AN ADAPTIVE LARGE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING THE ELECTRIC VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

by

MERVE KESKİN

Submitted to the Graduate School of

Engineering and Natural Sciences

in partial fulfillment of the requirements for the degree of

Master of Science

Sabancı University,

August, 2014

AN ADAPTIVE LARGE NEIGHBORHOOD SEARCH APPROACH
FOR SOLVING THE ELECTRIC VEHICLE ROUTING PROBLEM
WITH TIME WINDOWS

APPROVED BY:

Assoc. Prof. Dr. Bülent Çatay .....
(Thesis Supervisor)

Assoc. Prof. Dr. Güvenç Şahin .....

Asst. Prof. Dr. Tevhide Altekin .....

DATE OF APPROVAL: .....

ACKNOWLEDGEMENTS

Firstly, I would thank to my family for their endless trust, love and patience. They support and guide me without considering my instability for years.

I thank to Berk Özel for understanding and encouraging me at all time. Besides, he is the person who helped me improve my coding skills and made programming enjoyable to me.

I also express my special thanks to Bülent Çatay for his precious guidance not only to conduct my research and write my thesis but also to select my career path. I hope we would study for years.

I thank to all my friends for always assisting me.

I am also grateful to my professors for teaching me a lot of new aspects.

Finally, I would like to thank to TÜBİTAK for their financial support during my graduate education.

AN ADAPTIVE LARGE NEIGHBORHOOD SEARCH APPROACH FOR SOLVING THE ELECTRIC VEHICLE ROUTING PROBLEM WITH TIME WINDOWS

Merve Keskin

Industrial Engineering, Master's Thesis, 2014

Thesis Supervisor: Assoc. Prof. Dr. Bülent Çatay

Keywords: Electric vehicle, large neighborhood search, metaheuristics, recharging, vehicle routing

Abstract

The Electric Vehicle Routing Problem with Time Windows (E-VRPTW) is an extension to the well-known Vehicle Routing Problem with Time Windows (VRPTW). Different from VRPTW, the fleet in E-VRPTW consists of electric vehicles (EVs) which have a limited driving range due to their battery charge capacities. Since the battery charge level decreases proportional to the distance traveled, an EV may need to visit recharging stations to have its battery recharged in order to be able to continue servicing the customers along its route. The recharging may take place at any battery level and after the recharging the battery is assumed to be full. Recharging time is proportional to the amount charged. The number of stations is usually small and the stations are dispersed in distant locations, which increases the difficulty of the problem. In this thesis, we propose an Adaptive Large Neighborhood Search (ALNS) method to solve this problem**.** ALNS is based on the destroy-and-repair framework where at any iteration the existing feasible solution is destroyed by removing some customers and recharging stations from their routes and then repaired by inserting the removed customers to the solution along with the stations when recharging is necessary. Several removal and insertion algorithms are applied by selecting them dynamically and adaptively based on their past performances**.** The new solution is accepted according to the Simulated Annealing criterion. Our approach combines the removal and insertion mechanisms from the literature with some new mechanisms designed specifically for E-VRPTW. To test the performance of the proposed ALNS we use the instances and benchmark results presented in by Schneider et al (2014). Our computational results show that the proposed method is effective in finding good solutions in reasonable amount of time.

ZAMAN PENCERELİ ELEKTRİKLİ ARAÇ ROTALAMASI PROBLEMİ İÇİN BİR UYARLANABİLİR GENİŞ KOMŞULUK ARAMA YÖNTEMİ

Merve Keskin

Endüstri Mühendisliği, Yüksek Lisans Tezi, 2014

Tez Danışmanı: Doç. Dr. Bülent Çatay

Anahtar Kelimeler: Elektrikli araç, geniş komşuluk arama, metasezgisel, şarj, araç rotalama

Özet

Zaman Pencereli Elektrikli Araç Rotalama Problemi (E-ZARP), çokça bilinen Zaman Pencereli Araç Rotalama Problemi (ZARP)'nin genişletilmiş bir biçimidir. ZARP'den farklı olarak, E-ZARP'de filo, batarya şarj kapasitesinden dolayı sınırlı sürüş menziline sahip elektrikli araçlardan (EA) oluşmaktadır. Batarya şarj seviyesi, alınan yol ile orantılı bir şekilde azaldığından dolayı EA, rotasındaki müşterilere hizmet vermeyi sürdürebilmek için, bataryasını şarj etmek amacıyla rotasının herhangi bir yerinde, şarj istasyonuna uğramak durumunda kalabilir. Şarj işlemi herhangi şarj seviyesinde olabilmekte ve şarj işleminden sonra bataryanın tam şarj olduğu kabul edilmektedir. Şarj süresi, şarj edilen miktar ile doğru orantılıdır. İstasyon sayısı genellikle az olup istasyonlar uzak noktalarda konumlanmışlardır. Bu da problemin zorluk derecesini arttırmaktadır. Bu tezde, belirtilen problemi çözmek için bir Uyarlanabilir Geniş Komşuluk Arama Yöntemi (UGKA) önerilmiştir. UGKA yöntemi, boz-onar sistemine dayanmaktadır. Olurlu çözüm, bazı müşteri ve istasyonların rotalarından çıkarılmaları ile bozulmakta, çıkarılan müşterilerin, şarj işlemi de gerekli ise istasyonlar ile beraber çözüme tekrar eklenmeleri ile onarılmaktadır. Birçok çıkarma ve ekleme algoritması kullanılmış ve bu algoritmalar yöntem içinde, geçmiş performansları baz alınarak dinamik ve uyarlanabilir bir şekilde seçilmiştir. Elde edilen yeni çözüm Benzetilmiş Tavlama kriterine gore kabul edilmiştir. Bizim yaklaşımımız, literatürde var olan çıkarma ve ekleme algoritmaları ile E-ZARP için özel olarak tasarlanmış yeni mekanizmaları birleştirmektedir. Önerilen UGKA'nın performansını test etmek için, Schneider et al. (2014)'de sunulan örnekler ve sonuçlar kullanılmıştır. Sonuçlarımız, önerilen yöntemin, makul süreler içinde iyi sonuçlar bulmada etkili olduğunu göstermiştir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

Chapter 1

Introduction

Transportation systems account for about 20-25% of global energy consumption and CO_2 emissions. Road transport is a major contributor with 75% share. 95% of the world's transportation energy comes from fossil fuels, mainly gasoline and diesel (www.epa.gov). Transport accounts for 63% of fuel consumption and 29% of all CO_2 emissions in the EU. 45% of the goods are moved by trucks and road transport is predicted to grow by 33% in 2030 (http://ec.europa.eu). In the US, about 28% of total greenhouse gas (GHG) emissions are transport related. (www.epa.gov). 75% of the domestic freight is moved by trucks and the freight volume is expected to grow by 39% in 2040 (www.bts.gov).

Transportation will continue to be a major and still growing source of GHGs. Hence, governments are considering new environmental measures and targets for reducing emissions and fuel resource consumptions. The US Administration aims at cutting the overall GHG emissions 17% below 2005 levels by 2020 and has recently established the toughest fuel economy standards for vehicles (http://www.whitehouse.gov). The EU targets 80–95% reduction of GHGs below 1990 levels by 2050, where a reduction of at least 60% is expected from the transport sector. The European Commission aims at reducing the transport-related GHG emissions to around 20% below their 2008 level by 2030. The use of conventionally fuelled cars will be reduced by 50% in urban transport by 2030 and phased out by 2050. City logistics in major European urban centers will be CO_2-free by 2030 (White Paper on Transport, 2011).

The targets set by governments and the new regulations imposed encourage the usage of alternative fuel vehicles (AFV) such as solar, electric, biodiesel, LNG, CNG vehicles.

Many municipalities, government agencies, non-profit organizations and private companies are converting their fleets to include AFVs, either to reduce their environmental impact voluntarily or to meet new environmental regulations (Erdoğan and Miller-Hooks, 2012).

In a world where environmental protection and energy conservation are growing concerns, the development of electric vehicle (EV) technology has taken on an accelerated pace to fulfill those needs. Concerning the environment, EVs can provide emission-free urban transportation. Even taking into account the emissions from the power plants needed to fuel the vehicles, the use of EVs can still significantly reduce global air pollution.

EV is a vehicle which moves with electric propulsion. EVs may be classified as battery electric vehicles (BEV), hybrid electric vehicles (HEV), and fuel-cell electric vehicles (FCEV) (Chan, 2002). They include electric trains, airplanes, boats, motorcycles, scooters and spacecrafts. In the thesis, we refer to EV as a road vehicle such as a truck or van. A fleet of EVs can be used in a variety of transport needs such as public transportation, home deliveries from grocery stores, postal deliveries and courier services, distribution operations in different sectors.

Although EVs enable emission-free logistics services, operating an EV fleet has several drawbacks: (i) low energy density of batteries compared to the fuel of combustion engined vehicles; (ii) EV often have long recharge times compared to the relatively fast process of refueling a tank; and (iii) the scarcity of public charging stations (Touati-Moungla and Jost, 2011). Under these limitations, routing an EV fleet arises as a challenging combinatorial optimization problem among the Vehicle Routing Problems (VRPs).

In this thesis, we address the Electric Vehicle Routing Problem with Time Windows (E-VRPTW). The problem was introduced by Schneider et al. (2014) as an extension to the Green Vehicle Routing Problem (G-VRP) of Erdoğan and Miller-Hooks (2012). G-VRP concerns “green” vehicles which run with biodiesel, liquid natural gas, or CNG and have a limited driving range. Hence, the vehicles may need refueling along their route. Refueling is fast; however, the stations for these fuels are scarce. E-VRPTW is a variant of the classical VRPTW where the vehicles may need to visit stations to have their batteries recharged in order to continue their route, as in G-VRP. Recharging operation

may take a significant amount of time, especially when compared to relatively short fueling times of gasoline. Furthermore, unlike gasoline stations recharging stations are dispersed at distant locations, which significantly affects the route planning.

To solve this challenging problem, we propose an Adaptive Large Neighborhood Search (ALNS) approach. Our approach combines the ALNS schemes presented in Ropke and Pisinger (2006a, 2006b), Pisinger and Ropke (2007) and Demir et al. (2012) with new algorithms specific to E-VRPTW. We address the distance minimization objective as well as the hierarchical objective approach where minimizing the number of vehicles (routes) is the primary objective and minimizing total travel distance is the secondary. Our results show that the ALNS algorithm is effective in finding good quality solutions and improves some of the best-known solutions in the literature.

The remainder of the thesis is organized as follows: Chapter 2 reviews the related literature. Chapter 3 describes the problem and gives the mathematical model. The proposed ALNS is presented in Chapter 4 and the computational study is provided in Chapter 5. Finally, Chapter 6 concludes the thesis with some remarks and directions for future research.

Chapter 2

Literature Review

There are relatively few publications on optimization problems related to alternative fuels. Some works concentrate on finding the energy shortest path from a given origin to a destination. Artmeirer et al. (2010) studied this problem within a graph-theoretic context and proposes extensions to general shortest path algorithms that address the problem of energy-optimal routing. They formalize energy-efficient routing in the presence of rechargeable batteries as a special case of the constrained shortest path problem and present an adaption of a general shortest path algorithm that respects the given constraints. Wang and Shen (2007) developed a model that minimizes the number of tours and total deadhead time hierarchically. There is a constraint which limits the travel time of every vehicle after being recharged. The recharging durations, time windows and vehicle capacities are not considered. A multiple ant colony algorithm was developed to solve the problem.

Wang and Cheu (2012) investigated the operations of an electric taxi fleet. Their model minimizes total distance travelled under the recharging constraints and maximum route time. Charge of the battery is consumed with a given rate per traveled distance and can be replenished at the recharging stations. Recharging times are assumed to be fixed and after charging the battery becomes full. They construct an initial solution using one of the nearest-neighbor, sweep and earliest time window insertion heuristics and improve it using Tabu Search (TS). They also suggested three different charging plans which provide different driving ranges and compare the results against the full charging scheme.

Omidvar and R. Tavakkoli-Moghaddam (2012) addressed an AFV routing problem with time-windows and proposed a mathematical model that minimizes total costs associated with the vehicles, distance travelled, travel time and emissions. The refueling times are constant and the depot is considered as an alternative fuel station. They developed Simulated Annealing (SA) and Genetic Algorithm (GA) approaches and compared their performances.

Conrad and Figliozzi (2011) introduced the Recharging Vehicle Routing Problem (RVRP), a new variant of the VRP where the EVs are allowed to recharge at the customer locations they visit. The model has dual objectives: the primary objective minimizes the number of routes or vehicles whereas the secondary objective minimizes the total costs associated with the travel distance, service time and vehicle recharging which is a penalty cost if recharging is performed. Charging is done while servicing the customer and charging time is taken as a parameter which is a constant value. The battery level departing from a customer depends on the choice of normal charging or fast charging. In the fast charging case the battery is charged to a specified level such as 80% of battery capacity.

Worley et al. (2012) addressed the problem of locating charging stations and designing EV routes simultaneously. The objective is to minimize the sum of the travel costs, recharging costs, and costs of locating recharging stations. A solution method is not proposed and left as future work.

Erdoğan and Miller-Hooks (2012) considered the routing of AFVs within the context G-VRP and formulated the mathematical model. The model aims at minimizing the total distance travelled where the length of the routes is restricted. Fuel is consumed with a given rate per traveled distance and can be replenished at the alternative fuel stations (AFVs). Refueling times are assumed to be fixed and after refueling the tank becomes full. The model does not involve time windows and vehicle capacity constraints. Erdoğan and Miller-Hooks (2012) proposed two heuristics to solve the G-VRP. The first is a Modified Clarke and Wright Savings (MCWS) algorithm which creates routes by establishing feasibility through the insertion of AFSs, merging feasible routes according to savings values, and removing redundant AFSs. The second is a Density-Based Clustering Algorithm (DBCA) based on the cluster-first and route-second approach. DBCA forms clusters of customers such that every vertex within a given

radius contains at least a predefined number of neighbors. Subsequently, the MCWS algorithm is applied on the identified clusters. To test the performance of these two heuristics, they designed two sets of problem instances. The first consists of 40 small-sized instances with 20 customers while the second involves 12 instances with up to 500 customers.

Schneider et al. (2014) introduced the E-VRPTW where the customers are associated with time windows and the vehicles have capacity and driving range constraints. The recharging duration is proportional to the battery usage when arriving at the station and the battery is fully recharged when departing from the station. To solve this problem, Schneider et al. developed a hybrid metaheuristic that combines the Variable Neighborhood Search (VNS) algorithm with TS. They tested the performance of the proposed method on newly designed benchmark instances for E-VRPTW as well as on test instances of related problems, namely the Green VRP (G-VRP) and the Multi-Depot VRP with Inter-Depot Routes (MDVRPI).

ALNS was introduced by Ropke and Pisinger (2006a) as an extension of the Large Neighborhood Search (LNS) framework put forward by Shaw (1998). Ropke and Pisinger (2006b) developed a unified ALNS heuristic for a large class of VRP with Backhauls. Pisinger and Ropke (2007) improved this heuristic with additional algorithms and showed that the proposed framework gives competitive results in different VRP variants. Different implementations of ALNS include the resource-constrained project scheduling problem (Muller, 2009), scheduling of technicians and tasks in a large telecommunication company (Cordeau et al., 2010), lot-sizing problem with setup times (Muller et al., 2010), and consultation timetabling problem at Danish high schools (Kristiansen et al., 2013). Within the VRP framework, ALNS is used for solving, the Pick-up and Delivery Problem (PDP) where requests can be transferred between vehicles during their trip (Masson et al., 2012), Capacitated Vehicle Routing Problem (CVRP) which minimizes the sum of arrival times at customers (Ribeiro and Laporte, 2012) and the Pollution-Routing Problem (PRP) (Demir et al., 2012).

