post office informed its customers that he/she is going to receive his/her deliveries at his/her home, say location l_0, within a specific time period

$[t_{begin}, t_{end}]$. However, due to a traffic congestion the postman could not be able to arrive at the customers location within the specified time. A customer left his/her home and gone to his/her work place. In the meanwhile the customer informed the post office as he/she already left his/her original location and his/her postage deliveries shall be delivered to his/her work place at certain location l_1. In such cases the post office has to decide whether or not to deliver the postage deliveries to the new location of the customers. Rerouteing the current vehicle (*i.e.*, the postman) is essentially required to decide whether or not to service the customer by current vehicle. If it is profitable to deliver the deliveries to the new location l_1 and the service of the other customers in the route will not be affected; the deliveries will be done by the current postman. That is, by searching the best feasible insertion point in the route starting from the current location of the postman. Otherwise a new postman is required to service the customers who changed their location.

In both cases the total cost of the service $C_{total} = C_{static} + C_{dynamic}$. Where C_{static} is the cost required to service the customers up to the current location of the vehicle (*i.e.*, where the miss delivery and/or the location change is detected). While $C_{dynamic}$ is the cost required to service the customers after rerouteing starting from the vehicle location.

This problem of rescheduling the customers starting from the current location of the vehicle becomes another variant of VRP. That is, it is a kind of Open VRP, which is already discussed in section 2.2. After the vehicle finish servicing all the customers it is not required to go back to the point where the rerouteing is done. Instead, it has to go to the depot, where it started its tour.

In order to overcome the aforementioned problems it is necessary to reroute the vehicles, and the problem becomes a DVRP. So we need to solve the DVRP. To solve the DVRP different approaches and algorithms have been employed. Some of the recent works has been already discussed in section 2.4.1. By using savings method as a base algorithm to plan the static route at the beginning of the working day and adding a time window for each route we have proposed an algorithm that solves the DVRP. Especially for a dynamic problems which are caused by the two dynamism causes mentioned above. The time windows where

the customers are missed can be easily identified. Therefore, we need to look for another time window (*i.e.*, starting from the current location of the vehicle) which can accommodate to service the missed customers[1] and provide the service if it is feasible to do.
Inserting the missed customers into the planned time windows (*i.e.*, with a constant time window length) may highly affect the service time of the future customers[2]. They may not be serviced within the time windows assigned to them. Therefore, we have to extend the time window length so that it is possible to include the missed customers and the future customers can get their service within the time window assigned to them.
The time window is extended by adding an extra slot time which we call it slack time. By adding a slack time the total length of the time window becomes longer so that it can accommodate other customers without affecting the service time of the customers assigned within that time window. Off course, the missed customers can only be inserted to this time window if it is feasible and profitable to do. In the next subsection we will describe how the slack time is calculated and added to the existing time window.

3.3.1 Slack time

A slack time is a small length of time to be added to elongate the time window. It can have a constant length or a variable length that is computed by considering different constraints. In our algorithm we calculate the slack time by considering other constraints such as the length of the operation time (OT), the time window length (tp), and the number of customers initially assigned to the time window. More details about the calculation of the slack time is given in section 3.3.2. Figure 3.2 shows a time window extended with a constant length slack time. The time window without a slack time extension is already shown in figure 3.1. As it is shown in figure 3.2 the time window length (tp) is longer by an amount of slack time, let us call it st. For instance, in figure 3.2 the length of time window

[1] unless and otherwise it is stated explicitly, missed customer hereafter refers both customers which has a missed deliveries and customer which change their location.

[2] future customer is to refer those customers that are located next to the current location of the vehicle in the current route.

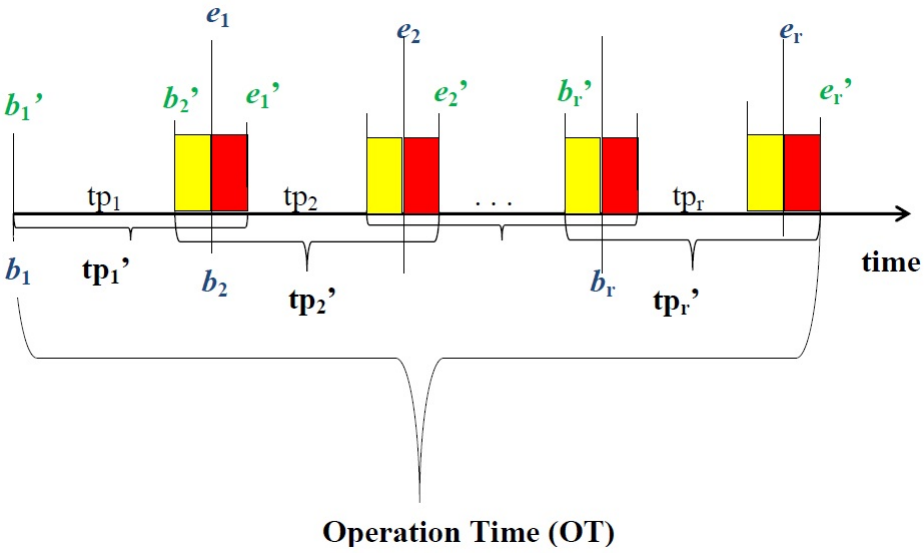

Figure 3.2: Extended time window with a constant length slack time

1 (tp_1) (*i.e.*, where its beginning time is b_1 and its end time is e_1) becomes to tp'_1 (*i.e.*, where its beginning is b'_1 and its end time is e'_1). The same is true for time window r and others.

