University of Arkansas, Fayetteville
ScholarWorks@UARK

Theses and Dissertations

5-2018

Essays on Applications of Transportation Network Design and Optimization

Amirali Ghaharikermani
University of Arkansas, Fayetteville

Follow this and additional works at: http://scholarworks.uark.edu/etd

Part of the Industrial Engineering Commons, Operational Research Commons, and the Transportation Engineering Commons

Recommended Citation

Ghaharikermani, Amirali, "Essays on Applications of Transportation Network Design and Optimization" (2018). *Theses and Dissertations*. 2745.
http://scholarworks.uark.edu/etd/2745

This Dissertation is brought to you for free and open access by ScholarWorks@UARK. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, ccmiddle@uark.edu.

Essays on Applications of Transportation Network Design and Optimization

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy in Engineering

by

Amirali Ghaharikermani
Sharif University of Technology
Bachelor of Science in Industrial Engineering, 2010
Sharif University of Technology
Master of Science in Socio-Economic System Engineering, 2012

May 2018
University of Arkansas

This dissertation is approved for recommendation to the Graduate Council.

____
Edward Pohl, Ph.D.
Dissertation Director

____
Manuel Rossetti, Ph.D.
Committee Member

____
Chase Rainwater, Ph.D.
Committee Member

____
Marc Scott, Ph.D.
Committee Member

Abstract

In this dissertation we address different transportation problems. The three main outcomes are: designing a battery swap station network, studying gaps in Empty Container Management literature, designing a model with similar characteristics to the vehicle routing problem as well as implementation scenario for it to be applied for a real-world case.

For the designed battery swap station, a model is developed for customer demand satisfaction that permits construction of different types of BSS in the planning network. Our solution methodology is a Tabu Search algorithm combined with a dynamic programming initialization. Numerous tests showed that the proposed TS approach provides improvement compared to CPLEX both in terms of time and solution quality.

The comprehensive literature review on Empty Container Management resulted in realizing intermodal environment as problem areas not being investigated much. Moreover, limited number of research has targeted large problem instances. The outcome of this investigation was to develop a demand forecasting model over the empty containers, a DSS for daily decision making, and a mathematical model to optimize the problem. Case studies with large instance of even more than 500 nodes as well as customization of the approach for a major transportation company in the US proves the applicability of the outcomes.

The final outcome of the presenting dissertation aims at implication of a novel process for TB diagnostic process. For this reason, a simulation model is developed that quantifies the impact of different scenarios which are feasible options to be executed. However, our investigation resulted that among different scenarios, TBOnDemand, in which patients request a vehicle to take their sample at their location, provides significant improvements. We developed a model with similar characteristics to vehicle routing problem with predetermined, but dynamic sample processing capacity allocated to each vehicle. Results of the solutions gained by CPLEX an offline assignment of vehicles to patients showed substantial reduction in serving time.

As stated above, the contribution of presenting dissertation includes but is not limited to intro-

ducing novel mathematical models, developing heuristic approaches, simulating uninvestigated problem area with largescale samples, constructing DSS tool with customized case studies.

Acknowledgement

I would like to express appreciation to my advisor Dr. Edward Pohl, who convincingly conveyed a spirit of adventure regarding research and teaching. This dissertation would not have been complete successfully without his guide. I would also like to thank my committee members, Dr. Chase Rainwater, Dr. Manuel Rossetti, and Dr. Marc Scott, whose works and suggestions demonstrated to me that concern for global problems supported by related literature and technology, should always transcend academia. In addition, a thank you to our industry partner who introduced actual restrictions and requirements that helped this research to fit into applicable real-world cases. I thank the University of Arkansas faculty and staff who in one way or another supported me during the academic years that resulted in this research. Last but not least, I appreciate my family and friends who made my journey valuable and meaningful.

Dedication

I would like to dedicate my work to my family who continuously supported me for following my dreams. They have always been the most encouraging members of my life.