Chapter 3

Problem Description and Formulation

In this chapter, we first describe the E-VRPTW and then provide its 0-1 mixed-integer linear programming model.

3.1. Problem Description

Similar to the classical VRPTW, E-VRPTW concerns a set of customers with known demands, delivery time windows and service durations. It constructs routes that begin with the depot and end at the depot. Different from VRPTW, the deliveries are performed by a homogeneous fleet consisting of EVs with fixed loading capacities and limited cruising ranges. While the vehicle is traveling, the battery charge level decreases proportionally with the distance traversed. So, the vehicle may need to visit a recharging station and have its battery recharged in order to be able to continue servicing customers along its route. The number of stations is usually small and the stations are dispersed in distant locations, which complicates the problem. There is one depot and it can also be used as a recharging station. Recharging may take place at any battery level and after the recharging the battery is assumed to be full. The recharge duration is proportional to the recharge quantity. Each vehicle departs from the depot with a full battery and returns to the depot at the end of its route before the due date.

Figure 3.1 illustrates a sample problem involving 10 customers and 4 stations and shows the optimal solution. In this figure, *D* refers to the depot. The customer set is $C = \{C1, C2, C3, C4, C5, C6, C7, C8, C9, C10\}$ and the station set is $S = \{S1, S2, S3, S4\}$. S1 is the depot.

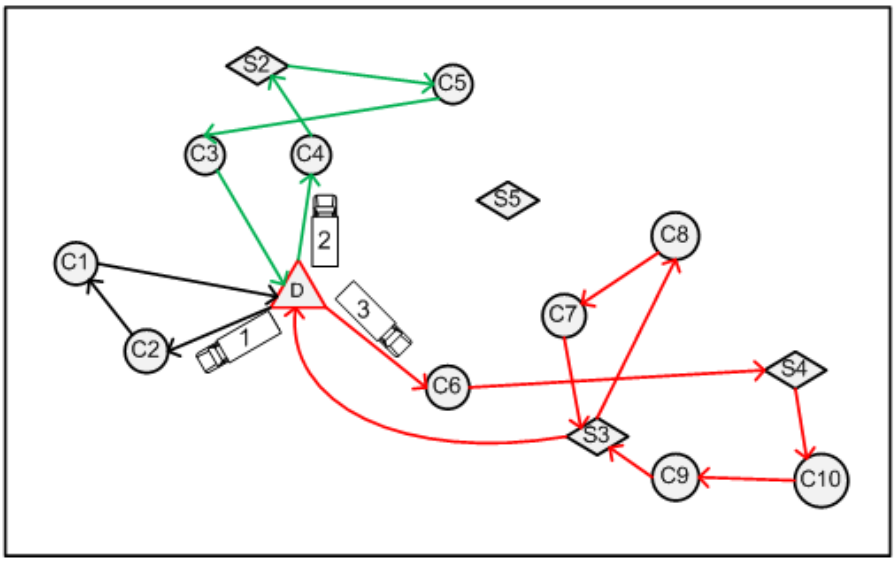

Figure 3.1: A sample network of 10 customers and the routes at the optimal solution

There are three routes in the optimal solution. EV1 services *C1* and *C2*, returns to the depot with its initial charge. EV2 visits *S2* after servicing *C4* and has its battery recharged before visiting *C5* and *C3*. On the other hand, EV3 is recharged once in *S4* and twice in *S3*, first after servicing *C9* and next after servicing *C7* on its way back to *D*. As it can be seen from this example, a station (*S3*) can be visited multiple times by the same or different vehicles and a station is not necessarily visited (*S5*).

3.2. 0-1 Mixed Integer Linear Programming Formulation

In this section, we provide the mathematical model of E-VRPTW formulated in Schneider et al. (2014). Let $V = \{1, \dots, N\}$ denote the set of customers and F denote the set of recharging stations. Since a recharging station may be visited more than once depending on the route structure, we must create F'which is the set of dummy vertices generated to permit several visits to each vertex in the set F. Vertices 0 and $N+1$ denote the depot and every route starts at 0 and ends at $N+1$. Let V' be a set of vertices with $V' = V \cup F'$. In order to indicate that a set contains the respective instance of the depot, the set is subscripted with 0 or $N+1$. Hence $V_0' = V' \cup \{0\}$ and $V_{N+1}' = V' \cup \{N+1\}$. Now we can define the problem on a complete directed graph $G = (V_{0,N+1}', A)$ with the set of arcs $A = \{(i,j) |\ i,j \in V_{0,N+1}',\ i \neq j\}$ where $V_{0,N+1}' = \{0\} \cup V_{N+1}'$. Each arc is associated with a distance d_{ij} and travel time t_{ij}. The battery charge is consumed at a rate of h and every traveled arc consumes $h \times d_{ij}$ of the remaining battery. Each vertex $i \in V'$ has positive demand q_i, service time s_i and time window $[e_i, l_i]$. All EVs

have a load capacity of C and battery capacity of Q. At a recharging station, the battery is charged at a recharging rate of g. The decision variables, τ_i, u_i and y_i keep track of the arrival time, remaining cargo level and remaining charge level at vertex $i \in V'_{0,N+1}$, respectively. The binary decision variable x_{ij} takes value 1 if arc (i,j) is traversed and 0 otherwise.

$$\min \sum_{\substack{i \in V'_0 \\ j \in V'_{N+1}, i \neq j}} d_{ij} x_{ij} \quad (1)$$

$$\sum_{j \in V'_{N+1}, i \neq j} x_{ij} = 1 \qquad \forall i \in V \quad (2)$$

$$\sum_{j \in V'_{N+1}, i \neq j} x_{ij} \leq 1 \qquad \forall i \in F' \quad (3)$$

$$\sum_{i \in V'_0, i \neq j} x_{ij} = \sum_{i \in V'_{N+1}, i \neq j} x_{ji} \qquad \forall j \in V' \quad (4)$$

$$\tau_i + x_{ij}\left(t_{ij} + s_i\right) - l_0\left(1 - x_{ij}\right) \leq \tau_j \qquad \forall i \in V_0, \forall j \in V'_{N+1},\ i \neq j \quad (5)$$

$$\tau_i + x_{ij}\, t_{ij} + g(Q - y_i) - (l_0 + gQ)\left(1 - x_{ij}\right) \leq \tau_j \quad \forall i \in F', \forall j \in V'_{N+1},\ i \neq j \quad (6)$$

$$e_j \leq \tau_j \leq l_j \qquad \forall j \in V'_{0,N+1} \quad (7)$$

$$0 \leq u_j \leq u_i - x_{ij}\, q_i + C\left(1 - x_{ij}\right) \qquad \forall i \in V'_0, \forall j \in V'_{N+1},\ i \neq j \quad (8)$$

$$0 \leq u_0 \leq C \quad (9)$$

$$0 \leq y_j \leq y_i - x_{ij}\left(d_{ij}\, h\right) + Q\left(1 - x_{ij}\right) \quad \forall i \in V, \forall j \in V'_{N+1},\ i \neq j \quad (10)$$

$$0 \leq y_j \leq Q - (d_{ij}\, h) x_{ij} \quad \forall i \in F'_0, \forall j \in V'_{N+1},\ i \neq j \quad (11)$$

$$x_{ij} \in \{0,1\} \quad \forall i \in V'_0, \forall j \in V'_{N+1},\ i \neq j \quad (12)$$

The objective function (1) minimizes total travelled distance. Constraints (2) and (3) handle the connectivity of customers and visits to recharging stations, respectively. The flow conservations constraints (4) enforce that the number of outgoing arcs equals to the number of incoming arcs at each vertex. Constraints (5) and (6) ensure the time feasibility of arcs leaving the customers (and the depot) and the stations, respectively. Constraints (7) enforce the time windows of the customers and the depot. In addition, constraints (5)-(7) eliminate the sub-tours by maintaining the schedule feasibility with respect to time considerations. Constraints (8) and (9) guarantee that demand of all

customers are satisfied and constraints (10) and (11) make sure that the battery level is never negative. Finally, (12) define the binary decision variables.

If the objective function is to minimize the number of vehicles, it is formulated as follows:

$$min \sum_{j \in V'_{n+1}} x_{0j} \qquad (1')$$

Chapter 4

Solution Methodology

In this chapter, we present the details of ALNS proposed for solving E-VRPTW.

4.1. Proposed Adaptive Large Neighborhood Search Approach

The ALNS approach proposed in this study includes the following five types of algorithms:

- $A_{CR}: Customer\ Removal\ Algorithms$
- $A_{CI}: Customer\ Insertion\ Algorithms$
- $A_{SR}: Station\ Removal\ Algorithms$
- $A_{SI}: Station\ Insertion\ Algorithms$
- $A_U: Update\ Algorithms$

It combines the strengths of the ALNS heuristics introduced by Ropke and Pisinger (2006a, 2006b), Pisinger and Ropke (2007), and Demir et al. (2012) by introducing new removal, insertion and station removal, insertion algorithms specific to this problem. The main components of the heuristic can be stated as follows:

- **General Flow:** The algorithm begins with an initial solution and iteratively improves it by removal and insertion mechanisms. The current feasible solution is destroyed by removing some customers and stations from their routes and then repaired by inserting the removed customers and necessary stations to the solution in an intelligent way. Let S_C be the current feasible solution at the beginning of a new iteration. At each iteration a customer removal algorithm cr_a and an update algorithm u_a is selected. Then the customers identified by the customer removal

algorithm are removed from S_C in the update phase. Depending on u_a some stations may also be removed from S_C. At the end of the update phase, we have another feasible solution S_1. Then a customer insertion algorithm ci_a is selected and each removed customer is inserted into S_1 according to the insertion algorithm. Let the feasible solution after the customer insertion be called S_2. If there have been N_{SR} iterations since last station removal-insertion procedure is applied, then it should be applied again. A station removal algorithm sr_a is selected and the identified stations are removed from S_2 and we obtain a solution called S_3. S_3 may be infeasible in terms of charge. If this is the case, then a station insertion algorithm si_a is selected and S_3 is made feasible by inserting stations according to the selected algorithm. Let S_4 denote this feasible solution. If there have been N_{RR} iterations since the last execution of the route minimization procedure, then it should be applied again. In this procedure, only route removal algorithms are used in the customer removal phase. Then the customers identified by the customer removal algorithm are removed from S_4. Let us denote this partial feasible solution by S_5. Since all the customers of a route or routes are removed from the solution we do not need an update phase which updates the features of the remaining customers and stations in the routes that are changed after the removal process. Then a customer insertion algorithm ci_a is selected and each removed customer is inserted into S_5 according to the insertion algorithm. Let the feasible solution after the customer insertion be called S_6. If there has been N_{SR} iterations since last station removal-insertion procedure is applied, then it should be applied again to the solution S_6. This route minimization procedure is applied in a loop which lasts for τ iterations. Then the algorithm continues with regular customer removal and insertion algorithms. The whole procedure is repeated until the maximum number of iterations is reached.

If the objective is to minimize the total distance, the general framework remains same; however, we do not apply the route minimization procedure and slightly modify some of the insertion algorithms and new solution acceptance criteria.

- **Adaptive Scoring:** Each algorithm has a score which measures how well the algorithm has performed recently. High scores correspond to a successful heuristic. We let the entire search last *N* iterations. Then we divide the entire search into a number of *segments* which is a part of the search consisting of a number of

iterations. Let π_a be the score of algorithm a. π_a value of all algorithms is set to zero at the beginning of each segment. If a new best solution is found in an iteration of a segment, then π_a values of corresponding algorithms are increased by σ_1. If customer removal, update and insertion were carried out in that iteration, then π_a values of cr_a , u_a , ci_a algorithms are increased. Since we do not know which algorithm has yielded the improvement, we increase the score of all algorithms used at that iteration. If station removal and insertion were carried out in that iteration, then the same procedure is applied to sr_a, si_a algorithms used. Similarly, if route minimization is operated in that iteration, this procedure is applied to rr_a and ci_a algorithms used. If a better solution is found in an iteration of a segment, then π_a values of corresponding algorithms are increased by σ_2 similar as above. Nevertheless, if a worse solution is found in an iteration of a segment and it is accepted by SA mechanism, then π_a values are increased by σ_3. If a worse solution is found but not accepted, then only the number of selections of algorithms used in that iteration are increased by 1. Their scores stay the same since they do not contribute to an improvement. The same scoring is applied if a solution with the same objective function is found.

- **Adaptive Weight Adjustment:** At the end of each segment, new weights of algorithms are calculated using total score during the last segment. Let $w_{a,j}$ and $\theta_{a,j}$ represent the adaptive weight of the algorithm and the number of times the algorithm has been selected during segment $j = 1,2 \dots \Delta$ respectively. Initially all weights are equal to 1, i.e. $w_{a,1} = 1 \ \forall a \in A$. At the end of segment j, scores are updated as in (13):

$$w_{a,j+1} = w_{a,j}(1 - r) + r\frac{\pi_a}{\theta_{a,j}} \qquad (13)$$

$r \in [0,1]$ is the *reaction factor* that controls how quickly the weight adjustment mechanism reacts to changes in the effectiveness of the algorithms. If r is 0, we do not use update mechanism and weights stay at their initial values. If r is 1, then the weight of previous segment has no effect on the new weight; only the score decides the weight of the current segment. Apparently, if an algorithm is not used in the previous segment, then its π_a value will be 0 and the new weight is determined by the first term in the formula.

Because there are 2 types of removal-insertion algorithms (customer and station), their adaptive weight adjustment will also be done at different intervals. Number of segments is different for station removal and insertion algorithms. Hence, their weight updates are done with different intervals.

- **Adaptive Selection:** All algorithms are selected by a roulette-wheel mechanism independent from each other. Each of them has a selection probability which is dependent to the adaptive weight of the algorithm. Given k algorithms with $l = 1 \dots k$, let p_a^s denote the selection probability of algorithm a during segment s. This is calculated as follows:

$$p_a^s = \frac{w_{a,s}}{\sum_{l=1}^{k} w_{l,s}} \qquad (14)$$

- **Acceptance and Stopping Criteria:** A simple acceptance criterion would be to accept only solutions that are better than the current solution. However, this may cause getting trapped in a local minimum. Instead, we use a criterion from Simulated Annealing that accepts some worse solutions according to a probability. The probability of accepting a worse solution S_T is calculated as:

$$p = e^{\frac{-\left(f(S_T)-f(S_C)\right)}{T}} \qquad (15)$$

where S_C is the current solution and $T > 0$ is the temperature. Temperature starts at T_{start}. Similar to Ropke and Pisinger (2006a), T_{start} is dependent on the problem and it is set in such a way that S_T is accepted with probability 0.5 if it is μ (start temperature control parameter) percent worse than S_C. The temperature is decreased every iteration using the expression $T = T \times \varepsilon$ where $0 < \varepsilon < 1$ is a parameter called cooling rate.

If the objective is distance minimization, we always accept the solution with lower total distance value. If the new solution has the same or worse distance value, then simulated annealing is applied to determine accepting the new solution or not.

When we solve the problem with the hierarchical objective function, we accept the new solution if:

- It uses less number of vehicles than the previous solution or,
- It uses the same number of vehicles with the previous solution but the total distance traveled is shorter.