The new time window length tp' is equals to the sum of the original time window length tp and twice the slack time length st (*i.e.*, one at the beginning and another one at the end of the time window). That is, $tp'_i = tp_i + 2(st)$ where $i = 2, ..., r$. However, for the first time window, the new length $tp'_1 = tp_1 + st$. This is so, because we do not add a slack time at the beginning of the initial time window. Therefore, this clearly shows that tp'_i is greater than tp_i, where $i = 1, ..., r$, and there is a high possibility to insert a missed customer(or a new request) in time window i.

In the next sub section we will describe how to calculate the slack time for the time windows in a route by considering the aforementioned different constraints.

3.3.2 Slack time calculation

A variable length slack time is computed by considering the different side constraints. Some of the basic constraints are listed out in the previous section, see

section 3.3.1. The slack time is computed as follows.
The available total slot time for a route k AST_k is the difference between the length of the operation time OT and the routeing time RT. *i.e.*,

$$AST_k = OT - RT \tag{3.2}$$

where:

- OT is the operation time of a day in hours,
- RT is the sum of the length of the time windows in route k in which customers are already assigned.

The maximum slack time for route $k, MaxST_k$ that can be added into a time windows within a route is calculated as:

$$MaxST_k = \frac{ASK_k \times TP_k}{RT} \tag{3.3}$$

where:

- TP_k is the original time window length in hours, which is the same for all the time windows in a route k.

From equation 3.3 we get the maximum possible slack time that can be added to each time window. However, this equation gives us an equal slack time length for all time windows within a route. This is almost the case of the constant slack time described in section 3.3.1. The only difference is that instead of adding just a constant length, the slack time length is computed from the operation time, and the original time window length. Therefore, we need to optimize it to get an optimal slack time for each time windows. So that every time window has different length depending on the number of customers assigned to it.
With this optimization the time window which contains larger number of customers will have higher slack time length. And if the time window do not contain any customer it will have a slack time length of 0. In the former case a time window has a less possibility to accommodate a new customer into it. While in the latter case, the time windows can accommodate relatively high number

of new customers. In order to give an equal probability of accommodation of missed customers for all time windows the slack time has to be optimized. The optimization of the slack time is described in appendix 1.

The optimal slack time is less or equal to the maximum slack time. With the

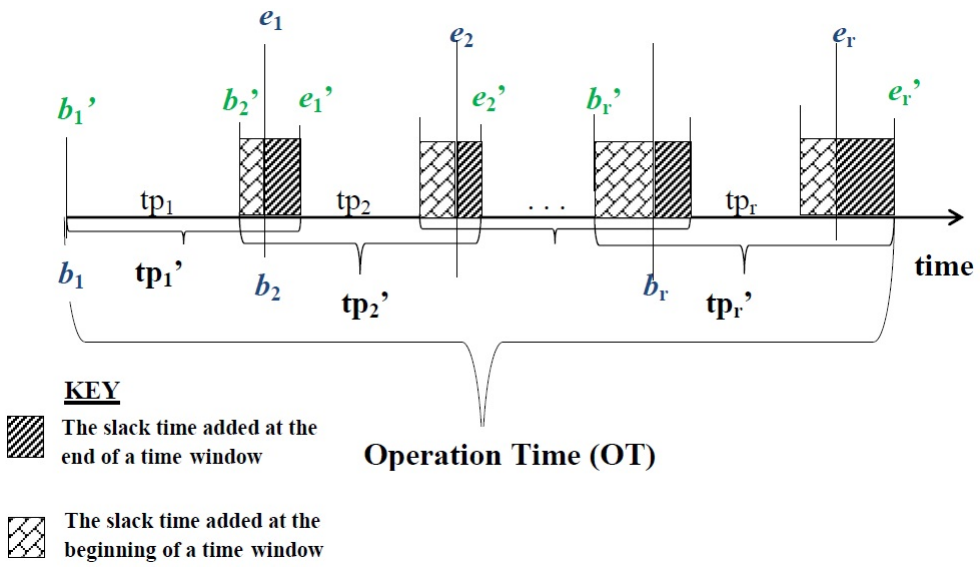

Figure 3.3: Extended time window with a variable length slack time

optimal slack time every time window within a route has different length. So a missed customer can be inserted into the appropriate time window in which the service time of the other customer is not going to be affected. Furthermore, the service of the customers in the other time windows do not be affected due the insertion of the missed customers.