Contents

List of Tables

List of Figures

1 Introduction 1

2 DESIGNING A TRANSPORTATION NETWORK FOR A UAV DELIVERY SERVICE 3

2.1 Introduction 3

2.2 Literature Review 4

2.3 Problem Description 6

2.3.1 Assumptions 8

2.3.2 Notations 8

2.3.3 Mathematical Model 10

2.3.4 Valid inequality 11

2.4 Solution methodology 11

2.4.1 Tabu Search 12

2.4.1.1 Current Solution 13

2.4.1.2 Operators 13

2.4.1.3 Performance Measures 13

2.4.1.4 Objective function evaluation 16

2.4.1.5 Initialization 20

2.4.1.6 Tabu Search steps 23

2.5 Computational Result 24

2.5.1 Characteristics of Instances 24

2.5.2 Results 26

2.6 Conclusion and future study 31

3 Empty Container Management in an intermodal Network 32

3.1 Intermodal Freight Networks 33

3.1.1 Comparison to Over-the-Road Networks 34

3.1.2 Explanation of Typical Problems that Arise 35

3.1.3 Empty Container Management 35

3.2 Empty Container Management Literature Review 36

3.2.1 Maritime Transportation 36

3.2.2 Inland Transportation 41

3.3 Minimum Cost Network Flow Problems 47

3.4 Empty Container Management Model 48

3.4.1 Specific Problem Instance 48

3.4.2 Modeling Approach 50

3.4.3 Data Analysis 52

3.4.3.1 Simulation 53

3.4.3.1.1 Scheduling a Move 53

3.4.3.1.2 Initializing a Move 54

3.4.3.1.3 Container Arrival 55

3.4.3.2 Simulation Result 57

3.4.3.2.1 Move Analysis 57

3.4.3.2.2 Containers Analysis 58

3.4.4 Forecasting Model 58

3.4.5 Decision Support System 65

3.4.5.1 DSS Architecture 66

3.4.6 Case Study 69

3.4.6.1 Rail Movement Only 69

3.4.6.2 Rail Movement and Truck Movement based on Zip Code 71

3.4.6.3 Rail Movement and Truck Movement with no zip codes 72

3.5 Future Work and Contribution 74

4 Designing a Supply Chain Network for TB response in high-risk infected countries 76

4.1 Introduction 76

4.2 Emergency Medical Service Literature review 77

4.2.1 Deterministic models 78

4.2.2 Stochastic models 79

4.2.3 Dynamic models 80

4.3 Simulation 81

4.3.1 Proposed Scenarios 81

4.3.1.1 Base Model 81

4.3.1.2 POC device at TB Microscopy Center 82

4.3.1.3 POC device at clinical centers 83

4.3.1.4 TB OnDemand 84

4.3.2 Simulation Model 85

4.3.2.1 Base Model 85

4.3.2.2 POC at Microscopy Center 86

4.3.2.3 POC at Clinics 87

4.3.2.4 TB On Demand 88

4.3.3 Simulation Experiment 88

4.3.3.1 Parameters 89

4.3.4 Sensitivity Analysis of Simulation's parameters 96

4.3.4.1 Warm up period analysis 96

4.3.4.2 Number of Replications 96

4.3.4.3 Incident Rate Analysis 98

4.3.4.4 Infrastructure Type 99

4.3.4.5 Doctor evaluation process 99

4.3.4.6 Geographical Location 100

4.3.4.7 Factorial Design . . . 100
4.3.5 TBOnDemand . . . 102
4.3.6 Comparison of Base Model Setting and TBOnDemand . . . 105
4.3.7 Simulation Conclusion . . . 107
4.4 Mathematical Model . . . 108
4.4.1 Notation and Parameters . . . 108
4.4.2 Model . . . 109
4.5 Optimal vs Simulation . . . 111
4.6 Conclusion and Future Research . . . 113

5 Contribution and Conclusion **115**

6 References **117**

Appendix A Chapter 2 Programming Codes **125**
A.1 Tabu Search Class . . . 125
A.2 Max Flow Solver Class . . . 131
A.3 Dynamic Programming Class . . . 133
A.4 Data Reader Class . . . 139
A.5 Demand Point Class . . . 147
A.6 BSS Station Class . . . 148
A.7 Solution Class . . . 151
A.8 CPLEX Solver Class . . . 161

Appendix B Chapter 3 Programming Codes **170**
B.1 Network Class . . . 170
B.2 Node Class . . . 171
B.3 Customer/Ramp Class . . . 173
B.4 Forcaster Class . . . 174