We do not accept the solution if it requires more vehicles than the previous solution. We apply the SA procedure if the new solution uses the same number of vehicles but its total distance is longer.

4.2. Customer Removal & Insertion Mechanism

4.2.1. Customer Removal Algorithms

In the first step of the algorithm, the current solution is destroyed by removing q customers from the solution according to different rules and adding them in a removal list $\mathcal{L}$. q is determined randomly using a uniform distribution and the removal rule cr_a is selected in an adaptive manner from the set of algorithms; A_{CR}. The generic structure of a customer removal procedure is given in Algorithm 1.

Algorithm 1: The generic structure of the customer removal procedure

input: Current feasible solution S_c, number of customers to be removed q
output: The set of customers which will be removed

1 Initialize removal list $(\mathcal{L} \leftarrow \emptyset)$
2 Apply a removal operator to find a set of customers for removal, ψ
3 $\mathcal{L} \leftarrow (\mathcal{L} \cup \psi)$
4 ***Return*** $\mathcal{L}$

Firstly, the number of customers which will be removed is determined. It is dependent to total number of customers and selected randomly between $\underline{n_c}$ and $\overline{n_c}$. Then, the selected rule is applied to the current feasible solution and q customers are selected and added to the removal list $\mathcal{L}$. In Figure 4.1a we see a feasible route. In Figure 4.1b *C6* is removed from the route and in Figure 4.1c, *C4* is also removed from the solution.

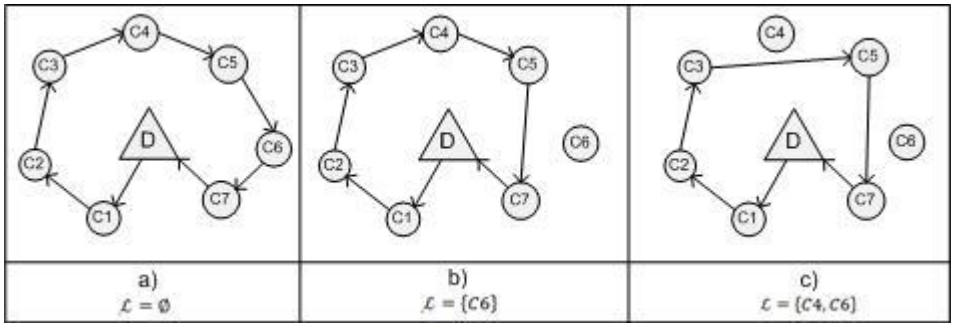

Figure 4.1: An illustration of a customer removal process.

We use 10 customer removal algorithms. The first eight are adapted from Ropke and Pisinger (2006a, 2006b), Pisinger and Ropke (2007), and Demir et al. (2012) and the last two are inspired from Emeç et al. (2013).

1. **Random Removal:** This algorithm simply selects q customers randomly. This random selection helps diversifying the search.
2. **Worst Distance Removal:** This algorithm calculates the cost of a customer as $g_j = \left|d_{ij} + d_{jk}\right|$ where d_{ij} is the distance between j and $i \in N$ which is the preceding node of j and d_{jk} is the distance between j and k which is the successing node of j in the corresponding route. If U is the ordered list of customers in this way, then the algorithm selects the customer $j^* = U[\lfloor \lambda^{\kappa} \rfloor]$ from S_C where $\lambda \in [0,1]$ is a random number and $\kappa \geq 1$ is a parameter called worst removal determinism factor which introduces randomness in the selection of customers in order to avoid choosing the same customers over and over again. This selection continues until q customers are chosen. This algorithm aims to make as much distance saving as possible with removal of customers with high deviation.
3. **Worst Time Removal:** This algorithm calculates time deviations of customers as $g_j = \left|\tau_j - e_j\right|$ where τ_j is the service start time and e_j is the early time window of customer j. Customers are ordered in non-increasing order of their time deviation. If O is the ordered list of customers in this way, then the algorithm selects the customer $j^* = O[\lfloor \lambda^{\kappa} \rfloor]$ from S_C. Selecting in this way continues until q customers are chosen. The idea behind this algorithm is to prevent long waits before going to a customer or delayed service starts by removing customers with high deviation.

4. **Shaw Removal:** The logic behind this algorithm which was introduced by Shaw (1998) is to remove customers that are similar to each other and therefore easy to change hence generating better solutions. If we choose customers which are different from each other, then we may gain worse solutions because we may only be able to insert the customers at their original positions or at some worse positions due to not finding any other proper positions to insert. The similarity of two customers *i* and *j* is defined with the relatedness measure

$$R(i,j) = \phi_1 c_{ij} + \phi_2 |\tau_i - \tau_j| + \phi_3 \omega_{ij} + \phi_4 |q_i - q_j| \qquad (16)$$

Where $\phi_1 - \phi_4$ are Shaw parameters.

$$\omega_{ij} = \begin{cases} -1, & if\ i\ and\ j\ are\ in\ the\ same\ route \\ 1, & otherwise \end{cases} \qquad (17)$$

The similarity of customers increases when $R(i,j)$ decreases. The algorithm firstly selects a customer randomly and adds it to the list. Then, other customers are sorted in non-decreasing order of their relatedness measures with the previous selected customer. If O is the ordered list, then algorithm selects the customer $j = O[\lfloor \lambda^{\eta} |O| \rfloor]$ where $\lambda \in [0,1]$ is a random number and $\eta \geq 1$ is a determinism parameter which introduces some randomness in the selection of the customers, i.e. low value of η corresponds to much randomness. This procedure is repeated until n_c customers are selected.

5. **Proximity Based Removal:** This algorithm is a special case of Shaw Removal which selects customers that are related in terms of the distance between them. Only difference is the parameter values which are taken as $\phi_1 = 1,\ \phi_2 = \phi_3 = \phi_4 = 0$. Figure 4.2 illustrates the mechanism of this algorithm.

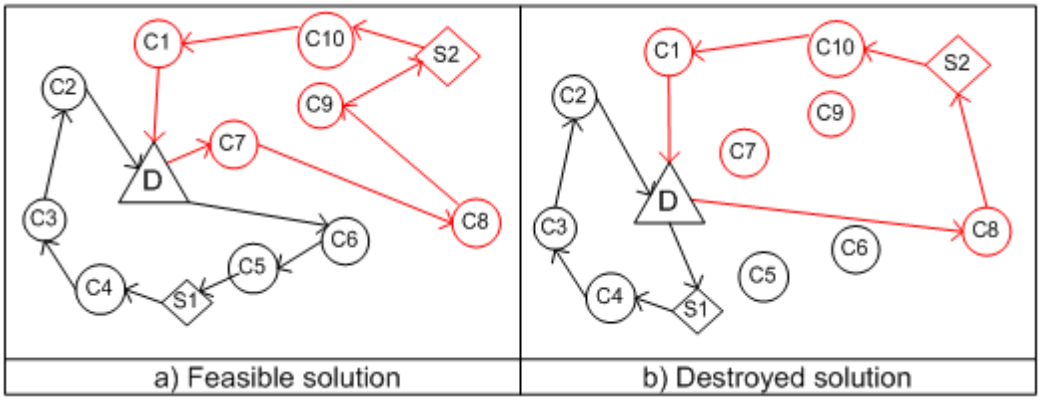

Figure 4.2: Proximity Based Removal. a) Feasible solution before removal, b) Partial solution after removal

6. **Time Based Removal:** This algorithm is another special case of Shaw Removal and selects customers which are similar in terms of their service beginning times. Only difference is the parameter values which are taken as $\phi_2 = 1,\ \phi_1 = \phi_3 = \phi_4 = 0$.
7. **Demand Based Removal:** This algorithm is again a special case of Shaw Removal and chooses customers which are similar in terms of their demands. Only difference is the parameter values which are taken as $\phi_4 = 1,\ \phi_1 = \phi_2 = \phi_3 = 0$.
8. **Zone Removal:** This algorithm is based on removal of nodes in a predefined area in the Cartesian coordinate system in which nodes are located (Demir et al., 2012). Firstly, the corner points of the area are specified by the maximum and minimum *x* and *y* coordinates of the customers, stations and the depot. Then the whole region is horizontally split up into smaller areas which are zones, according to the number of zones which is a parameter. At the end, each customer and station belong to a zone. The algorithm chooses a zone randomly and selects all customers in this zone. If that zone does not contain any customer, then another zone is selected randomly. Let $\mathbb{Z} = \{Z_1, Z_2, \dots Z_k\}$ be the set of randomly selected zones and $n_c^Z = n_c^{Z_1} + n_c^{Z_2} + \cdots + n_c^{Z_k}$ be the number of customers in these zones where Z_k represents the k^{th} randomly selected zone for which n_c^Z becomes firstly greater than or equal to n_c. The algorithm selects all customers in the zones $Z \in \mathbb{Z} \setminus Z_k$. Furthermore, if the number of customers in zone Z_k is greater than the remaining number of customers to be removed, it means all customers from that zone cannot be selected. Then the customers which belong to zone Z_k are sorted in non-decreasing order of their distance to the center of zone Z_k. Then $\left(n_c - \sum_{i=1}^{k-1} n_c^i\right)$ many closest customers are selected. Figure 4.3a illustrates the zones and the distribution of customers in the zones and Figure 4.3b shows the destroyed solution which will be obtained in the

update step of ALNS with zone removal algorithm. Here the customers in zone 5 are removed from the feasible solution. After the removal, the routes 2 and 3 (with dense dashed and dashed lines, respectively) are changed.

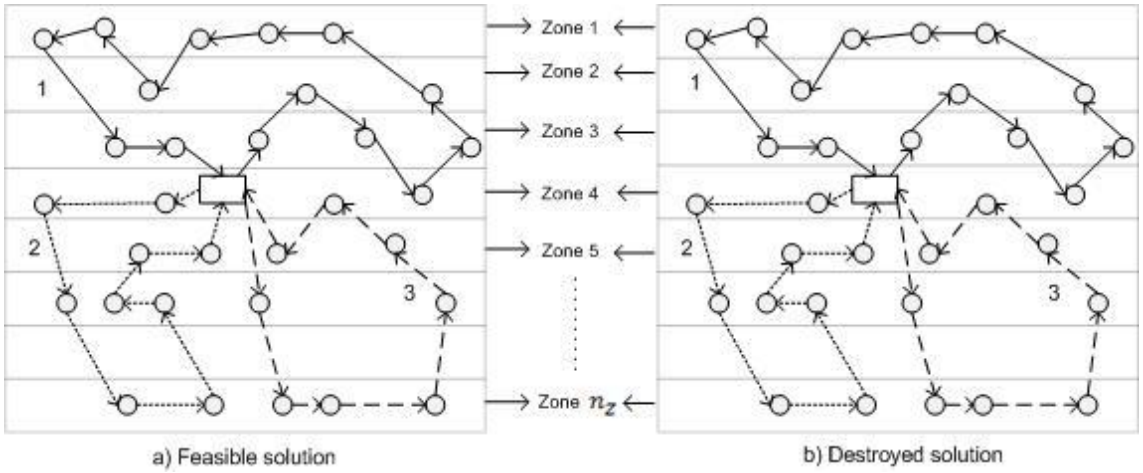

Figure 4.3: Zone Removal. a) Feasible solution before removal, b) Partial solution after removal

9. **Multiple Random Route Removal:** This operator randomly chooses r routes and removes all the customers in those routes. r is a parameter and depends on the number of routes in the current solution. It is determined randomly between 10% and m_r% of total number of routes.
10. **Multiple Greedy Route Removal:** This operator removes some routes in a greedy way. r is determined in the same way with Multiple Random Route Removal. The number of customers in each route is identified and then the route which has the least number of customers is removed from the solution. This continues until r routes are removed. This operator helps to distribute the customers in shorter routes into other routes in an attempt to reduce the total distance traveled. The process is illustrated in the Figure 4.4. Let us assume that r is 2, then firstly the customers in the route whose arcs are shown with dense dashed line will be removed because that route has the least number of customers. Then the customers in the route whose arcs are shown with dashed line will be removed due to having fewer customers than the other route.

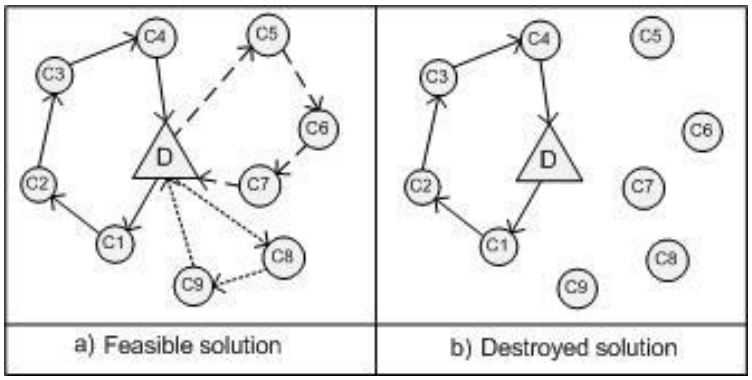

Figure 4.4: Multiple Greedy Route Removal. a) Feasible solution before removal, b) Partial solution after removal

4.2.2. Update Algorithms

After identifying the customers to be removed from the current feasible solution, those customers should be removed and the corresponding routes must be updated because at some nodes, the arrival charge, time and capacity features of some nodes will change with the removal of some customers. Those operations are done in the update procedure. Unlike the classical VRPTW, there may be stations in some routes due to charge constraint. Hence, in some cases, it might be useful to remove not only the customer but also the predecessor or successor station of the customer. We introduce three update procedures which are Remove Customer Only (RCO), Remove Customer with Predecessor Station (RCwPS), Remove Customer with Successor Station (RCwSS). Since route removal algorithms remove all customers and stations from the selected routes, update phase is not necessary when those algorithms are used at the removal step.

1. **Remove Customer Only:** RCO removes only the customers in the removal list from their routes. However, if the predecessor and successor of a customer are the same stations, then after the removal of that customer, two identical stations become successive in the route which is an unnecessary situation. Hence, this is checked in the update operation and if such a case occurs, one of those stations is also removed from the solution. Additionally, if the depot is used as a station in a route, after removal of intermediary customers, the depot which the vehicle begins or ends its route with and the depot which is used as a station become successive likewise. In

this situation, the depot which is used as station is removed from the solution. After the removal operation, the arrival time, arrival charge, capacity, departure time and departure charge of the customers and stations in the route are recalculated according to the new sequence. In order to decrease computational time by eliminating unnecessary operations, only customers and stations which come after the position of removed customer are considered since other nodes are not affected by the removal operation and the features of the vehicle at those nodes stay the same. In some cases, after removal of customers, the destroyed route may become infeasible in terms of charge. We need to make this route feasible again because in the next step the removed customers will be inserted to the routes and if the route stays infeasible, then during the insertion of customers, we need to insert additional stations which will increase the cost. Hence, Best Station Insertion algorithm is applied to make it feasible. The pseudo code of this algorithm is given in Algorithm2.

Algorithm 2: Remove Customer Only

input: Set of customers to be removed ψ, current feasible solution S_C
output: Destroyed solution S_P

```
1   while ψ ≠ ∅ do
2       Remove the first customer in ψ from S_C
3       Update the time and charge features of the corresponding route
4       if route is charge infeasible then
5           Perform Best Station Insertion
6       delete the removed customer from ψ
7   end while
8   Return S_P
```

2. **Remove Customer with Predecessor Station:** RCwPS does not only remove the customer in the removal list but it also removes the preceding station if any. The idea behind this removal is that the station which precedes the removed customer may be relevant to the customer. In other words, the station before the removed customer may have been located there in order to make the vehicle enough charged to go to the removed customer. Hence, with the removal of that customer, the station may become redundant because there would be enough charge to connect other customers. Under this assumption, if there is a station before the removed customer,

the algorithm removes it and then recalculates the arrival time, arrival charge, capacity, departure time and departure charge of the customers and stations at the nodes according to the new sequence. Like in previous procedure, only the nodes after the removed customer or station –if there is- are considered in order to eliminate unnecessary operations. Figure 4.5 shows a route before and after removing the customer *C4* and the station before *C4* from the route.