3.4 Dynamic scheduling

In section 3.3 we described the two dynamism causes (*i.e.*, miss deliveries and customers location change) that we considered in our proposed algorithm. Moreover, we described that how a slack time is computed and added to the time window concept(see section 3.2) so as to elongate its length in order to accommodate the missed customers. In this section we describe how the dynamic scheduling can be done.

Once the slack time for each time window is computed and added to the respective time window, a missed customer can be inserted into the best possible feasible insertion point and get serviced. In order to find the feasible insertion point different algorithms can be employed. In our algorithm we applied the brute force search algorithm into the savings method. The approach we used works as follows.

First, it selects the time windows that are not reached yet starting from the current location of the vehicle. That is, the time windows that contain customers which are not serviced yet. By doing so the search space for the brute force algorithm is also minimized. After selecting the time windows it compute the savings of servicing the missed customer in between two consecutive customers[1] or next to the last customer in the search space.

The feasible insertion point is selected in an iterative process based on the savings. That is, the highest saving is considered first. If there is enough time to include the missed customer within the time window which contains the customers that gives the current saving under consideration, this time window is selected as the best feasible insertion point. Otherwise the iteration continues and check the next highest saving.

The complexity of computing the savings of inserting the missed customers is $\mathcal{O}(n^2 + m^2)$, where m is the number of missed customers and n is the number of customers in the route which are not serviced yet. The feasibility checking has a constant complexity. While finding the feasible route has a complexity of $\mathcal{O}(n)$. The whole complexity of the algorithm is $\mathcal{O}(n^3 + m^2 n)$. The following pseudo code describes how the algorithm works.

[1] consecutive customers, is mean that the customers which are found next to each other in the route.

Algorithm 3: Dynamic routeing of missed customers

input : A list of missed customers, current vehicle location, and the route r.

output: A new route

```
1  begin
       /* divide the route into visited and not visited
          to get the un-visited part of the route       */
2      route nR ← r.getUnVisited()
3      List newSavings ← ∅
4      List unRoutedCustomers ← ∅
5      foreach missed customer m do
6          boolean b ← false
7          foreach subRoute s in nR do
8              foreach customer c in s do
9                  newSavings.add(computeSaving(c, m, c.next()))
10         newSavings.sort(descending)
11         foreach s in newSavings do
12             if b = true then
13                 break
14             customer c ← s.getFirstCustomer()
               /* get the sub route where customer c found.
                  */
15             sr ← c.getSubRoute()
16             if m.serviceTime() + travelTime(c, m, c.next())+
               sr.totalTimeIn() ≤ sr.OptimalSlackTime() then
17                 insert m next to c
18                 b ← true
19                 nR.updateRoute()
```

Chapter 4

Experimental Results

In this chapter we describe the tests we performed and the results we obtained. This chapter is organized as follows. The first section describes the implementation and testing of the prototype. The KPI that was used to evaluate the performance of our algorithm and the developed prototype is also described in this section. The obtained results are given in the second section.

4.1 Implementation and testing

We implemented the prototype with Java technology. In addition with the implementation of the main algorithm the prototype has a simple graphical user interface (GUI) which is used to simulate the routes. Also the GUI is used for providing the inputs during the testing.
We tested the prototype on a machine which has a Windows 7 operating system, 4GB of RAM, and Intel duo core of CPU with speed of 3.07 GHz each.

4.1.1 Test data

To execute tests and measure the performance of the prototype and hence the our algorithm we used the Solomon's test instance [20]. The Solomon's instance has the format shown in table 4.1. From table 4.1 only the first four columns were necessary for the purpose of our prototype. The first column is the identifier of the customers, the second and the third columns are the X and Y coordinates