B.5 Math Model Code . . . 176
B.6 Simulation . . . 179
B.6.1 Container Class . . . 179
B.6.2 Move Class . . . 180
B.6.3 Ramp Class . . . 184
B.6.4 Control Class . . . 185

Appendix C Chapter 4 Programming Codes **190**

C.1 Care Center Class . . . 190
C.2 Clinic Class . . . 191
C.3 Microscopy Center Class . . . 194
C.4 Major Hospital Class . . . 196
C.5 Patient Class . . . 198
C.6 Vehicle Class . . . 200
C.7 Simulator Class . . . 202

List of Tables

1 Results of 10 Instances run with and without the valid inequality 12

2 Characteristics of instances 25

3 BSS attributes 26

4 Tabu search results for 300 instances 27

5 Summary of results of 300 instances based on the region size. 28

6 Result of 15 Instances 29

7 Classification of the literature of empty containers management modeling 46

8 System status since 2008 to 2015. 52

9 Sample Data for Simulation 55

10 The forecasted value against the actual value and cost associated with SS calculation for a customer. 64

11 The incident rate and infrastructure type of each region 92

12 The geographical information of 10 regions. 94

13 ADP Values in the pilot study. 97

14 Speedś bounds for each region type in the simulation model 99

15 Different Doctor Visit bounds 99

16 Parameters List for simulation model 102

17 Setting description for simulation experiment 106

18 Number of devices and Result of 25 replications for each setting 107

19 Result of CPLEX runs against simulation runs 112

List of Figures

1 Small sample to show the difference of facility location and coverage problem with proposed problem. 7

2 An example of different steps of calculating the lost sale for a network. 17

3 Transformation of each arc of fixed charge network design problem to our research problem. 19

4 Example of how value function of dynamic programming works. 22

5 Example of an initial solution for Tabu Search 23

6 Final Solution from Tabu Search for an instance 30

7 Number of Empty Containers Management by Publisher 43

8 Number of Empty Containers Management Research in each domains 44

9 Empty container flows- regional and local level (adapted from [87]) 45

10 Ramps Location throughout USA, Canada and Mexico 50

11 Flow Diagram for a move in the System. 56

12 Number of loaded and empty moves 57

13 Status of containers 58

14 15 Different Configurations for forecasting model of ”Customer A” 61

15 15 Different Configurations for forecasting model of ”Customer B” 62

16 Process Flow of developed DSS. 67

17 First window of DSS before entering/uploading any input file. 67

18 Window of DSS after entering/uploading any input file. 68

19 DSS Window while it is processing. 68

20 Location of 41 Ramps. 70

21 The comparison between the historical flow of empty containers between ramps to optimal solution obtained from DSS. 71

22 Location of 41 Ramps and 110 center of zip codes. 72

23 Location of 41 Ramps and 480 customers. 73
24 Empty Container Demand Volume of each node on the system in 2014. 74
25 Current process of TB diagnostic 82
26 Current process of TB diagnostic 83
27 Current process of TB diagnostic 84
28 Simulation Process for the base model 86
29 Simulation process for POC at Microscopy Center 87
30 Simulation process for POC at Microscopy Center 87
31 Simulation process for TB On Demand 88
32 Ghana's map with its different regions (adapted from [38]) 89
33 Ghana's map with its different regions (adapted from [38]) 91
34 Poverty Level of Ghana's districts (adapted from [38]) 92
35 Microscopy centers locations in Ghana 95
36 Approximation of Country Representation 95
37 Welchś Plot based on 5 runs for Average Diagnostic Period (ADP) 97
38 An example of different facilities' locations in the simulation model 100
39 Different Plots on the Residuals 101
40 The Result of Minitab Software for full factorial design of 4 factors and 3 levels. . 101
41 The main effects plot for the 4 factors. 102
42 ADP Average over 5 replication for different number of vehicle and devices under Nearest Vehicle Policy 103
43 ADP Average over 5 replication for different number of vehicle and devices under region based Vehicle Policy 104
44 NDP Average over 5 replication for different number of vehicle and devices under Nearest Vehicle Policy 105
45 NDP Average over 5 replication for different number of vehicle and devices under region based Vehicle Policy 105

46 An example of TBOnDemand optimal solution vs traditional VRP 110

1 Introduction

This dissertation investigates and implements novel approaches in the field of transportation. Each chapter deals with a different transportation concept which is briefly outlined in the following.