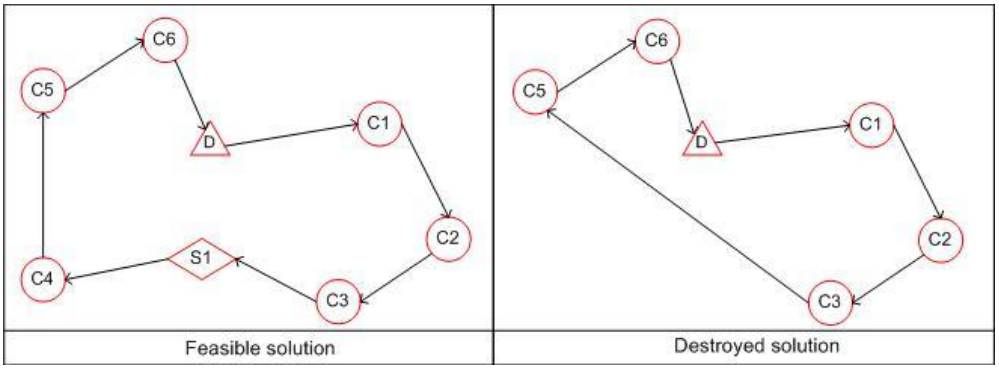

Figure 4.5: Removing customers and their predecessor stations. a) Feasible solution before removal, b) Partial solution after removal

Like in the Algorithm 2, if a route becomes infeasible after removal of some customers and stations, then we will call Best Station Insertion algorithm to make it feasible. The pseudo code of this algorithm is given in Algorithm 3.

Algorithm 3: Remove Customer with Predecessor Station

input: Set of customers to be removed ψ, current feasible solution S_C
output: Destroyed solution S_p

```
1   while ψ ≠ ∅ do
2       Remove the first customer in ψ from S_C
3       if there is a station just before that customer then
4           Remove that station from S_C
5       Update the time and charge features of the corresponding route
6       if route is charge infeasible then
7           Perform Best Station Insertion
8       delete the removed customer from ψ
9   end while
10  Return S_p
```

3. **Remove Customer with Successor Station:** RCwSS removes the customer in the removal list along with the succeeding station, if any. The idea is similar to RCwPS. The station after the removed customer may have been located there in order to make the vehicle enough charged to go from the removed customer. However, with the removal of that customer, the station may become redundant because there would be enough charge to connect other customers. After the removal process, arrival time, arrival charge, capacity, departure time and departure charge of the customers and stations in the route are recalculated according to the new sequence. Also in this algorithm, only the nodes after the removed customer are considered in order to eliminate unnecessary operations. Figure 4.6 illustrates the removal operation of customer *C3* and the station after *C3*.

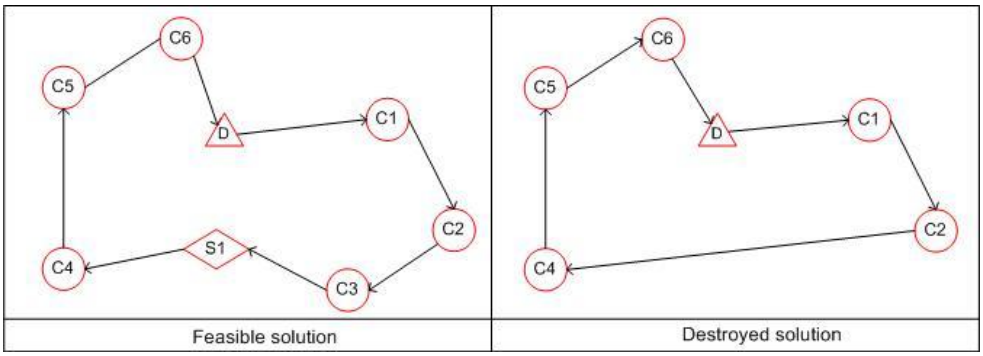

Figure 4.6: Removing customers and their successor stations. a) Feasible solution before removal, b) Partial solution after removal

Also in this algorithm, if a route becomes infeasible after removal of some customers and stations, then we will apply Best Station Insertion algorithm to make it feasible.

4.2.3. Customer Insertion Algorithms

After removing some customers from the current feasible solution, we need to repair the solution by reinserting the customers in the removal list to the partial feasible solution. The first four algorithms, Greedy Insertion, Zone Insertion, Regret-2 Insertion and Regret-3 Insertion are adopted from Ropke and Pisinger (2006a), Pisinger and Ropke (2007), and Demir et al. (2012). The last one, Time Based Insertion is newly proposed.

1. **Greedy Insertion:** This algorithm simply inserts customers to their best position in the fleet and the customer which has the least cost is chosen among others and

inserted first. The cost criteria c_i is determined for all customers in the removal list as follows: Customer *i* is inserted to a position *j* in route *k* and if this insertion does not violate time windows of any customer, the increase of the total distance of that route is calculated as f_{ijk}. If that insertion is feasible in time but infeasible in charge then a station should also be inserted into that route. If this is the case, the Greedy Station Insertion algorithm is used to find a station which will make the route feasible. After inserting a station, the cost of this insertion is calculated as the increase of the total distance after the insertion of that customer and corresponding station. After trial of all positions in the route, the position which has the minimum distance increase is determined and a cost $f_{i,k} = argmin_j\{f_{ijk}\}$ is assigned to customer *i* for route *k*. If the customer cannot be inserted into route *k*, then we set $f_{i,k} = \infty$. After analyzing all routes, the cost of opening a new route for that customer is also considered because that customer may not be inserted to any position in the existing routes. The cost of opening a new vehicle is just the multiple of the distance from depot to the customer if a vehicle can service that customer without visiting a station. If a station is needed to complete the route, then the whole distance including the station is considered as the new route opening cost. The position which increases the objective function the least is the one that has $c_i = min_k\{f_{i,k}\}$ for customer *i*. After analyzing all customers, the customer which has the $min_i\{c_i\}$ is selected to be inserted to its minimum cost position. If this insertion requires a station insertion, then the corresponding station is also inserted to the predetermined position in the route. Moreover, if the position which has $min\{c_i\}$ is in a new route, then a new vehicle is opened and added to the fleet. After the insertion is performed, that customer is removed from the removed customers list and arrival time, arrival charge, capacity, departure time and departure charge of the vehicle in which the insertion is performed are recalculated according to the new sequence. On the other hand, only the nodes after the removed customer are considered in order to eliminate unnecessary operations. Then, $f_{i,k}$ values of remaining customers for the selected route are recalculated because the route is changed due to the insertion of the previous customer. Thus, the insertion costs and insertion places would be different. Furthermore, if a new route is opened in the previous iteration, then insertion costs $f_{i,k}$ where *k* is the new route are calculated for remaining customers. After updating the costs of remaining customers, the

customer which has the minimum c_i is selected to be inserted to its minimum cost position again and until all removed customers are inserted, this procedure is repeated. Pseudo code of the Greedy Insertion is given in Algorithm 4.

Algorithm 4: Greedy Insertion

input: Destroyed feasible solution S_1, removal list $\mathcal{L}$
output: Repaired feasible solution S_2

```
Initialize the insertion cost hash map as (Costs ← ∅)
for each customer in ℒ
    for all positions in all routes
        Insert the customer to the position
        if this insertion does not violate time windows of the route then
            if charge feasibility of the route is violated then
                Perform Greedy Station Insertion
                    if time and charge feasibility is satisfied then
                        Record the total insertion cost to Costs
            else
                Record the total insertion cost to Costs
    end for
end for
while ℒ ≠ ∅ do
    Sort the values in Costs in non-decreasing order
    Select the first value in Costs and perform the insertion corresponding
      to that cost
    if this insertion requires a station insertion then
        Insert the station to the corresponding position
    Update the features of the route to which the insertion is done
    Delete the costs of recently inserted customer from Costs
    Update the costs of insertions to the recently updated route
      for other customers in Costs
    Delete the customer from ℒ
end while
Return S2
```

2. **Regret-*k* Insertion:** Greedy heuristic often postpones the placement of customers which are expensive to insert to the last iterations because it always selects the customer with the least cost. The regret heuristics try to circumvent this problem by incorporating a kind of look-ahead information when selecting the customer to insert (Ropke and Pisinger, 2007a). Let Δf_i^k denote the change in the objective function value incurred by inserting customer i into its k^{th} best position in all routes. For example, Δf_i^2 corresponds to the change in the objective function value

incurred by inserting customer *i* into its second best position. If an insertion requires a station insertion due to charge infeasibility, then the increase in the objective function is calculated with the station insertion like in the greedy insertion. After calculating Δf_i^k values for all customer in the removal list, the heuristic chooses the customer *i* which has $argmax_i\{\Delta f_i^k - \Delta f_i^1\}$. Then the customer is inserted to its minimum cost position. After an insertion is performed, the route to which a customer is inserted is updated by means of time, capacity and charge. Only the nodes after the removed customer are considered in order to eliminate unnecessary operations. Then, $\Delta f_i^k - \Delta f_i^1$ values of remaining customers for the changed route are recalculated because the route is changed due to the insertion of the customer. Thus, the insertion costs and insertion places would be different. Furthermore, if a new route is opened, then insertion costs to the new route are calculated and considered in the recalculation of $\Delta f_i^k - \Delta f_i^1$ values for the remaining customers. Finally, the procedure is repeated for remaining customers in the removal list until all customers are inserted to the solution.

3. **Zone Insertion:** This algorithm inserts customers in a time based manner. The logic behind this algorithm is leaving enough space for future insertions by selecting the insertions according to time windows instead of distance. Firstly, the routes which pass through each zone are determined. For instance, the route 1 in the Figure 4.7 (with solid arcs) passes through zones 1, 2, 3 and 4 whereas the route 2 (with dense dashed arcs) passes through zones 4, 5, ... n_Z. Moreover, zone 1, 2 and 3 only have the route 1 whereas zone 4 has routes 1, 2 and 3 (with dashed arcs).

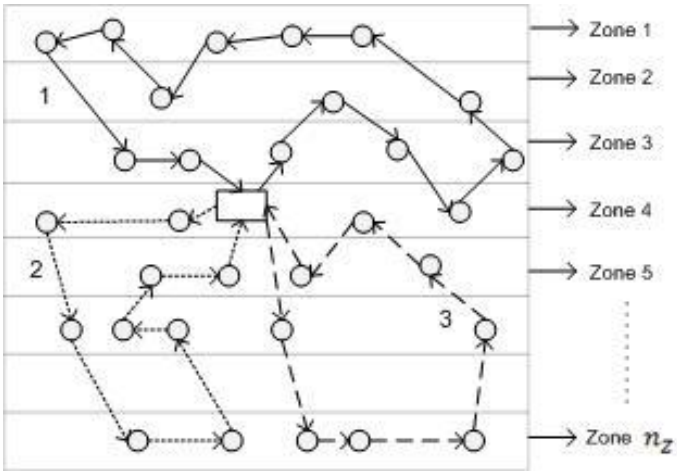

Figure 4.7: Routes in the zones

After determining route distribution among the zones, a customer is selected randomly from the removal list. Then it is inserted to the positions in the routes which pass through the zone in which that customer is located and the customer which has the least insertion cost is chosen among others and is inserted first. For instance, if the first customer of the red route is selected, insertions to all routes will be considered. However, if the first customer of brown route is selected, then insertions to red and brown routes will be considered. The cost criteria c_i is determined for all customers as follows: When customer *i* is inserted to a position *j* in route *k* and if this insertion does not violate time windows of any customer, the time delay of the inserted customer is calculated as $f_{ijk} = max\{0, (\tau_i - e_i)\}$. This expression stands for the waiting time of customer *i* when it is inserted to the position *j* in the route *k*. If that insertion is feasible in time but infeasible in charge then a station should also be inserted to that route. If this is the case, Greedy Station Insertion algorithm is used to find a station which will make the route feasible. After inserting a station, cost of this insertion is calculated as mentioned above. After trial of all positions in the route, the position which has the minimum waiting time is determined and a cost $f_{i,k} = argmin_j\{f_{ijk}\}$ is assigned to customer *i* for route *k*. If the customer cannot be inserted route *k*, then we set $f_{i,k} = \infty$. Then the position which yields least waiting time is determined as having $c_i = min_k\{f_{i,k}\}$ and customer *i* is inserted to that position. If this insertion requires a station insertion, then the corresponding station is also inserted to the predetermined position in the route. In some cases, any position may not be found for those routes. For such customers, Greedy Insertion algorithm is applied instead of Zone Insertion. After the insertion is performed, that customer is removed from the removed customers list and arrival time, arrival charge, capacity, departure time and departure charge of the customers and stations in the route are recalculated according to the new sequence. Again, only the nodes after the removed customer are updated in order to eliminate unnecessary operations.

After the insertion, route *k* may pass through a new zone. Hence, if the updated route passes through a new zone, new route distribution is determined for those zones. Then another customer from the list is selected randomly and the above procedure is repeated until all removed customers are inserted.

4. **Time Based Insertion:** This algorithm combines greedy insertion with the logic of zone insertion. It inserts customers to their best position in the fleet and the customer which has the least cost is chosen among others and inserted first. Here, the cost criteria c_i is the waiting time of the customer i as in Zone Insertion. All other steps of the algorithm are the same as those in the Greedy Insertion.

In customer insertion algorithms, assigning a customer to a new route costs the total of distance from depot to that customer and the distance from that customer to the depot. This is valid for distance minimization objective. If we solve the problem with hierarchical objective function, then we need to assign a big cost to the new route in order to motivate decreasing number of vehicles.

4.3. Station Removal & Insertion Mechanism

After customer removal and insertion, the first part of the ALNS framework is completed. In the second part, we will destroy the current feasible solution by removing recharging stations because stations are the crucial part of this problem. Hence, changing their positions in the visit sequence of a route may also improve the solution. Between a pre-determined number of iterations, a station removal and insertion procedure is applied. The number of stations to be removed n_s is determined in a similar fashion to q. Firstly, the total number of stations in the current solution is calculated. Then n_s is selected randomly between 10% and 40% of total number of stations. There are three types of station removal algorithms which are Worst Distance, Worst Charge Usage and Random Station Removal.

4.3.1. Station Removal Algorithms

1. **Random Station Removal:** This algorithm simply selects n_s stations randomly and removes them from the current feasible solution. This random selection contributes diversification of the search.
2. **Worst Charge Usage Station Removal:** The main idea of this algorithm is to increase the efficiency of usage of the stations. We want the number of stations to be as small as possible because going to a station causes an increase in distance. Hence, a vehicle should go to a station at its minimum charge level. In other words, a vehicle should go to a station if it does not have enough charge to travel any other

customer. From this point of view, it would be reasonable if we remove those stations to which a vehicle goes with high charge level. Firstly, the arrival charge of the vehicle for each station in the fleet is examined and added to a list, i.e. List1. After analyzing all stations, List1 is ordered in non-increasing order of charge levels which are found as indicated above. Then the algorithm removes the station which has the first value in the List1. If the number of stations in the whole fleet is smaller than or equal to n_s, then all the stations in the solution are removed. For the other case, the above procedure is repeated until n_s stations are selected and removed from the current solution. With this algorithm, it is expected that a new station which causes less distance increase is inserted to the route. Hence, the utilization of stations increases while distance decreases. Pseudo code of this removal operation is given in Algorithm 5.