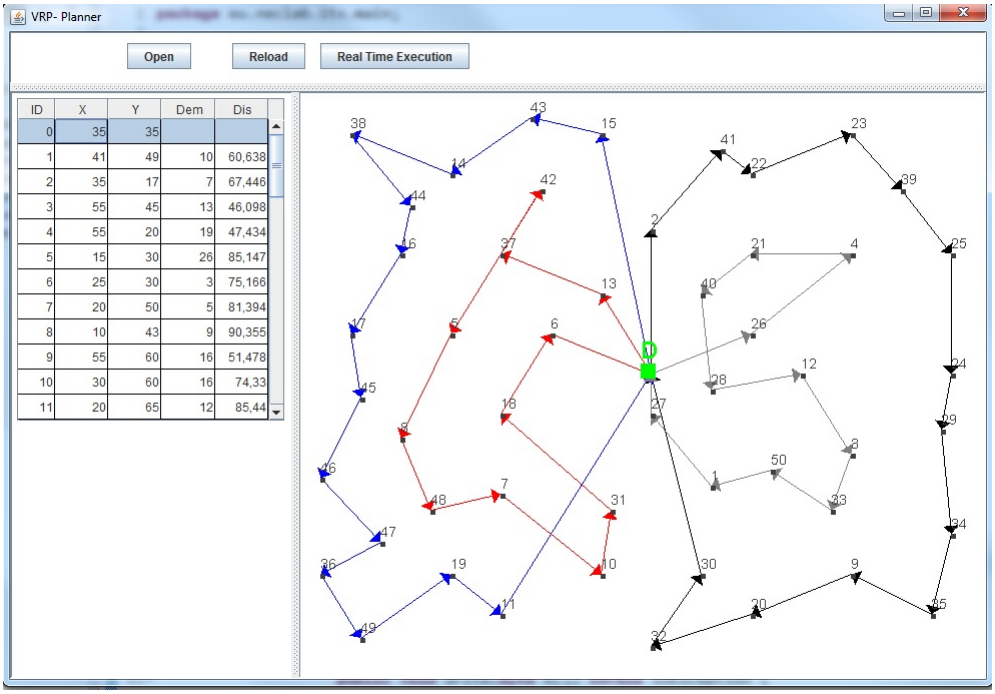

Figure 4.1: A simple GUI of the developed prototype

Table 4.1: Solomon's test instance format

ID	X	Y	DEMAND	READY TIME	DUE DATE	SERVICE TIME
0	35	35	0	0	230	0
1	47	87	35	151	181	10
2	31	25	10	60	70	10
...	...	...	...	...	...	...

of customers location respectively, and the fourth one is the demand of the customers. The euclidean distance (considered as the cost) between the customers was calculated from the X and Y coordinates of the customers location.

Solomon has generated his test instances in three different categories, *i.e.*, clustered, random and random clustered mixed. We used the random instance to test our prototype.

We took an assumption for the speed and the capacity of the vehicle. The average drive speed of vehicle in the town is from 30 - 40 kilometers per hour and we assumed 30 kilometers per hour for our test. Based on the demands given in the test instance we assumed the capacity of the vehicle to 200 units.

4.1.2 Testing

We provided the Solomon's test instance data to our prototype and first it planned the static route based on the savings method. This gave us the list of routes which in turn contain a list of customers and the cost incurred to service these customers. For example, figure 4.2 shows the final routes after the execution of the algorithm for 25 customers.

```
Number of Routes = 2
Route 1: 0 - 23 - 22 - 2 - 15 - 14 - 16 - 17 - 8 - 5 - 18 - 6 - 13 - 21 - 24 - 0 , Cost = 261.74
Route 2: 0 - 9 - 20 - 10 - 11 - 19 - 7 - 1 - 12 - 3 - 4 - 25 - 0 , Cost = 229.83
Total cost: 491.57
```

Figure 4.2: Illustrative example: final routes of 25 customers.

```
Route 1:
Max Slot Time: 1.6666666666666667
Slote inconsistency: 2.8
Time Window 1:
        Optimal slot time: 0.5952380952380952
        [9:00 - 11:06]: 23 - 22 - 2 - 15 - 14 -  (Time: 0.874530 )
Time Window 2:
        Optimal slot time: 0.5952380952380952
        [10:30 - 13:12]: 16 - 17 - 8 - 5 - 18 -  (Time: 0.856522 )
Time Window 3:
        Optimal slot time: 0.4761904761904763
        [12:43 - 15:05]: 6 - 13 - 21 - 24 -  (Time: 0.735295 )

Route 2:
Max Slot Time: 1.6666666666666667
Slote inconsistency: 2.2
Time Window 1:
        Optimal slot time: 0.7575757575757576
        [9:00 - 11:16]: 9 - 20 - 10 - 11 - 19 -  (Time: 0.811301 )
Time Window 2:
        Optimal slot time: 0.6060606060606061
        [10:39 - 13:17]: 7 - 1 - 12 - 3 -  (Time: 0.744068 )
Time Window 3:
        Optimal slot time: 0.30303030303030304
        [12:58 - 14:48]: 4 - 25 -  (Time: 0.190909 )
```

Figure 4.3: Illustrative example: time windows for the routes of 25 customers.

After the initial routes were planned a time window was assigned to each customer in the route. Based on the number of customers assigned to each time window,

the length of the operational time (which was assumed to be 8 hours per day), and the length of the time window (which was assumed 1 hour length) the slack time was calculated. Moreover, the time windows length was adjusted accordingly in order to accommodate the missed customers.
Figure 4.3 depicts the obtained time windows for routes of the 25 customers. In the figure "Time" is to refer the actual time in hours that was required to service all the customer within the time window. For example, 0.87 hour was required to service the customer in the first time window of route 1.

```
 Dynamic Scheduling
==================
Missed delivery is at customer: 10,
Current vehicle location is at customer: 1
-----------------------------------------------------
Intial route: 9 - 20 - 10 - 11 - 19 - 7 - 1 - 12 - 3 - 4 - 25 - 0
Initial route cost: 229.82794491567955
Static cost until vehicle location: 128.92542252176784

The new route after miss delivery is inserted:
1 - 10 - 12 - 3 - 4 - 25 - 0 , Cost: 131.83117579008677
Total cost after miss delivery is inserted: 260.75659831185465
```

Figure 4.4: Illustrative example: a new route after a miss delivery is rerouted and inserted into another time window.