In the first chapter, we investigate the use of Unmanned Aerial Vehicles (UAV) as new means of transportation in last mile delivery. Recently, the use of UAVs for delivery services has become a topic of interest and research for large commercial service providers such as Google and Amazon. The delivery speed of UAVs provides such companies with significant advantage in their market. Although utilizing the UAVs in product delivery has received tremendous excitement, there are several issues that need to be resolved prior to real world implementation. The first issue with using UAVs as a transportation mode is their limited flying range. Small UAVs are not able to fly long distances due to their limited battery life. Flight range can be improved by using battery swapping stations in the planning horizon. UAVs can change their depleted batteries at these stations and continue their flights. In our research, we attempt to develop a model to construct a network of those stations in order to enable the UAVs to fly long distances and make deliveries to different demand points. We developed a Tabu-Search heuristic to solve different instances.

In the second chapter, we investigate and develop a decision support system (DSS) for a major transportation company in order to facilitate their tactical problem. Freight transportation networks are responsible for delivering filled containers from one point to another point. These are profitable movements for transportation companies. When not every node in a network has the same number of containers going into it as it has going out of it, surpluses develop at some nodes while shortages occur at others. If companies do not address this behavior, most of the containers eventually end up at a few nodes. Network operators must perform rebalancing moves in order to allow the network to continue to operate. However, they are generating cost without any direct revenue. In this chapter, a mathematical model for minimum cost flow is formulated in terms of a company's requirement. Also, a customer dependent forecasting model is developed to manage

the uncertainty of each customer's demand.

The third chapter deals with logistic issues associated with supporting the diagnosis process of Tuberculosis (TB) in developing countries. The increased chance of mortality for drug resistant patients makes it important to diagnose TB infection in the early stage. Conventional diagnosis method normally take significant amounts of time to prepare a result for a suitable treatment solution. Infrastructure issues in the developing countries makes the supply chain process difficult. A newly developed device, GeneXpert Omni, has recently made a significant decrease in TB diagnosis time. In this chapter, we aim to model the utilize of this device in a real-world health care system. Our model will consider a fleet of devices for which the capacity, location, and routes must be determined.

2 DESIGNING A TRANSPORTATION NETWORK FOR A UAV DELIVERY SERVICE

2.1 Introduction

Recently, the application of Unmanned Aerial Vehicles (UAVs) for delivery has become a topic of increasing interest. Two well-known companies, Google and Amazon, are exploring this type of delivery mechanism as part of their newest technology innovation strategy (Google X [2] Amazon Prime Air [1]). Other applications of UAVs include disaster relief, border security, and uses in agriculture. In undeveloped countries with road deficiencies or for developed countries with congestion problems, UAVs are a qualified mechanism of transport. When a disaster occurs, the roads might become unpassable and the UAVs can become an appropriate way to deliver medicine and first aid packages.

Small UAVs can fly at low altitudes and can easily avoid obstacles at low altitudes. However, they are limited in the size of the payloads they can carry and they can only fly for short distances due to their battery capacity. For example, Micro UAVs can fly for only one hour and carrying a maximum load of 5 kilograms [10]. Long operations in terms of distance or time will require a system of UAVs and automated service stations to swap batteries making the UAVs capable of traveling longer distances. These kinds of stations currently exist and they operate automatically without the need for any labor [85].

In this chapter, the problem of locating these battery swap stations is studied. The remainder of this paper is structured as follows: Section 2 reviews the existing literature. Latter sections describe the problem, discusses the details and present a mathematical model. In the fourth and fifth sections, a heuristic algorithm is developed and the results are discussed. The chapter ends by explaining the conclusions and contributions of the research.