Algorithm 5: Worst Charge Usage Station Removal

input: Current feasible solution S_1, number of stations to be removed n_s
output: Destroyed solution S_2, Route list to be repaired $\mathcal{L}$

```
Initialize the charge status hash map as C ← ∅
Initialize number of removed stations as q ← 0
for all stations in S_1
        Determine the arrival charge of the vehicle to that station
        Record that charge level in C with the station ID
end for
Sort the charge levels in C in non-increasing order
while q < n_s
        Select the first element in C and identify the station
        corresponding to that charge level
        Remove that station from its route
        q ← q + 1
        if the route number is not added in L before then
            Add the route number to L
end while
Return S_2 and L
```

3. **Worst Distance Station Removal:** This algorithm is similar to the worst distance customer removal. Costs of the stations are calculated as $g_j = \left|d_{ij} + d_{jk}\right|$ where d_{ij} is the distance between j and $i \in N$ which is the preceding node of j and d_{jk} is the distance between j and k which is the successing node of j in the corresponding

route and added to a list, i.e. List1. Then List1 is sorted in non-decreasing order and the station which has the highest cost $j^* = argmax_{j \in N}\{g_j\}$ is removed from the solution. This process is repeated until n_s stations are removed. This algorithm aims to decrease total distance by removing stations with high distance deviation.

4.3.2. Station Insertion Algorithms

After removing some stations, the solution may become charge infeasible. In order to make infeasible routes feasible, station insertion algorithms are used. These algorithms insert stations to the infeasible routes. The difference with customer insertion algorithms is that the algorithm does not necessarily insert the stations which are removed in the station removal phase. Because stations are always available and assumed to be infinitely many, any station can be inserted throughout the algorithm. There are three station insertion mechanisms.

At the beginning of all station insertion algorithms, feasibility of destroyed routes is checked. A destroyed route may have become infeasible in terms of charge, time or both of them. We can explain these situations through the following figures:

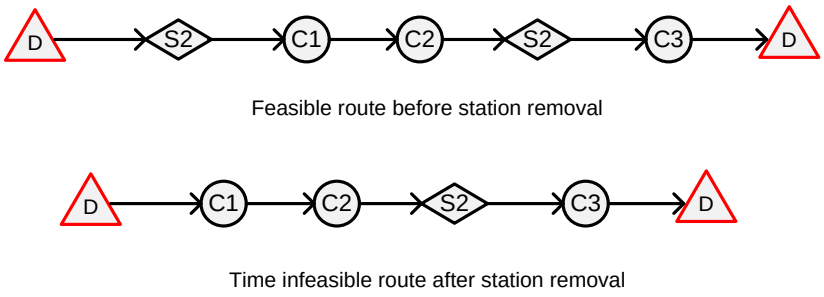

Figure 4.8: Simple illustration of a station removal process.

Consider the route shown in Figure 4.8. After the removal of station *S2* which is before *C1*, arrival times of *C1* and *C2* will either be the same (if arrival time of *C1* is earlier than its early arrival time) or earlier than the former case due to elimination of charging and traveling time of the station. However, the arrival charge of the vehicle at *S2* which is after *C2* will be smaller because the vehicle did not visit a station like before. That means charging will be longer and arrival time to *C3* will be later. If that time is later than the late arrive time of *C3*, then the vehicle is late for *C3* which means the route is infeasible in time. In order to accomplish this situation, we need to insert a station or stations before the node of which arrival time is later than its late arrive time.

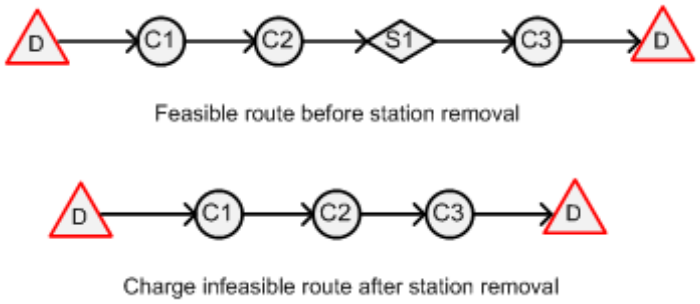

Figure 4.9: Routes before and after station removal

Figure 4.9 illustrates how the charge infeasibility may occur after the station removal. Features of *C1* and *C2* will be the same after the removal. However, departure charge at *C2* may not be enough to travel from *C2* to *C3*. Or it may arrive to *C3* but not have enough charge to return Depot. In order to overcome this situation, we need to insert a station or stations before the node of which arrival charge is smaller than 0.

At the beginning of the station insertion, we need to discard time infeasibility. Firstly, we identify the node whose arrival time is greater than its early arrive time. Then we insert the stations beginning from the position just before that node. We insert the stations beginning with the nearest station for the corresponding position. If the nearest station does not make the route feasible, then we continue trying with further stations until we obtain feasibility. If any station for the position between *C1* and *C2* still does not make the route feasible, we repeat the same operations for the preceding positions until feasibility is satisfied.

1. **Greedy Station Insertion:** This station insertion algorithm inserts stations in a greedy manner. Firstly, the algorithm identifies the node which has the first negative arrival charge in the route. Then it inserts the nearest station to the position just before the negative node. If the arrival charge of the node which has the first negative arrival charge becomes positive and time window feasibility of all nodes in the route does not violated, then this station is inserted to that position. If that station does not make the arrival charge of that node positive or violates time windows, then the previous positions are investigated until a proper station is found. We are looking at the positions between the negative node and the first station before that node. If there is no station before that node, we look at the positions between that

node and the depot at the beginning of the route. We restrict the positions in this way because we can make the arrival charge of that node positive by inserting only to the previous positions. And because vehicles are fully charged at a station, it is useless to look at the positions before a station; the vehicle will leave that station fully charged anyway. In addition, since we did not check the route feasibility while analyzing the stations, that insertion may not make the whole route feasible. If there are any nodes which have negative arrival charge, then the procedure is repeated for the new negative charged node until the whole route become charge feasible. The pseudo code of this algorithm is given in Algorithm 6.

2. **Best Station Insertion:** This algorithm tries to insert the best station in terms of distance in order to make the arrival charge of the first negative charged node positive. Firstly, the algorithm inserts the nearest station to the position just before the first negative charged node. If this insertion makes the arrival charge of the negative charged node positive and does not violate time window feasibility of all nodes in the route, then the increase of the total distance of the route is added to a list i.e. List1. This distance increase is kept for comparing the stations and selecting the best one. Then the previous position is analyzed likewise and this backward investigation continues until we reach a station.

   Then List1 is sorted in non-decreasing order and the insertion which has the smallest distance increase, first in List1, is performed. After the insertion, arrival time and charge, departure time and charge information of the station inserted and the customers which are after the insertion position are updated accordingly. Since we did not check the route feasibility while analyzing the stations, that insertion may not make the whole route feasible. Thus, if it is the case, the node which has the first negative charge is again identified and the algorithm is reoperated in the same manner until all nodes have positive arrival charge.

Algorithm 6: Greedy Station Insertion

input: Route list to be repaired $\mathcal{L}$ and destroyed solution S_1
output: Repaired feasible solution S_2

```
1   while ℒ ≠ ∅ do
2   Select the first route from ℒ
3   if the route is time feasible but charge infeasible then
4       while the route is charge infeasible do
5           Identify the first node which has negative arrival charge
6           while the arrival charge of that node is negative do
7               for all positions before that node
9                   Insert that nearest station to that position
10                  if the arrival charge of that node
                        becomes positive then
11                      if time window feasibility of the route
                            is not violated then
12                  end for
13              end for
14              if the arrival charge of that node is still negative then
15                  Insert the stations as before the station removal
16                  Go to step 36
17          end while
18      end while
19  else if the route is time infeasible then
20      while the route is time infeasible do
21          Identify the first node whose arrival time is
              greater than its late arrive
22          while its time window feasibility is not satisfied do
23              for all positions before that node
24                  for all stations
25                      Insert the station to that position
26                      if the time window of the identified node
                            becomes feasible then
27                      end for
28                  end for
29              end for
30              if the arrival time of that node is still greater than
                    its late arrive then
31                  Insert the stations as before the station removal
32                  Go to step 36
33          end while
34      end while
35      Recall steps 3 − 18
36  Delete the route from ℒ
37  end while
38  Return S2
```

3. **Greedy Station Insertion with Comparison:** This algorithm is a more forward looking version of the greedy station insertion. While greedy station insertion performs insertion to the first position of which a feasible station is found, this algorithm also analyzes the previous position of that position. Firstly, the nearest station for the position just before the negative arrival charged node is inserted to that position. If the arrival charge of the node which has the first negative arrival charge becomes positive and time window feasibility of all nodes in the route is not violated, the distance increase of this insertion is kept. Then the same procedure is applied to the previous position. After that, those two stations are compared in terms of distance increase and the better one is selected for insertion. After the insertion, arrival time and charge, departure time and charge information of the station inserted and the customers which are after the insertion position are updated accordingly. Since we did not check the route feasibility while analyzing the stations, that insertion may not make the whole route feasible. Thus, if it is the case, the node which has the negative charge is again identified and the algorithm is reoperated in the same manner until all nodes have positive arrival charge.

   This algorithm works like above if the stations for the first and second positions are feasible. In other words, if the nearest station for the first position is not feasible, then we assume that the cost of inserting a station in the first position ∞. Hence, the nearest station of the previous position will have less cost compared to ∞ and it will be inserted. The same rule is valid when the station of the first position is feasible and the station of the previous position is not. Then insertion cost in the second position will be ∞ and the station of the first position will be inserted. Consequently, in those cases, this algorithm works like greedy station insertion.

In some cases, those station insertion algorithms are not able to make the whole route feasible. It may occur because the route might have destroyed too much and algorithms cannot find a feasible station because instead of looking all stations in the station list, we look the nearest stations for each position. Hence, a proper sequence may not have been found by those stations. If this is the case, we cancel the removal process for this route and reinsert the stations which are removed in the station removal step.

Figure 4.10 illustrates a possible improvement after the station removal and insertion in a route.

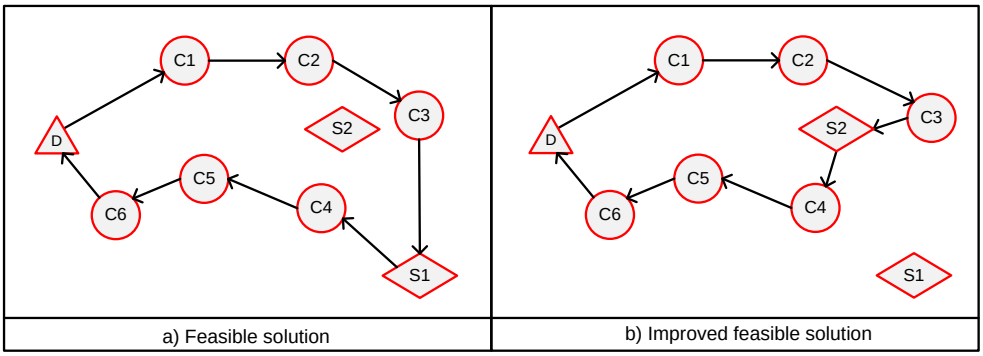

Figure 4.10: Improvement after station removal and insertion operations

The generic structure of the algorithm for the hierarchical objective is given in Algorithm 7. The algorithm for distance minimization objective is very similar and is provided in Appendix C.

Algorithm 7: ALNS algorithm for hierarchical objective case

input: $A_{CR}, A_{CI}, A_{SR}, A_{SI}, A_U, N_I, N_{RW1}, N_{RW2}, N_{RR}, \mu, \varepsilon, \tau$

output: The best solution S_{best}

Generate an initial solution S_i by using Greedy insertion algorithm
Initialize selection probabilities $p^s_{cr_a}, p^s_{ci_a}, p^s_{sr_a}, p^s_{si_a}, p^s_{u_a}$ for each $cr_a \in A_{CR}$, $ci_a \in A_{CI}$, $sr_a \in A_{SR}$, $si_a \in A_{SI}$, $u_a \in A_U$
Initialize T and T_{start} by using $f(S_i)$ and μ
Let j be the outermost iteration counter initialized as $j \leftarrow 1$
Let $S_C \leftarrow S_B \leftarrow S_i$
while $j \leq N_I$ ***do***
    Select a customer removal algorithm $cr_a \in A_{CR}$ with probability $p^s_{cr_a}$
    Select an update mechanism $u_a \in A_U$ with probability $p^s_{u_a}$
    Generate S_1 by applying cr_a and u_a to S_C
    Select a customer insertion algorithm $ci_a \in A_{CI}$ with probability $p^s_{ci_a}$
    Generate S_2 by applying ci_a to S_1
    if $z(S_2) < z(S_C)$ ***then***
        $S_C \leftarrow S_1$
    else if $z(S_2) = z(S_C)$ ***then***
        if $f(S_2) < f(S_C)$ ***then***
            $S_C \leftarrow S_1$
        else
            Let $p = e^{\frac{-(f(S_T)-f(S_C))}{T}}$
            Generate a random number $\vartheta \in [0,1]$
            if $\vartheta < p$ ***then***
                $S_C \leftarrow S_1$
    if $f(S_C) < f(S_B)$ ***then***
        $S_B \leftarrow S_C$
    if $j \equiv 0 \ (mod\ N_{RW1})$ ***then***
        Update the probabilities $p^s_{cr_a}, p^s_{ci_a}, p^s_{u_a}$ using the adaptive weight procedure
    Update the current temperature as $T \leftarrow T.\varepsilon$
    Increase the counter as $j \leftarrow j + 1$
    if $j \equiv 0 \ (mod\ N_{SR})$ ***then***
        Select a station removal algorithm $sr_a \in A_{SR}$ with probability $p^s_{sr_a}$
        Generate S_3 by applying sr_a to S_2
        if S_3 is infeasible ***then***
            Select a station insertion algorithm $si_a \in A_{SI}$ with probability $p^s_{si_a}$
            Generate S_4 by applying si_a to S_3
        Recall the steps $\mathbf{12-23}$ with S_4 instead of S_1
        if $j \equiv 0 \ (mod\ N_{RW2})$ ***then***
            Update the probabilities $p^s_{sr_a}, p^s_{si_a}$ using the adaptive weight procedure
        Recall steps $\mathbf{26-27}$

```
38  if  j ≡ 0 (mod N_RR) then
39      Let k the counter of route minimization procedure as k ← 1
40      while k ≤ τ do
41          Let  S_C ← S_B
42          Select a route removal algorithm from cr_a ∈ A_CR with probability p^s_cr_a
43          Generate S_5 by applying cr_a to S_C
44          Select a customer insertion algorithm ci_a ∈ A_CI with probability p^s_ci_a
45          Generate S_6 by applying ci_a to S_5
46          Recall the steps 12 − 23 with S_6 instead of S_1
47          if  j ≡ 0 (mod N_RW1) then
48              Update the probabilities p^s_ci_a using the adaptive weight procedure
49          Recall the steps 26 − 27
50          if  z(S_C) < z(S_B) then
51              Update the probabilities p^s_cr_a using the adaptive weight procedure
52              break while
53          else
54              k ← k + 1
55          Recall steps 28 − 36
56          Recall steps 50 − 58
57      end while
58  end while
59  Return S_B
```

Chapter 5

Computational Experiments

In this chapter, we perform computational experiments to validate the performance of the proposed ALNS approach. We first tune the parameters using a subset of instances and determine their values separately for the distance minimization and hierarchical objectives. Then we test the performance of the proposed ALNS using the 36 small and 56 large instances generated by Schneider et al. (2014) based on the well-known VRPTW instances of Solomon. The large set involves three main problem classes where 100 customers and 21 recharging stations are clustered (C), randomly distributed (R), and both clustered and randomly distributed (RC) over a 100×100 grid. Each set has also two subsets, type 1 and type 2, which differ by the length of the time windows and the vehicle capacity. The small set includes three subsets of 12 problems with 5, 10 and 15 customers in each subset, respectively.