Figure 4.4 shows the resulted route after a missed delivery is reinserted. Customer 10 from route 2 in figure 4.2 had been missed and it was detected when the vehicle had arrived at customer 1. Applying the algorithm inserting next to the vehicle location was profitable and the missed customer was inserted just next to the vehicle location. As a result of this reinsertion the total cost of the route was also increased from 229.83 to 260.76.

4.1.3 Evaluator

The following are some of the KPI that were used to measure the performance and the scalability of our proposed algorithm and the developed prototype.

- Cost:
  By measuring the total cost of the routes for a given set of customers and

the total cost after a missed customer is reinserted into the route we aimed to see the changes in the cost.

- Number of routes:
  By measuring the total number of routes for a given set of customers we need to see the change in the number of routes after the missed customers from another set customers are added. That is, if 'x' number of customers are missed from a set of 'y' number of customers, we wanted see the change in the number of routes after these 'x' customers are added to another 'z' set of customers.

- Execution time:
  Although it do not directly measure the performance of our proposed algorithm and we were not able to draw any conclusion by using it, we also wanted to know the execution time it took to route and to add the time window for a given set of customer.

4.2 Results

We performed a set of tests by varying the percentage (*i.e.*, to 10% and 20%) of the missed customers. We used 25, 50, 100, and 200 customers for testing our prototype. In this section we present the results we obtained for the evaluation criteria which are mentioned in section 4.1.3.
Figure 4.5 shows that the cost increased upto 20% of the initial cost when there was 10% of missed delivery and reroute them into the same route by using our algorithm. Figure 4.7 shows that there was upto 14% increase in the number of routes if those missed deliveries were added to and got the service with another set of customers. These two graphs shows that our algorithm performs better. Because the cumulative cost of servicing those missed customers by adding them to another set of customers was higher. That is, first the number of routes had been increased. Second, there was also a service cost in the new route. Third, the customers service time would definitely be affected. The customers might not got their service in the same date or even within a couple of dates (for example if the next day was either weekend or holiday).

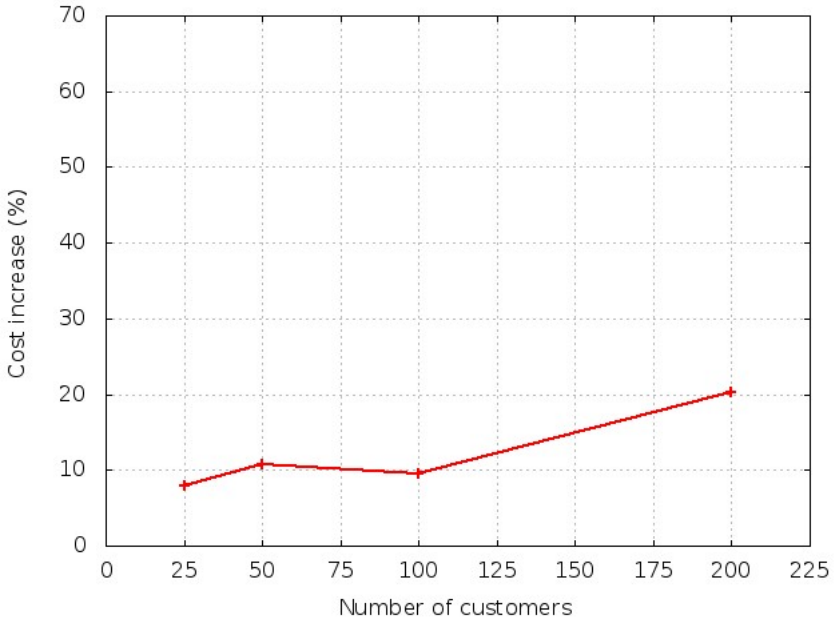

Figure 4.5: The cost increase after 10% missed deliveries are rerouted into the same route.

Figure 4.6 and 4.8 shows that the cost increased and the increase in the number of routes respectively for 20% missed deliveries. Servicing those missed customers in the same route added an extra cost of about 55% of the initial planned cost. Also, the increase in the number of routes was more than 20%. The total cost of servicing those missed customer with another set of customers was still high. However, we cannot predict what would happen in a real and could not be able to draw any conclusion from it.

We observed that the depot in the Solomon's test instance is located nearly in the center of the customers distribution. However in the real world scenarios this is not always true. For instance, there are cases where the depot is out of the city and the customers could be within the city. We changed the location of the depot to out of the center and we observed that the cost of servicing the missed customers within their route was relatively low that adding them to another set of customers. From these observation we came to conclude that the customers distribution and the depot location determines our proposed algorithm.