2.2 Literature Review

Several research studies have begun to explore the design of the service stations for UAVs. Ji and Xia [50] proposed an analytical method to minimize the number of identical automated guided vehicles (AGVs) in order to guarantee the stability of the system (stability implies that the number of waiting orders at any time should be maintained at a stable level). Also, they used a simulation model to solve the numerical instances. Godzdanker et al. [43] proposed a p-median problem to locate a fixed number of stations. They extended their work in [44] by studying the station location problem while considering the UAV flight path. Suzuki et al. [85] developed a Petri net to find the required number of UAVs and service stations to maintain a desired number of UAVs in flight. Russell and Lamont [70] studied a Genetic Vehicle Representation (GVR), a recent approach to solve instances of the Vehicle Routing Problem (VRP) using genetic algorithms. VRP focuses on finding the best route to serve multiple customers using a fleet of vehicles. Kim et al. [55] developed a mixed integer linear program (MILP) model to handle the problem of scheduling a system of UAVs with multiple shared bases in disparate geographic locations and they solved the problem using a genetic algorithm. Kim and Morrison [54] developed a mixed integer linear program to describe the joint design and scheduling of the UAV problem. The authors assumed returning flights, deterministic demand and multiple tasks conducted by a UAV. Song et al. [77] extended the work in [54] by allowing for arbitrary UAV initial locations and fuel levels over a finite horizon. They improved the heuristic method described in [54] and proposed a new heuristic algorithm to solve the problem efficiently. Lee and Morrison [57] developed a method to efficiently use UAVs for maritime search and rescue operations over a planning horizon. They formulated a mixed integer linear program to search and rescue task planning considering limited UAV fuel capacity and changing the priority of the task search. They conducted numerical examples based on CPLEX to show the performance of their model. Levy et al. [58] considered the routing problem for multiple target locations, fuel stations (or depots) and UAVs such that each target is visited at least once by a vehicle and the fuel constraint is satisfied. Two heuristic methods were developed to find reasonable solutions for large problems and tested on a set

of large instances (up to 280 targets) obtained from a standard library using simulation. Sunder et al. [84] considered routing problem for a general single Unmanned Aerial Vehicle with fuel constraints and multiple depots and the UAV is allowed to refuel at any depot. They proposed a mixed-integer linear model to find optimal solutions to the problem. An approximate algorithm and fast heuristics were developed to solve the problems in a reasonable time. He et al. [46] optimized the location of public charging stations for electric vehicles on a network of roads with the assumption of determining tour paths and recharging plans simultaneously by drivers. They formulated the problem as a bi-level mathematical program to minimize the travel and recharging time by drivers and solved it using a genetic algorithm. Then, a numerical example consisting of 24 nodes and 74 links (Sioux Fall network) is solved to demonstrate the performance of the mathematical model and heuristic method to find the location and types of charging stations. Cavadas et al. [13] proposed a mixed integer programming model to locate the electric vehicles slow-charging stations. In this problem, drivers can stop at various locations but they can only charge the EV at one of the charging stations. The travelers parking locations and their daily activities are considered in this model because tour-based network equilibrium can trace the state-of-charge of battery for each vehicle and stopping time at each destination. They applied their model to a real problem in the city of Coimbra, with nine stations comparing the optimal solution generated by the multi-period model with the solutions obtained without using optimization. Yang and Sun [92] determined the location of electric vehicles battery swap stations and vehicle routing plans simultaneously by considering the limited driving range due to the battery. They model their problem as an integer program and develop two heuristics to solve the large-scale problems. They applied these methods on three sets of small, medium, and large instances (480). The results revealed that the proposed heuristic methods can find good solutions in a reasonable time in comparison with CPLEX. Gimenez-Gaydou et al. [39] proposed a new approach to locate the battery charging stations for electric vehicles in urban areas. The approach is based on a novel location-allocation model considering the charging requirements of electric vehicles and range of charges. The proposed approach was applied on a set of random instances. By comparing the

results with the classic gradual maximal covering model, the authors found that the proposed approach had better performance when the locations of charging stations are known. Hiermann et al. [47] introduced a new approach by combining two problems, Fleet Size Mix Vehicle Routing Problem with Time Windows (FSMFTW), and Electric Vehicle Routing Problem with Time Windows and Recharging Stations (EVRPTW). They proposed a MIP model for their problem and solved smaller instances using a branch-and-price method. To solve real size problems, they developed a metaheuristic approach based on an Adaptive Large Neighborhood Search. The computational results revealed that the proposed approach could solve small instances optimally and perform well for larger problems.