The battery capacity is set to the maximum of the following two values: (i) the charge needed to travel 60% of the average route length of the best known solution to the corresponding VRPTW instance; and (ii) twice the amount of battery charge required to travel the longest arc between a customer and a station. This procedure ensures that instances with geographically disperse and remote customers stay feasible. Furthermore, the instances guarantee that recharging stations have to be used. For the sake of simplicity, the consumption rate is assumed 1.0 and the recharging rate g is set so that a complete recharge requires three times the average customer service time of the respective instance.

The algorithm is coded on Java programming language and all experiments are performed on an Intel Core i7 processor with 3.40 GHz speed and 16 GB RAM, and 64-bit Windows 7 operating system.

5.1. Parameter Tuning

We adopted a tuning methodology similar to that of Ropke and Pisinger (2006a). We selected six large problems and performed ten runs for each parameter by considering the initial values as described in Ropke and Pisinger (2006a, 2006b), Pisinger and Ropke (2007), and Demir et al. (2012). For the new parameters, we determined a selection of reasonable values inspiring from Ropke and Pisinger (2006a). We omitted C1 and C2 problem classes since they usually converge to same solutions for different parameter values and do not provide much information about the contribution of the parameter value on the solution quality. Consequently, we selected the instances R107, RC101, RC104, RC105, R205 and RC205 for parameter tuning.

At each step, we allow one parameter to take a number of predefined values while the rest of the parameters are kept fixed. For each parameter, we run the heuristic ten times on the tuning instances and we select the value that gives the least average deviation from the best achieved solution. After a parameter value is determined, its value is fixed and this procedure is repeated for the remaining parameters until all parameters have been tuned. The details of the parameter setting, tuning sequence, deviations and final values are given in Appendix A.

Although many parameters take different values we observe that the score of the worse solution (σ_3) is greater than the score of the better solution (σ_2) which allows diversification by rewarding non-improved solutions as in Ropke and Pisinger (2006a) and Demir et al. (2012).

Ropke and Pisinger (2006a) set the number of iterations to 25,000 and noted that additional runtime had minor contribution to the solution quality. Our convergence analysis showed similar results. So, we also performed 25,000 iterations.

In addition, $\underline{n_c}$ and $\overline{n_c}$ are taken as $0.1|N|$ and $0.4|N|$ in Ropke and Pisinger (2006a). We also use those values, did not include them in the parameter tuning.

5.2. Experimental Study on Small Instances

We solve all small instances to optimality by CPLEX. ALNS is run for 25 times for each instance and it was able to find the optimal solutions. We give the optimal results for distance minimization case in the Appendix B. Results for hierarchical objective case given in Schneider et al. (2014) include optimal solutions and upper bounds obtained in 7200 seconds. However, we solved all of them optimally and proved that those upper bounds are optimal. ALNS also successfully found those results.

5.3. Experimental Study on Large Instances

5.3.1. Hierarchical Objective Function Case

5.3.1.1. Numerical Results

We compare our solutions on large instances with the results reported by Schneider et al (2014) in Table 5.1. Schneider et al. (2014) presented the best solution found using (i) the hybrid VNS and TS with an SA acceptance criterion (denoted as VNS/TS), (ii) VNS/TS only accepting improving solutions, (iii) pure TS as well as the best known solution they observed throughout all computational study including the parameter tuning. The first column denotes the instance. The second and third columns report the best solutions, i.e. the number of vehicles (#Veh) and total distance (TD), found by Schneider et al. (2014) throughout all experiments they performed while the results achieved by ALNS for the hierarchical objective denoted by ALNS (Hier) are given in the following columns. The column "#Rech" reports the total number of recharges in the best solution found. The column "ΔTD" shows the percentage difference between the distance in Schneider et al. (2014) and that found by ALNS (a negative value implies improvement). The following four columns compare the best solutions found by VNS/TS (the best method among the three algorithms) to those of ALNS after all the parameters of the two approaches were tuned. Note that Schneider et al. (2014) performed 10 runs while ALNS was run 25 times with fixed parameters. The last column gives the average computational time in minutes. Finally, "#Better" and "#Better or Same" at the bottom of the table denote the number of instances in which ALNS showed better performance and better than or same performance, respectively, compared to Schneider et al (2014).

Table 5.1: ALNS results for hierarchical objective function

	Best in All Computational Tests						Best with Fixed Parameters					
	Schneider et al.		ALNS (Hier)			ΔTD	VNS/TS		ALNS (Hier)		ΔTD	
Problem	**#Veh**	**TD**	**#Veh**	**TD**	**#Rech**	**(%)**	**#Veh**	**TD**	**#Veh**	**TD**	**(%)**	***t* (min)**
c101	12	1053.83	**12**	**1053.83**	8	0.00	12	1053.83	**12**	**1053.83**	0.00	1.28
c102	11	1056.47	**11**	**1056.12**	9	-0.03	11	1057.16	**11**	**1056.12**	-0.10	2.28
c103	10	1041.55	11	1001.81	7	-	10	1041.55	11	1002.60	-	4.18
c104	10	979.51	**10**	**951.57**	7	-2.85	10	980.82	11	969.46	-	7.02
c105	11	1075.37	**11**	**1075.37**	9	0.00	11	1075.37	11	1080.85	0.51	1.51
c106	11	1057.87	**11**	**1057.65**	9	-0.02	11	1057.87	**11**	**1057.65**	-0.02	1.71
c107	11	1031.56	**11**	**1031.56**	9	0.00	11	1031.56	**11**	**1031.56**	0.00	1.80
c108	10	1100.32	11	1015.68	8	-	10	1100.32	11	1015.68	-	2.17
c109	10	1036.64	11	993.77	9	-	10	1051.84	11	1004.36	-	2.97
c201	4	645.16	**4**	**645.16**	4	0.00	4	645.16	**4**	**645.16**	0.00	3.12
c202	4	645.16	**4**	**645.16**	4	0.00	4	645.16	**4**	**645.16**	0.00	16.61
c203	4	644.98	**4**	**644.98**	4	0.00	4	644.98	**4**	**644.98**	0.00	38.18
c204	4	636.43	**4**	**636.43**	4	0.00	4	636.43	**4**	**636.43**	0.00	74.21
c205	4	641.13	**4**	**641.13**	3	0.00	4	641.13	**4**	**641.13**	0.00	8.42
c206	4	638.17	**4**	**638.17**	4	0.00	4	638.17	**4**	**638.17**	0.00	18.42
c207	4	638.17	**4**	**638.17**	4	0.00	4	638.17	**4**	**638.17**	0.00	21.63
c208	4	638.17	**4**	**638.17**	4	0.00	4	638.17	**4**	**638.17**	0.00	27.41
r101	18	1670.80	18	1679.06	23	0.49	18	1672.55	19	1659.47	-	2.42
r102	16	1495.31	17	1480.10	19	-	16	1535.81	17	1480.10	-	3.00
r103	13	1299.17	14	1269.20	17	-	13	1299.64	14	1269.20	-	3.45
r104	11	1088.43	12	1071.89	11	-	11	1088.43	12	1073.75	-	4.29
r105	14	1461.25	15	1383.29	19	-	14	1473.59	15	1428.10	-	2.65
r106	13	1344.66	14	1276.33	18	-	13	1344.66	14	1276.33	-	3.13
r107	12	1154.52	**12**	**1148.43**	14	-0.53	12	1154.52	**12**	**1148.62**	-0.51	3.48
r108	11	1050.04	11	1051.59	13	0.15	11	1065.89	11	1067.32	0.13	4.88
r109	12	1294.05	13	1214.72	14	-	12	1294.05	13	1246.65	-	3.87
r110	11	1126.74	12	1097.89	12	-	11	1143.52	12	1104.72	-	2.51
r111	12	1106.19	12	1109.14	15	0.27	12	1124.06	**12**	**1111.86**	-1.09	2.49
r112	11	1026.52	11	1038.74	14	1.19	11	1026.52	12	1045.42	-	3.20
r201	3	1264.82	3	1265.67	7	0.07	3	1264.82	3	1325.90	4.83	10.36
r202	3	1052.32	**3**	**1052.32**	3	0.00	3	1052.32	3	1055.48	0.30	25.83
r203	3	895.91	**3**	**895.54**	4	-0.04	3	912.86	**3**	**895.54**	-1.90	40.78
r204	2	790.57	**2**	**780.98**	3	-1.21	2	790.57	3	720.51	-	65.74
r205	3	988.67	**3**	**987.36**	3	-0.13	3	988.67	**3**	**987.36**	-0.13	17.39
r206	3	925.20	**3**	**922.70**	3	-0.27	3	925.20	3	925.37	0.02	27.79
r207	2	848.53	2	850.80	2	0.27	2	852.73	**2**	**851.75**	-0.11	38.98
r208	2	736.60	**2**	**736.12**	2	-0.07	2	736.60	**2**	**736.12**	-0.07	160.85
r209	3	872.36	**3**	**871.22**	4	-0.13	3	872.36	3	876.54	0.48	27.82
r210	3	847.06	**3**	**843.65**	3	-0.40	3	847.06	**3**	**846.96**	-0.01	43.03
r211	2	847.45	3	761.56	1	-	2	866.21	3	761.56	-	46.81
rc101	16	1731.07	**16**	**1731.07**	17	0.00	16	1731.07	16	1757.09	1.50	1.26
rc102	15	1554.61	**15**	**1551.69**	17	-0.19	15	1554.61	**15**	**1552.58**	-0.13	1.77
rc103	13	1351.15	13	1351.73	14	0.04	13	1353.55	13	1365.91	0.91	2.12
rc104	11	1238.56	**11**	**1232.45**	13	-0.49	11	1249.23	12	1232.91	-	2.90
rc105	14	1475.31	**14**	**1473.24**	16	-0.14	14	1483.38	14	1499.42	1.08	1.65
rc106	13	1437.96	14	1414.99	15	-	13	1440.19	14	1425.64	-	1.75
rc107	12	1279.08	12	1283.05	14	0.31	12	1275.89	12	1304.89	2.27	2.33
rc108	11	1209.61	**11**	**1209.11**	14	-0.04	11	1238.81	12	1231.89	-	2.80
rc201	4	1444.94	4	1446.84	4	0.13	4	1447.20	4	1453.87	0.46	5.52
rc202	3	1412.91	3	1450.34	6	2.65	3	1412.91	4	1243.55	-	12.08
rc203	3	1073.98	**3**	**1069.27**	5	-0.44	3	1078.28	3	1082.04	0.35	23.76
rc204	3	885.35	3	887.76	4	0.27	3	889.22	3	892.15	0.33	79.69
rc205	3	1321.75	**3**	**1277.60**	6	-3.34	3	1321.75	4	1158.72	-	13.12
rc206	3	1190.75	3	1221.07	4	2.55	3	1191.13	3	1223.10	2.68	12.86
rc207	3	995.52	3	1001.33	4	0.58	3	995.52	3	1003.01	0.75	32.44
rc208	3	837.82	3	841.34	5	0.42	3	838.03	3	844.23	0.74	44.24
#Better			17						10			
#Better or Same			30						20			

The results in Table 5.1 show that ALNS performs better in type 2 problems where the time windows are wider and vehicle capacities are larger. In these problems, the number of routes is small and each vehicle visits many customers along its route. In addition, the run times of type 1 problems are significantly shorter. These problems involve narrow time windows and the feasible region is smaller; hence, converging to a solution is faster. However, ALNS fails to reach the number of vehicles found by VNS/TS in many instances, in particular in r1 problems. Overall, ALNS improves the BKS of 17 problems. In these problems, the number of vehicles is usually same as in Schneider et al. (2014) and the improvement is in the distance. Better solutions are shown as bold and underlined while the same solutions are shown as bold in the table. Schneider et al. (2014) did not give any details of the computational effort and only reported an overall average run time of 15.34 minutes on an Intel Core i5 processor with 2.67 GHz speed and 4 GB RAM, operating Windows 7 Professional. The computation time of ALNS is 18 minutes on the average.

Table 5.1 also shows that on the average each vehicle visits a station for recharging. In addition, recharging is more frequent in r- and rc- type problems compared to the c-type problems where the average number of recharges is less than 1 (0.75 in c1 and 0.97 in c2 problems). There is no significant difference between type-1 and type-2 problems in terms of the number of recharges; nevertheless, the recharges are slightly more frequent in type-2 problems where the time windows are wider.

5.3.1.2. Analysis of the ALNS Algorithms

In this section, we investigate the utilization of the removal and insertion algorithms. We record the number of times that the algorithms are selected in a run and take the average of 25 runs for 56 large instances. This average value indicates the number of times that an algorithm is selected throughout the search. Figure 5.1 shows the percentage usage of customer removal algorithms compared to each other.

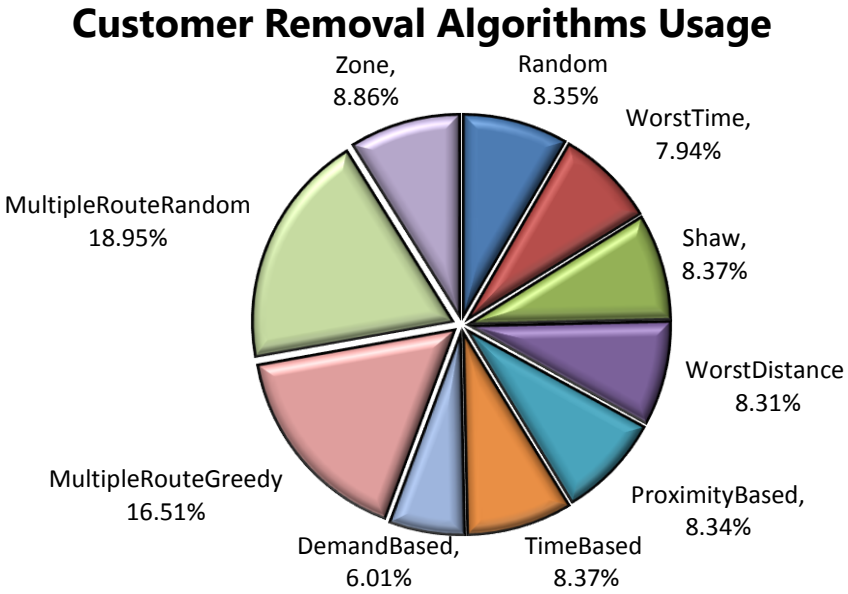

Figure 5.1: Average usage of customer removal algorithms

We see that route removal algorithms are used much more than others. There may be two reasons of this situation. Firstly, only route removal algorithms are used during the route minimization procedure which is performed for 4 times (4,000 iterations) throughout the search. Secondly, route removal algorithms are more likely to contribute decreasing the number of vehicles. Hence, they are awarded more than others meaning that their scores will be higher and their selection probabilities will be greater. On the other hand, demand based removal is used least among removal algorithms.