We have seen that our prototype can efficiently route upto 200 customers. It is

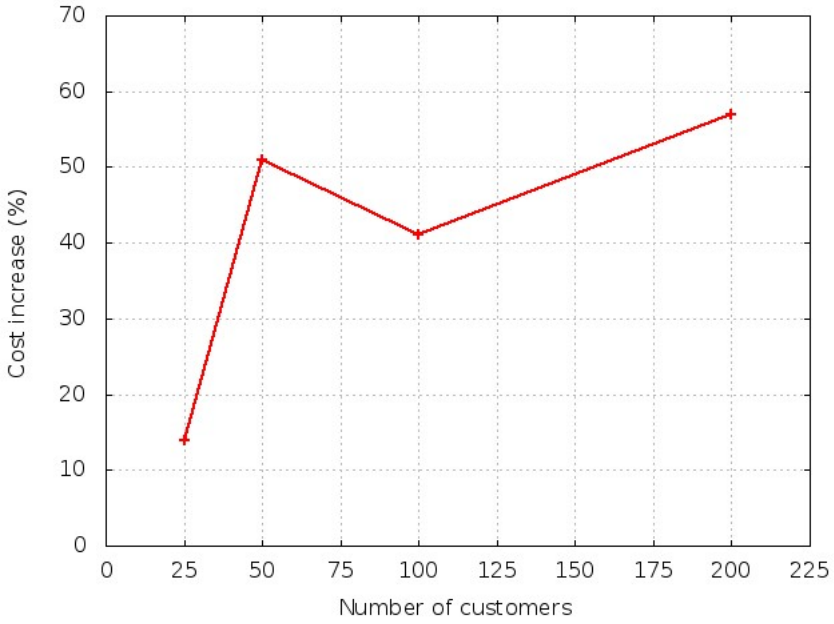

Figure 4.6: The cost increase after 20% missed deliveries are rerouted into the same route.

fascinating to test the prototype with large number of customers. This shows that our prototype is scalable to work with large instances of VRP. Furthermore, our proposed algorithm performed good for a missed deliveries upto 10%. Also the results we got for 20% of missed deliveries is not enough to conclude that the algorithm could not perform well for more than 10% of miss deliveries.

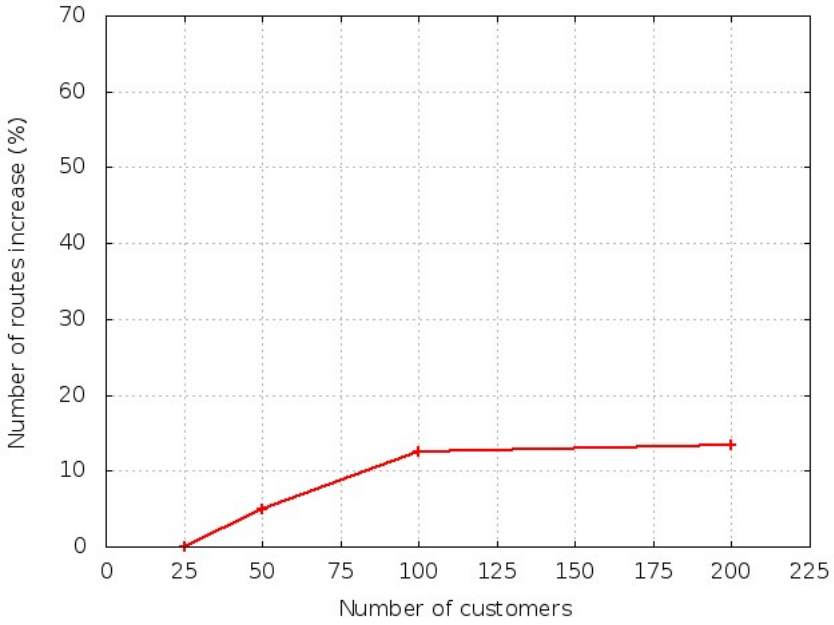

Figure 4.7: The increase in the number of routes after 10% missed delivery of customers were added to another set of customers.

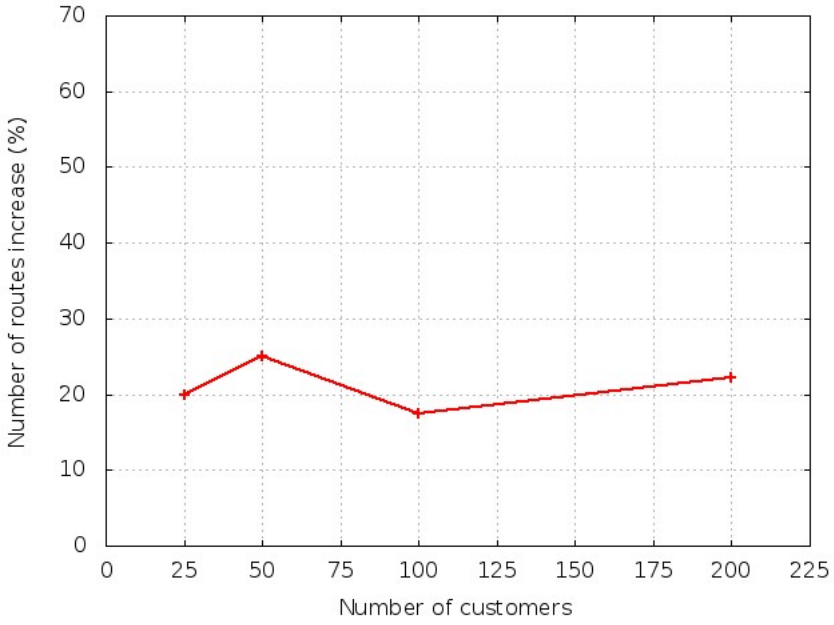

Figure 4.8: The increase in the number of routes after 20% missed delivery of customers were added to another set of customers.