2.3 Problem Description

The problem that is considered in this paper is constructing a network of battery swap stations (BSS) to support UAV delivery routes. In order to enhance the flying range of the UAVs, BSS should be located in the planning area such that a UAV can use them in order to satisfy the demand at the delivery points. It is worth noting that the difference between classic problems in the literature and the proposed research problem are in two areas. The first area of difference is within the facility location problem (FLP). Our research question is partly a facility location problem for a network of BSS. The classical FLP assumes independence among facility locations, meaning that there is no relationship between them. However, the nature of our problem requires consideration of relationships between locations because each UAV depends on several BSS in order to get to the demand location. In other word, a single BSS cannot support the UAV trip independently. The closest class of FLP to our problem, that is studied in the literature is FLP problems with a backup facility for each customer [76] but still does not consider the relationship between two facilities. The second part of our problem is a location routing problem (LRP). In this class of research, the main focus is to locate the facilities such that the cost of routing between customers is minimized [7], [56]. Similar to FLP, this category of problem does not study the routing between depots. It focuses on the vehicles starting their trips from a depot

and delivering products to customers on the route and finally going back to the depot. In our research problem UAVs rely on a network of BSS (facilities not customers) to deliver a product to a customer. As a result, unlike LRP that studies the location of one depot, our research problem requires to determine the locations of multiple battery swap stations. The only known location in the problem is the demand point. The same argument is true for the coverage problem (CP), since it does not study the relationship between facilities nor the routing between them. The only impact of two facilities on each other is whether their covered areas overlap or not. In figure 1, the difference between FLP, CP and the proposed problems are illustrated. Let $C1$, $C2$ and $C3$ be considered as potential locations, in addition let S and D be the supplier and a customer respectively. The arcs around $C1$ and $C3$ are the coverage radius. A facility location formulation would locate $C3$ in order to cover the customer and a coverage formulation would locate $C1$ and $C3$ to cover most of the area. However, for the proposed problem, all three locations should be properly located to get an UAV from the supplier to the customer.

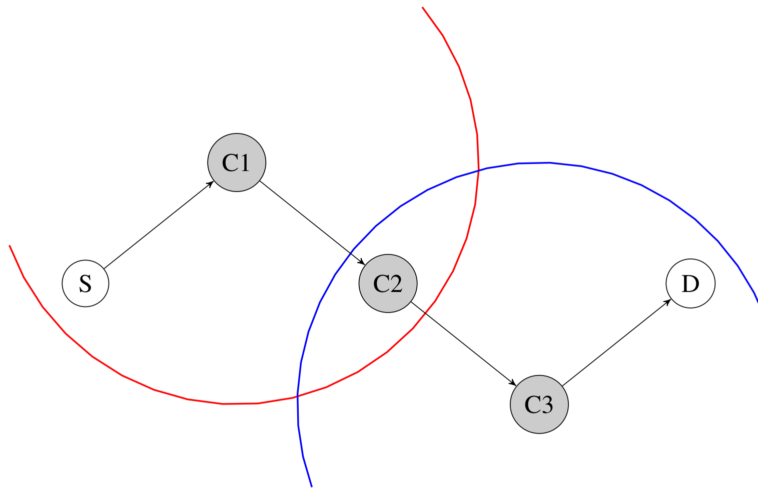


Figure 1: Small sample to show the difference of facility location and coverage problem with proposed problem.

Perhaps the most relevant literature is the electrical vehicle facility location problem. These studies mostly focus on the maximal covering problem such as [34]. The most related study is [90] which investigates how to locate the recharging stations for a commercial company. There are

several key differences between our proposed research and theirs. First, we assume capacitated BSS which means there is no queue at each station. This requires a UAV to not arrive at an occupied BSS. Second, there is a limited number of batteries available at each station. Third, in our approach we are accepting a lost sale by using a large penalty.
In this section we have reviewed the current literature and identified the key differences between this paper and the most relevant ones.