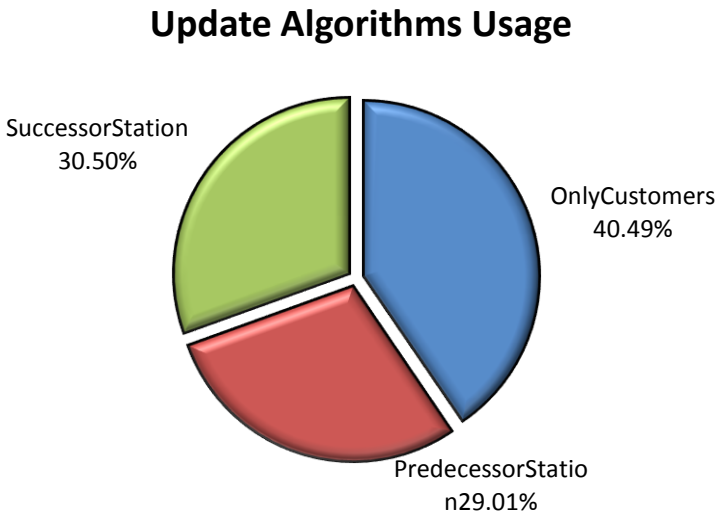

Figure 5.2: Average usage of update algorithms

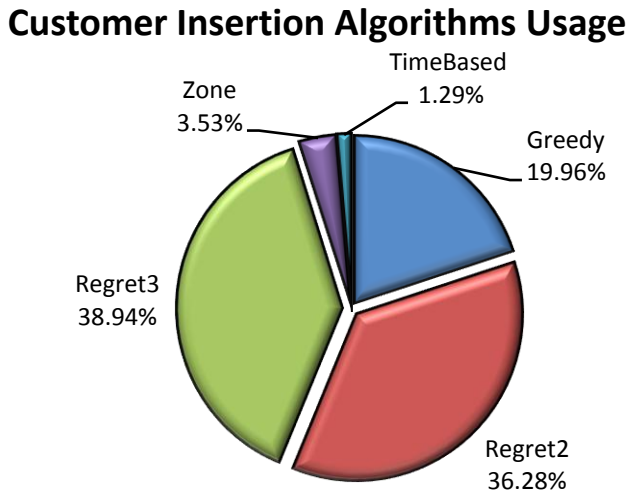

Figure 5.3: Average usage of customer insertion algorithms

Figure 5.2 shows that all three of the update algorithms are frequently utilized with ROC slightly more than the other two. According to Figure 5.3, regret2 and regret3 insertions correspond to 75% of the algorithms used for reinsertion of removed customers to the solution whereas zone and time based insertions are rarely utilized.

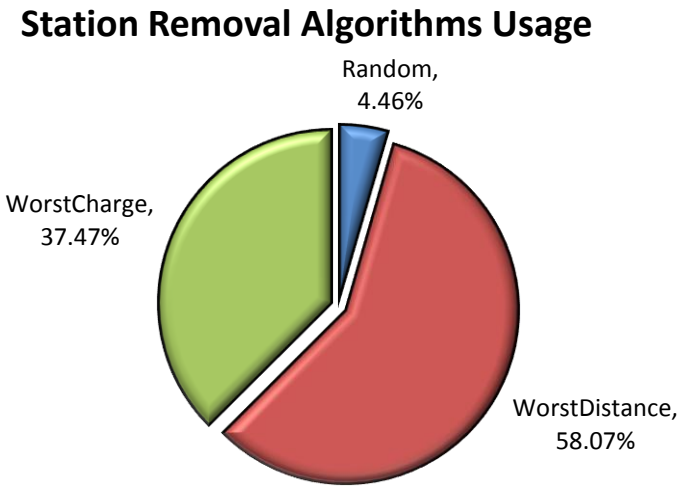

Figure 5.4: Average usage of station removal algorithms

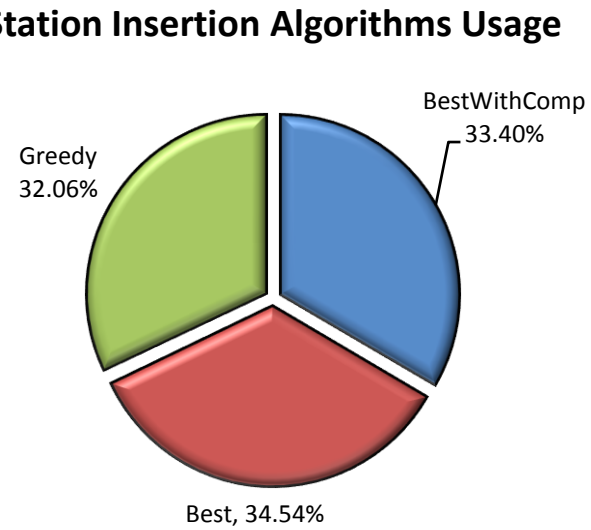

Figure 5.5: Average usage of station insertion algorithms

The results regarding the station removal usage statistics reported in Figure 5.4 show that the worst distance station removal algorithm is most commonly used with a 58% share whereas the random removal is not very effective (less than 5% utilization). On the other hand, Figure 5.5 illustrates that all three station insertion algorithms are equally used throughout the search.

5.3.2. Distance Minimization Case

5.3.2.1. Numerical Results

Table 5.2 shows the results of proposed ALNS for each instance. The notation is similar to that used in Table 5.1. ALNS (Dist) refers to the ALNS implementation minimizing the distance traveled. As in the previous case, we performed 25 runs with fixed parameters. We also provide the best results of hierarchical objective case for comparison.

The values in Table 5.2 show that the total distances are usually shorter than the distances observed in the hierarchical objective case while the numbers of vehicles used are greater (see the values in bold). This is an expected result considering the objective functions addressed in the two ALNS implementations and is an indicator of their effectiveness. On the other hand, in the four instances (see the values underlined) we observe that the distance minimizing ALNS found solution with longer distance and greater number of vehicles. This is an unexpected result which, we think, shows that a solution with a shorter travel distance is only achievable if the number of vehicles is reduced. This phenomenon also differentiates the E-VRPTW from the VRPTW. In addition, we see that the numbers of vehicles used are significantly more in the r2 and rc2 problem sets as expected due to wider time windows.

Table 5.2: ALNS results of distance minimization objective

Problem	Best in All Computational Tests				Best with Fixed Parameters			
	ALNS (Dist)		ALNS (Hier)		ALNS (Dist)		ALNS (Hier)	
	#Veh	TD	#Veh	TD	#Veh	TD	#Veh	TD
c101	12	1053.83	12	1053.83	12	1053.83	12	1053.83
c102	12	1022.58	11	1056.12	**12**	**1022.58**	11	1056.12
c103	11	1001.81	11	1001.81	11	1002.60	11	1002.60
c104	10	951.57	10	951.57	11	969.46	11	969.46
c105	12	1033.93	11	1075.37	**12**	**1033.93**	11	1080.85
c106	12	1027.25	11	1057.65	**12**	**1027.25**	11	1057.65
c107	12	1025.63	11	1031.56	**12**	**1025.63**	11	1031.56
c108	11	1015.68	11	1015.68	11	1019.45	11	1015.68
c109	11	993.77	11	993.77	**11**	**1000.75**	11	1004.36
c201	4	645.16	4	645.16	4	645.16	4	645.16
c202	4	645.16	4	645.16	4	645.16	4	645.16
c203	4	644.98	4	644.98	4	644.98	4	644.98
c204	4	636.43	4	636.43	4	636.43	4	636.43
c205	4	641.13	4	641.13	4	641.13	4	641.13
c206	4	638.17	4	638.17	4	638.17	4	638.17
c207	4	638.17	4	638.17	4	638.17	4	638.17
c208	4	638.17	4	638.17	4	638.17	4	638.17
r101	20	1646.07	18	1679.06	**20**	**1657.92**	19	1659.47
r102	19	1466.94	16	1505.53	**19**	**1466.94**	17	1480.10
r103	14	1266.45	13	1320.65	15	1270.21	14	1269.20
r104	12	1071.89	12	1071.89	12	1073.19	12	1073.75
r105	15	1383.29	15	1383.29	**17**	**1394.63**	15	1428.10
r106	14	1276.33	14	1276.33	15	1290.81	14	1276.33
r107	12	1148.43	12	1148.43	13	1160.75	12	1148.62
r108	11	1051.59	11	1051.59	**12**	**1053.92**	11	1067.32
r109	14	1223.17	11	1233.28	**14**	**1223.17**	13	1246.65
r110	12	1097.89	12	1097.89	12	1108.43	12	1104.72
r111	12	1109.14	12	1109.14	12	1122.76	12	1111.86
r112	11	1038.74	11	1038.74	12	1045.42	12	1045.42
r201	7	1100.27	3	1265.67	**7**	**1105.14**	3	1325.90
r202	6	994.35	3	1052.32	**6**	**994.35**	3	1055.48
r203	5	864.32	3	895.54	**5**	**864.32**	3	895.54
r204	3	720.82	2	780.98	3	720.82	3	720.51
r205	6	950.45	3	987.36	**4**	**955.17**	3	987.36
r206	5	896.61	3	922.70	**5**	**896.61**	3	925.37
r207	4	800.48	2	850.80	**4**	**800.48**	2	851.75
r208	3	706.81	2	736.12	**3**	**706.81**	2	736.12
r209	4	856.13	3	871.22	**4**	**856.13**	3	876.54
r210	5	833.08	3	843.65	**5**	**833.08**	3	846.96
r211	3	761.56	3	761.56	4	765.60	3	761.56
rc101	17	1730.26	16	1731.07	**17**	**1733.61**	16	1757.09
rc102	16	1551.61	15	1551.69	**16**	**1551.61**	15	1552.58
rc103	13	1351.43	13	1351.43	**14**	**1353.68**	13	1365.91
rc104	12	1227.05	11	1232.45	12	1232.91	12	1232.91
rc105	14	1473.24	14	1473.24	**15**	**1493.03**	14	1499.42
rc106	14	1414.99	14	1414.99	**14**	**1423.27**	14	1425.64
rc107	12	1283.05	12	1283.05	**12**	**1300.10**	12	1304.89
rc108	12	1208.31	11	1209.11	**12**	**1208.31**	12	1231.89
rc201	9	1257.83	4	1446.84	**9**	**1257.83**	4	1453.87
rc202	7	1142.15	3	1450.34	**7**	**1142.15**	4	1243.55
rc203	6	956.78	3	1069.27	**6**	**956.78**	3	1082.04
rc204	5	829.72	3	887.76	**5**	**829.72**	3	892.15
rc205	6	1071.62	3	1277.60	**6**	**1071.62**	4	1158.72
rc206	6	1073.33	3	1221.07	**6**	**1073.33**	3	1223.10
rc207	6	928.52	3	1001.33	**6**	**928.52**	3	1003.01
rc208	5	799.75	3	841.34	**5**	**799.75**	3	844.23

5.3.2.1. Analysis of the Algorithms

Figure 5.6 shows that the Proximity based, Shaw, random, time based and worst time removal algorithms are the most preferred customer removal algorithms while multiple route removal algorithms are the least chosen algorithms by the search. Different from the hierarchical objective case, usage of the multiple route removal algorithms are not frequently utilized in the distance minimization case as the reduction of the number of vehicles is not the primary objective. Since the other figures show similar behavior to the ones illustrated in the previous sections, we omitted them here.

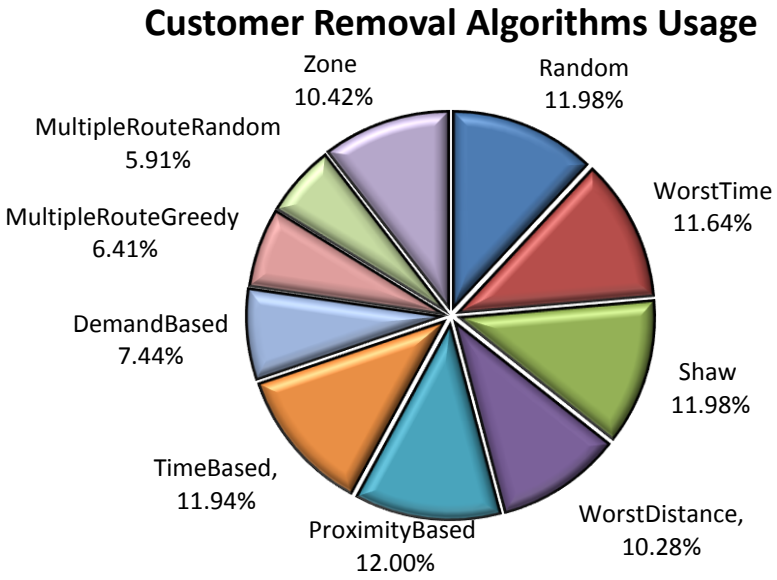

Figure 5.6: Customer Removal Algorithms Usage

Chapter 6

Conclusion and Future Research

In this thesis, we proposed an ALNS framework for solving E-VRPTW. We addressed the problem using both the hierarchical and distance minimization objectives. Some of the existing mechanisms are adopted from the literature whereas new mechanisms specific to E-VRPTW were developed to handle the visits to recharging stations. Furthermore, we proposed new mechanisms for customer removal and insertion. The general framework of the ALNS is same for both objective cases. However, new procedures were attempted to decrease the number of vehicles.

We used the instances generated by Schneider et al. (2014) to validate the performance of the proposed ALNS. We first solve the small instances by CPLEX. For hierarchical objective, we cannot obtain the optimal solutions of some instances in 7200 seconds. For those we make the comparison with the best integer results of 7200 seconds. Then we observe that our algorithm is also able to find the optimal solutions for distance minimization objective. For hierarchical objective, we obtained all optimal and best integer solutions of 7200 seconds with ALNS. For large instances, we benchmarked our results with those of Schneider et al. (2014) and reported new best known solutions in 19 instances. Since the results for the distance minimization objective are not comparable, we reported our results as benchmarks for future studies.

In this study, we assumed that the battery of the vehicle is fully charged at the recharging station. This assumption might be unnecessarily restrictive in real-world. For instance, when the vehicle visits a station near the end of its route, full charge may not be needed for the vehicle to return to the depot. A similar situation may exist between two recharges. Saving from recharging time may allow the vehicle to catch the time

window of otherwise unvisited customer, thus, may improve the solution. So, further research on this topic may focus on considering different recharging schemes such as quick charge, medium charge, full charge options as well as allowing the variable recharge, i.e. recharge as you need. The latter case is more general; however, solving the new problem may be significantly more difficult as it will involve determining the charge amount at each station as well.

Bibliography

Artmeier, A., Haselmayr, J., Leucker, M. and Sachenbacher, M. (2010) The optimal routing problem in the context of battery-powered electric vehicles, *2nd International Workshop on Constraint Reasoning and Optimization for Computational Sustainability*, Bologna, Italy.

Bramel, J., Simchi-Levi, D., (1995) A location based heuristic for general routing problems. *Operations Research*, 43(4): 649-660.

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

Chan, C. C., (2002) The state of the art of electric and hybrid vehicles, *Proceedings of the IEEE,* 90(2), *February 2002*: 247-275.

Chen, J. F., Wu, T. H., (2006) Vehicle routing problem with simultaneous deliveries and pickups. *The Journal of the Operational Research Society*, 57(5): 579-587.