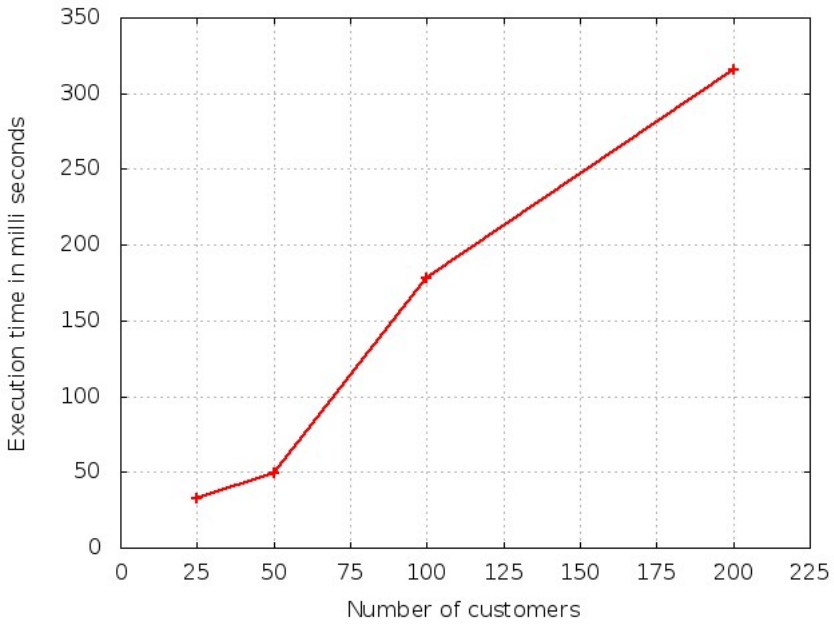

Figure 4.9: The execution time elapsed to route customers, to add the time windows and extend them by computing the optimal slack time.

Chapter 5

Conclusions

5.1 Summary

In this thesis we have introduced a concept of slack time and proposed an algorithm which solves a DVRP. Especially we considered two dynamism causes (*i.e.*, missed deliveries and customer location change) and solved them using our proposed algorithm. Customers assigned to a route may have a missed delivery or change their location after the initial plan has done and rescheduling is essentially required to service those customers.

The time windows in a route is elongated by adding an extra slack time and can accommodate missed customers. To give an equal probability of adding missed customers to the time window, the optimal slack time for each time window has to be computed. This optimal slack time is calculated by considering the number of customer initially assigned to the time window, the length of the operation time and the length of the time window.

The tests we performed in our prototype shows that our proposed algorithm is good enough to dynamically schedule a missed customers upto 10% of the total customers. The results we got by rerouteing the missed customers upto 20% of the total customer are not that bad. Instead it is interesting to make other tests with a real data and see the maximum amount of missed customers that the algorithm can efficiently reroute. Furthermore the test we performed showed that our prototype is scalable.

5.2 Future work

This work may continue in many directions. The following are some of our suggestion as a future research direction.

1. The slack time computation is just been developing. The slack time optimization can be further improved.

2. The time window adjustment has been done initially during the planning. It would be potentially interesting to adjust the length of the time window immediately when the route execution is going on. The slack times which are added to the time window can be collected and pushed forward if they are not used in the preceding time windows. So that this slack time will be used to elongate the subsequent time windows. This would give a high possibility of accommodation of missed customers.

3. It is fascinating to use other algorithm such as meta-heuristic or any intelligent algorithm to do the dynamic scheduling. This might reduce the complexity caused by the brute force searching algorithm during the dynamic routing.

4. We need to test the prototype with a real transportation data and perform more test with high number of customers.

5. We need also to apply this prototype in a mobile environments. This would give us the chance to track the exact location of the vehicle where the missed customer is detected and to do the rescheduling from this point, not only from the successor customer of the vehicle location.

6. Integrate the prototype with mobility simulation tools such as SUMO.

Appendix 1: Slack time optimization

The optimal slack time for a time window i, OST_i is calculated as follows.

$$OST_i = \alpha_i \times MaxST_i \tag{1}$$

Where α_i is a slot inconsistency factor for a time window i. α_i for a time window i is computed by considering the slot inconsistency(SI) of the time window and the route as a whole. The formulas to calculate the slot inconsistency factor α_i is given below.

A slot inconsistency for a time window i is calculated as follows:

$$SI_i = n \times \rho \tag{2}$$

where:

- n is the number of customers in a time window i, and
- ρ is the percentage of the maximum number of missed customer in a route that can be tolerated and its value is 0.2.