2.3.1 Assumptions

This section explains the key assumptions that are considered for developing the mathematical model. All UAVs are considered as identical and have a limited flying range which is assumed to be constant. A UAV can only carry one unit of product during a trip from a supplier to a customer. The return flight from a customer to a supplier is through the same route of the delivery flight. Any point within inside the planning area can be a positional location for BSS due to the small size of this device. It can be placed on the top of a building or in a vacant field. There can be different BSS. The main difference between them, is the number of batteries that they can hold. Different number of batteries directly impacts the construction and operational cost. Since the number of available batteries at each station reduces as it serves the UAVs, a pre-determined schedule is considered to replenish the available batteries at each BSS. This can be alternatively done by a ground unit that delivers batteries to the stations or allocating the required amount of time in order to recharge the used batteries. Each customer location has a stochastic demand with a known probability function prior to the planning. A lost sale is acceptable but there is large cost associated with it.

2.3.2 Notations

This section lists the notations used in the paper and their associated meaning.

- Sets
    - S: Set of all suppliers

- D: Set of all demand points
- B: Set of all candidate locations for battery swap stations
- A: Set of all arcs between locations
- T: Set of all time slots. At the end of each time slot, all the batteries assume to be available, charging or ground crew replenishing.
- K: Set of all types of battery swap stations
- Ω: Set of all scenarios

• Parameters

- d_i^{ω}: Demand size of demand point i in scenario ω
- b_k: Number of batteries available at battery swap stations type k
- c_l: Penalty cost for each lost sale
- c_s^k: Cost of constructing a battery swap station type k
- c_o^k: Operational cost of a battery swap station in one period type k
- p^{ω}: The probability of each scenario ω

• Decision Variables

- $X_{ij}^{\omega t}$ Number of full travels between locations i and j at time t in scenario ω
- $V_i^{\omega t}$ the total amount of satisfied demand for location i in period t in scenario ω
- $L_i^{\omega t}$ Lost sale for demand point i in period t in scenario ω
- Z_i^k 1 if we construct batteries swap station i. 0 Otherwise
- O_i^{kt} 1 if battery swap station type k is functional in period t. 0 otherwise.

2.3.3 Mathematical Model

The mathematical model of the problem using the defined notations is presented below:

$$Min \quad \sum_{i\in B,k\in K} c_s^k * Z_i^k + \sum_{i\in d,\omega\in\Omega} p^{\omega} * c_l * L_i^{\omega} + \sum_{i\in B,k\in K,t\in T} c_o^k * O_i^{kt} \tag{1}$$

Subject to

$$\sum_{j\in FS(i)} X_{ij}^{\omega t} - \sum_{j\in RS(i)} X_{ji}^{\omega t} + V_i^{\omega t} = 0, \quad \forall i \in (S\cup D\cup B), \forall t \in T, \forall \omega \in \Omega \tag{2}$$

$$\sum_{i\in D} V_i^{\omega t} + \sum_{i\in S} V_i^{\omega t} = 0, \quad \forall t \in T, \forall \omega \in \Omega \tag{3}$$

$$L_i^{\omega} + \sum_{t\in T} V_i^{\omega t} = d_i^{\omega}, \quad \forall i \in D, \forall \omega \in \Omega \tag{4}$$

$$\sum_{j\in FS(i)} X_{ij}^{\omega t} \leq \sum_{k\in K} b_k * O_i^{kt}, \quad \forall i \in B, \forall t \in T, \forall \omega \in \Omega \tag{5}$$

$$\sum_{k\in k} Z_i^k \leq 1, \quad \forall i \in B \tag{6}$$

$$\sum_{t\in T} O_i^{kt} \leq |T| * Z_i^k, \quad \forall i \in B, \forall k \in K \tag{7}$$

$$V_i^{\omega t} = 0, \quad \forall i \in B, \forall t \in T, \forall \omega \in \Omega \tag{8}$$

$$X_{ij}^{\omega t} \geq 0,\ O_i^{kt}, Z_i^k \in \{0,1\} \quad \forall i \in S, t \in T, k \in K, \forall \omega \in \Omega \tag{9}$$

The objective function (1) includes three parts; First is the construction cost function associated with the selected locations. The second term represents the operational cost of the network while the latter part is the penalty cost for the lost sales. Constraint set (2) are the flow balance constraints for suppliers, customers and BSS. Constraint set (3) assures that the amount of supplied product is equal to the demanded product at each time slot and scenario. Constraint set (4) calculates the amount of lost sales over all time slots. Constraint set (5) assures that a UAV can only use an operational BSS which has an available battery. Constraint set (6) allows the problem to only construct one type of station at each location. Constraint set (7) assures that stations can only be operational if we construct them and finally constraint set (8) does not allow the BSS to

become a demand location or supplier.