Conrad, R. G. and Figliozzi, M. A. (2011). The recharging vehicle routing problem. In: Doolen, T. and Van Aken, E. (eds.) *Proceedings of the 2011 Industrial Engineering Research Conference.*

Demir, E., Bektaş, T., Laporte, G. (2012) An adaptive large neighborhood search heuristic for the pollution-routing problem. *European Journal of Operational Research* 223(2): 346-359.

Emeç, U. (2013) *Vehicle routing problem with vendor selection, intermediate pick-ups and deliveries*. (Master's thesis).

Erdogan, S. and Miller-Hooks, E. (2012). A green vehicle routing problem, *Transportation Research Part E: Logistics and Transportation Review* 48(1): 100–114.

Ioannou, M., Kritikos, M., Prastacos, G. (2001). A greedy look-ahead heuristic for the vehicle routing problem with time windows, *The Journal of the Operational Research Society* 52(5): 523-537.

Laporte, G., Gendreau, M., Potvin, J. Y., Semet, F. (2000) Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research* 7: 285-300.

Muller, L.F. (2009) An adaptive large neighborhood search algorithm for the resource-constrained project scheduling problem. *Proceedings of the VIII Metaheuristics International Conference* (MIC) *2009* (Hamburg, Germany).

Muller, L.F., Spoorendonk, S., Pisinger, D. (2012) A hybrid adaptive large neighborhood search heuristic for lot-sizing with setup times. *European Journal of Operational Research* 218(3): 614-623.

Omidvar, A. and Tavakkoli-Moghaddam, R. (2012). Sustainable vehicle routing: Strategies for congestion management and refueling scheduling. In *IEEE International Energy Conference and Exhibition, Florence,* 1089–1094

Pisinger, D., Ropke, S. (2007) A general heuristic for vehicle routing problems. *Computers & Operations Research* 34(8): 2403-2435.

Pisinger, D., Ropke, S. (2010) Large neighborhood search. In: Gendreau M, Potvin J-Y, (eds.) *Handbook of Metaheuristics* 146: 399-419.

Ribeiro, G.M., Laporte, G. (2012) An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research* 39(3): 728-735.

Ropke, S., Pisinger, D. (2006a) An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science* 40(4): 455-472.

Ropke, S., Pisinger, D. (2006b) A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research* 171(3): 750-775.

Schneider, M. Stenger, A., and Goeke, D. (2014). The electric vehicle routing problem with time windows and recharging stations. *Transportation Science* (to appear).

Shaw, P. (1998) Using constraint programming and local search methods to solve vehicle routing problems. *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, Springer, New York: 417-431.

Solomon, M. M., (1987) Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2): 254-265.

Toth, P., Vigo, D. (2002) An overview of vehicle routing problems. In: Toth P, Vigo D, (eds.) *The Vehicle Routing Problem*, SIAM, Philadelphia: 1-26.

Wang, H., Shen, J. (2007) Heuristic approaches for solving transit vehicle scheduling problem with route and fueling time constraints. *Applied Mathematics and Computation* 190: 1237-1249

Wang, H. and Cheu, R. L. (2012). Operations of a taxi fleet for advance reservations using electric vehicles and charging stations, *Journal of the Transportation Research Board* 2352: 1-10

Worley, O., Klabjan, D. (2012) Simultaneous vehicle routing and charging station siting for commercial electric vehicles. *In IEEE International Electric Vehicle Conference, Greenville, SC*: 1-3.

Appendix A: Parameter tuning details

In this part, we provide tuning sequence of the parameters which are used in the proposed ALNS algorithms.

For the hierarchical objective (distance minimization) case, for each parameter value, we take the average of the number of routes (total distances) of 10 runs for each instance which are selected for parameter tuning. Then we take the average of those instance specific values and determine the deviation from the best solution for that parameter value. The value which has the lowest deviation is selected and fixed.

The first column in Table A.1 (Table A.2) shows the parameters tuned in the hierarchical objective (distance minimization) case. The second column gives the initial value of the parameter and the corresponding deviation whereas the following columns provide the range of parameter values and the observed deviations. The sequence from top to bottom in the tables is the tuning sequence.

Table A.1: Parameter tuning details for hierarchical objective case

Parameters		Parameter Settings and Corresponding Deviations for Hierarchical Objective Case									
σ_2	Value	**6**	0	2	4	9	12	14	16	18	20
	Deviation	**0.33**	0.42	0.42	0.43	0.40	0.42	0.42	0.37	0.35	0.42
N_{RW1}	Value	300	50	100	150	200	250	350	400	450	**500**
	Deviation	0.38	0.55	0.52	0.40	0.40	0.37	0.38	0.38	0.38	**0.32**
ρ	Value	0.35	0.05	0.1	0.15	**0.2**	0.25	0.3	0.4	0.45	0.5
	Deviation	0.60	0.55	0.62	0.60	**0.50**	0.53	0.58	0.53	0.57	0.58
σ_1	Value	**33**	5	10	20	25	30	35	40	45	50
	Deviation	**0.50**	0.57	0.50	0.55	0.53	0.58	0.58	0.57	0.53	0.53
σ_3	Value	3	6	9	12	13	15	**21**	24	27	30
	Deviation	0.53	0.55	0.55	0.57	0.58	0.55	**0.50**	0.62	0.55	0.57
ϕ_1	Value	0.5	1	3	**5**	7	9	11	13	15	
	Deviation	0.55	0.53	0.58	**0.50**	0.60	0.57	0.52	0.52	0.65	
ϕ_2	Value	9	0.25	**1**	3	5	7	11	13	15	
	Deviation	0.55	0.48	**0.48**	0.55	0.55	0.53	0.55	0.62	0.55	
ϕ_3	Value	11	0.15	1	3	5	7	9	**13**	15	
	Deviation	0.52	0.57	0.50	0.50	0.53	0.55	0.50	**0.48**	0.53	
ϕ_4	Value	8	**0.25**	1	2	3	4	5	6	7	9
	Deviation	0.53	**0.52**	0.58	0.62	0.60	0.52	0.55	0.57	0.52	0.55
ε	Value	0.9996	0.999	0.9991	0.9992	0.9993	0.9994	**0.9995**	0.9997	0.9998	0.9999
	Deviation	0.53	0.52	0.55	0.50	0.57	0.55	**0.48**	0.52	0.55	0.58
μ	Value	0.4	**0.05**	0.1	0.15	0.2	0.25	0.3	0.35	0.45	0.5
	Deviation	0.58	**0.50**	0.52	0.60	0.52	0.58	0.62	0.60	0.63	0.63
κ	Value	6	**1**	2	3	4	5				
	Deviation	0.58	**0.48**	0.53	0.53	0.53	0.50				
η	Value	12	2	4	6	8	**10**				
	Deviation	0.55	0.57	0.57	0.58	0.53	**0.48**				
n_Z	Value	15	5	7	9	11	13	19	**21**	25	30
	Deviation	0.52	0.60	0.55	0.58	0.57	0.60	0.53	**0.52**	0.57	0.60
N_{SR}	Value	50	**10**	20	30	40	60	70	80	90	100
	Deviation	0.58	**0.52**	0.62	0.55	0.55	0.52	0.58	0.57	0.58	0.62
N_{RW2}	Value	3500	**1000**	1500	2000	2500	3000	4000	4500	5000	5500
	Deviation	0.58	**0.50**	0.58	0.60	0.53	0.60	0.60	0.62	0.52	0.53
m_r	Value	**0.4**	0.3	0.5	0.6						
	Deviation	**0.50**	0.55	0.52	0.57						
N_{RR}	Value	5000	2000	2500	3000	3500	4000	4500	5500	**6000**	6500
	Deviation	0.60	0.55	0.63	0.52	0.58	0.58	0.58	0.57	**0.52**	0.55
τ	Value	**1000**	750	1250	1500	1750	2000	2250	2500	2750	3000
	Deviation	**0.47**	0.58	0.52	0.50	0.50	0.55	0.53	0.55	0.57	0.58

Table A.2: Parameter tuning details for distance minimization case

Parameter		Parameter Settings and Corresponding Deviations for Distance Minimization Case									
σ_2	Value	14	0	2	4	**6**	9	12	16	18	20
	Deviation	20.50	26.45	24.31	23.77	**19.75**	20.77	20.35	21.62	20.03	24.63
N_{RW1}	Value	50	100	150	200	250	**300**	350	400	450	500
	Deviation	21.04	21.78	22.40	22.52	25.45	**20.85**	23.01	24.41	22.39	26.59
ρ	Value	0.25	0.05	0.1	0.15	0.2	0.3	**0.35**	0.4	0.45	0.5
	Deviation	23.22	26.93	26.52	25.85	25.73	24.83	**21.99**	22.09	24.51	24.18
σ_1	Value	30	5	10	20	25	**33**	35	40	45	50
	Deviation	18.25	18.71	20.53	22.02	21.31	**17.24**	17.70	23.16	19.69	21.77
σ_3	Value	21	**3**	6	9	12	13	15	24	27	30
	Deviation	25.22	**19.53**	20.12	23.06	21.90	21.27	21.70	23.74	20.88	24.01
ϕ_1	Value	**0.5**	1	3	5	7	9	11	13	15	
	Deviation	**18.68**	21.25	21.78	25.65	20.45	21.60	24.45	21.45	22.22	
ϕ_2	Value	3	0.25	1	5	7	**9**	11	13	15	
	Deviation	21.24	20.63	19.57	20.64	19.56	**17.63**	20.24	20.06	23.09	
ϕ_3	Value	13	0.15	1	3	5	7	9	**11**	15	
	Deviation	23.51	26.09	23.41	25.54	24.50	22.59	23.78	**22.38**	23.41	
ϕ_4	Value	**8**	0.25	1	2	3	4	5	6	7	9
	Deviation	**20.65**	21.73	20.72	21.12	21.60	21.07	21.03	23.06	23.06	21.07
ε	Value	0.9997	0.999	0.9991	0.9992	0.9993	0.9994	0.9995	**0.9996**	0.9998	0.9999
	Deviation	20.03	29.72	29.79	27.70	24.66	23.27	22.37	**19.87**	29.20	127.01
μ	Value	0.1	0.05	0.15	0.2	0.25	0.3	0.35	**0.4**	0.45	0.5
	Deviation	24.21	20.25	23.58	21.19	21.24	22.21	23.53	**19.26**	20.38	23.11
κ	Value	**6**	1	2	3	4	5				
	Deviation	**21.46**	22.84	23.51	23.46	22.22	21.74				
η	Value	10	2	4	6	8	**12**				
	Deviation	23.42	22.60	21.59	20.82	21.20	**19.51**				
n_Z	Value	9	5	7	11	13	**15**	19	21	25	30
	Deviation	24.81	23.39	23.82	23.79	23.79	**21.09**	22.90	22.44	23.41	22.12
N_{SR}	Value	20	10	30	40	**50**	60	70	80	90	100
	Deviation	21.98	26.21	24.21	25.17	**20.22**	23.98	23.13	20.92	22.80	22.99
N_{RW2}	Value	4500	1000	1500	2000	2500	3000	**3500**	4000	5000	5500
	Deviation	25.14	25.00	23.90	23.99	22.59	24.92	**21.47**	22.88	23.97	25.80
m_r	Value	**0.4**	0.3	0.5	0.6						
	Deviation	**20.73**	23.59	20.76	22.85						

Appendix B: Optimal Solutions of Small Instances of Schneider et al. (2014)

In Table B.1, results of 25 runs for small instances are given for distance minimization case. "#Veh" and "TD" denote the number of vehicles and total distance traveled, respectively.

Table B.1: Results for small instances

Instance	#Veh	TD	Instance	#Veh	TD	Instance	#Veh	TD
c101-5	3	247.15	**c101-10**	3	393.76	**c103-15**	4	371.7
c103-5	2	165.67	**c104-10**	2	273.93	**c106-15**	3	275.13
c206-5	2	236.58	**c202-10**	2	243.2	**c202-15**	3	376.79
c208-5	1	158.48	**c205-10**	2	228.28	**c208-15**	2	300.55
r104-5	2	136.69	**r102-10**	3	249.19	**r102-15**	5	413.93
r105-5	2	156.08	**r103-10**	3	202.85	**r105-15**	4	336.15
r202-5	1	128.78	**r201-10**	3	217.67	**r202-15**	2	358.00
r203-5	1	179.06	**r203-10**	1	218.21	**r209-15**	2	293.20
rc105-5	3	238.05	**rc102-10**	4	423.51	**rc103-15**	4	397.67
rc108-5	2	253.93	**rc108-10**	3	345.92	**rc108-15**	3	370.25
rc204-5	1	176.39	**rc201-10**	3	310.06	**rc202-15**	2	394.39
rc208-5	1	167.98	**rc205-10**	2	325.98	**rc204-15**	2	310.58

Appendix C: The generic structure of the ALNS algorithm for distance minimization case

Algorithm 8: ALNS algorithm for distance minimization

input: $A_{CR}, A_{CI}, A_{SR}, A_{SI}, N_I, N_{RW1}, N_{RW2}, \mu, \varepsilon$

output: The best solution S_{best}

Generate an initial solution S_i by using Greedy insertion algorithm
Initialize selection probabilities $p^s_{cr_a}, p^s_{ci_a}, p^s_{sr_a}, p^s_{si_a}, p^s_{u_a}$ for each $cr_a \in A_{CR}$, $ci_a \in A_{CI}$, $sr_a \in A_{SR}$, $si_a \in A_{SI}$, $u_a \in A_U$
Initialize T and T_{start} by using $f(S_i)$ and μ
Let j be the outermost iteration counter initialized as $j \leftarrow 1$
Let $S_C \leftarrow S_B \leftarrow S_i$
while $j \leq N_I$ ***do***
Select a customer removal algorithm $cr_a \in A_{CR}$ with probability $p^s_{cr_n}$
Select an update mechanism $u_a \in A_U$ with probability $p^s_{u_n}$
Generate S_1 by applying cr_a and u_a to S_C
Select a customer insertion algorithm $ci_a \in A_{CI}$ with probability $p^s_{ci_a}$
Generate S_2 by applying ci_a to S_1
if $f(S_2) < f(S_C)$ ***then***
    $S_C \leftarrow S_1$
else
    Let $p = e^{\frac{-(f(S_T)-f(S_C))}{T}}$
    Generate a random number $\vartheta \in [0,1]$
    if $\vartheta < p$ ***then***
        $S_C \leftarrow S_1$
if $f(S_C) < f(S_B)$ ***then***
    $S_B \leftarrow S_C$
if $j \equiv 0 \ (mod\ N_{RW1})$ ***then***
    Update the probabilities $p^s_{cr_a}, p^s_{ci_a}, p^s_{u_a}$ using the adaptive weight procedure
Update the current temperature as $T \leftarrow T.\varepsilon$
Increase the counter as $j \leftarrow j + 1$

25 ***if*** $j \equiv 0\ (mod\ N_{SR})$ ***then***
26 Select a station removal algorithm $sr_a \in A_{SR}$ with probability $p^s_{sr_a}$
27 Generate S_3 by applying sr_a to S_2
28 ***if*** S_3 is infeasible ***then***
29 Select a station insertion algorithm $si_a \in A_{SI}$ with probability $p^s_{si_a}$
30 Generate S_4 by applying si_a to S_3
31 Recall the steps $\mathbf{12-20}$ with S_4 instead of S_1
32 ***if*** $j \equiv 0\ (mod\ N_{RW2})$ ***then***
33 Update the probabilities $p^s_{sr_a}, p^s_{si_a}$ using the adaptive weight procedure
34 Recall steps $\mathbf{23-24}$
35 ***end while***
36 ***Return*** $\boldsymbol{S_B}$