The total slot inconsistency for a route k is the sum of the slot inconsistencies of all the time windows within the route, *i.e.*,

$$SI_k = \sum_{i=1}^{t} SI_i \tag{3}$$

Where:

- SI_i is the contribution of a time window i for the total slot inconsistency of route k.
- t is the number of time windows a route k.

Finally from equation 2 and 3 we can compute the slot inconsistency factor α for time window i as follows:

$$\alpha_i = \frac{SI_i}{SI_k} \tag{4}$$

References

[1] G. Laporte and Y. Nobert, "Exact algorithms for the vehicle routing problem," *Annals of Discrete Mathematics*, vol. 31, pp. 147–184, 1987. 4, 8

[2] G. K. Rand, "The life and times fo the saving methods for vehicle routing problems," *ORiON: The Journal of ORSSA*, vol. 25, no. 2, pp. 125–145, October 2009. 5, 6, 7, 9, 19

[3] V. Pillac, "Dynamic vehicle routing: solution methods and computational tools," Ph.D. dissertation, Université Nantes Angers Le Mans, 2012. 5

[4] G. Laporte, "The vehicle routing problem: An overview of exact and approximate algorithms," *European Journal of Operational Research*, vol. 59, no. 3, pp. 345–358, June 1992. [Online]. Available: http://ideas.repec.org/a/eee/ejores/v59y1992i3p345-358.html 6, 8

[5] R. K. Tantikorn Pichpibul, ""a modified savings heuristic for the close-open mixed vehicle routing problem"," in *"Proceedings of the Asia Pacific Industrial Engineering & Management Systems Conference "*. "V. Kachitvichyanukul, H.T. Luong, and R. Pitakaso Eds", "2012". 8

[6] G. Clarke and J. Wright, ""scheduling of vehicles from a central depot to a number of delivery points"," *"Operations Research "*, vol. 12, pp. 568–581, 1964. 9

[7] "Cluster-first route-second method," http://neo.lcc.uma.es/vrp/solution-methods/heuristics/cluster-first-route-second-method/, accessed: 2013-08-28. 9

[8] "Route-first cluster-second method," http://neo.lcc.uma.es/vrp/solution-methods/heuristics/route-first-cluster-second-method/, accessed: 2013-08-28. 9

[9] A. E. Rizzoli, F. Oliverio, R. Montemanni, and L. M. Gambardella, "Ant colony optimisation for vehicle routing problems: from theory to applications," Tech. Rep., 2004. 10

[10] J. E. Bell and P. R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Advanced Engineering Informatics*, vol. 18, pp. 41–48, 2004. 10

[11] Y. Zhang, J. Liu, F. Duan, and J. Ren, "Genetic algorithm in vehicle routing problem," Institute of Computer and Software, Taiyuan University of Technology, Taiyuan, Shanxi, China, Tech. Rep. 10

[12] B. M. Baker and M. Ayechew, "Agenetic algorithm for the vehicle routing problem," *Computers & Operations Research*, vol. 30, pp. 787–800, 2003. 10

[13] J.-F. Cordeau and G. Laporte, "Tabu search heuristics for the vehicle routing problem," 2002. 11

[14] V. Pillac, M. Gendreau, C. Guret, and A. L. Medaglia, "A review of dynamic vehicle routing problems," *European Journal of Operational Research*, vol. 225, no. 1, pp. 1 – 11, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0377221712006388 12, 13

[15] A. Larsen, "The dynamic vehicle routing problem," Ph.D. dissertation, Department of Mathematical Modelling, The Technical University of Denmark, Building 321, DTU, DK-2800 Kgs. Lyngby, 2000. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?143 12, 13

[16] R. Bent and P. V. Hentenryck, "Dynamic vehicle routing with stochastic requests," http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.61.3096&rep=rep1&type=pdf, 2003. 13

[17] V. Zeimpekis, G. Giaglis, and I. Minis, "A dynamic real-timefleet management system for incident handling in city logistics," in *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, vol. 5, 2005, pp. 2900–2904 Vol. 5. 13

[18] B. K.-S. Cheung, K. Choy, C.-L. Li, W. Shi, and J. Tang, "Dynamic routing model and solution methods for fleet management with mobile technologies," *International Journal of Production Economics*, vol. 113, no. 2, pp. 694 – 705, 2008, ¡ce:title¿Special Section on Advanced Modeling and Innovative Design of Supply Chain¡/ce:title¿. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925527308000418 14

[19] R. Montemanni, L. Gambardella, A. E. Rizzoli, and A. V. Donati, "A new algorithm for a dynamic vehicle routing problem based on ant colony system," in *In Second International Workshop on Freight Transportation and Logistics*, 2003, pp. 27–30. 14

[20] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Oper. Res.*, vol. 35, no. 2, pp. 254–265, Apr. 1987. [Online]. Available: http://dx.doi.org/10.1287/opre.35.2.254 21, 31