2.3.4 Valid inequality

In this section, we introduce a valid inequality in order to tighten the feasible region of the problem. This inequality is estimated to reduce the time needed to reach optimality by 35%. In order to develop this inequality, the following theorem is defined

Proposition 1 *Each selected BSS candidate in the optimal solution is adjacent to at least one other selected BSS candidate or a supplier.*

Proposition Assume the selected BSS candidate in the optimal solution is not adjacent to any other selected BSS candidate or a supplier. Thus, UAV can access it only from demand locations. Since, any flight out of the demand locations is an empty flight returning to depot by taking the same delivery route, loaded UAVs will not be able to use the selected BSS candidate on their delivery route. Therefore, we can remove this BSS candidate from the solution and obtain a better objective function value which is a contradiction to our assumption. By using this theorem we can introduce a new valid inequality to the problem in order to tighten the feasible region.

$$\sum_{k \in K} Z_i^k \leq \sum_{j \in RS(i), k \in K} Z_j^k \qquad \{i \in B | (i,s) \notin A, s \in S\} \tag{10}$$

This constraint set is defined on the BSS candidates that are not adjacent to any supplier. In order to evaluate the impact of the valid inequality, 10 random instances were generated and solved optimally. Table 1 shows the number of processed nodes when CPLEX solves the instances with the valid inequality and without it. Based on the random sample, the valid inequality on average reduce the number of nodes processed by 11%.

2.4 Solution methodology

In this section, the solution methodology for the proposed model is discussed.

Table 1: Results of 10 Instances run with and without the valid inequality

	# of Nodes Processed		
	With Valid Inequality	Without Valid Inequality	Difference
1	37	37	0%
2	91	91	0%
3	457	573	20%
4	721	722	0%
5	120	120	0%
6	4366	5245	17%
7	16461	28032	41%
8	6017	6871	12%
9	14439	16902	15%
10	147167	148891	1%

2.4.1 Tabu Search

Tabu search was first introduced by Fred W. Glover [42] as a metaheuristic search method to find a potential solution to an optimization problem and explore its neighbors in order to find an improved solution. This approach uses a memory to enhance the quality of the neighborhood search. In TS, a list is utilized to escape local optima and make the neighborhood search more efficient. At each iteration, a set of neighbors for the current solution is generated and the best neighbor in terms of objective function will be selected. If the selected neighbor does not violate the tabu list moves, it will be used in the next iteration as the current solution. The tabu list consists of all moves that we cannot make at each iteration. These moves are determined based on previous iterations. The tabu list facilitates the process of escaping the local optima . Note that in the selection process of TS, the best neighbor among the generated neighbors will selected which means it is possible to move to an inferior solution compared to the current solution. For our problem, the tabu list structure is straightforward however, tuning the tabu tenure and aspiration criterion can significantly improve the results. Since, the element of our tabu list is BSS, an acceptable ratio between the number of BSS and tabu tenure has to be determined. If the length of tabu tenure' is too long, it will avoid several candidates from entering as new solutions. On the other hand, if it is short, the tabu list will not be effective in finding new areas for good solutions.

2.4.1.1 Current Solution

The structure of the solution for the proposed algorithm consists of three components. First, which BSS candidates are selected to be constructed. Second, which types of BSS will be constructed at the selected location and the last one is at which time slot the stations are functioning. In order to incorporate all three components in the solution, the following structure is developed. Each solution consists of three arrays. The size of the first array is equal to the number of candidates (n), while the size of the second one is equal to the number of different types of stations multiplied by number of candidates ($k*n$). The last array is the number of time slots multiplied by the number of candidates ($t*n$). To further clarify, assume we have two candidates, three time slots, and two types of BSS. The following is an example of a solution:

<table>
<tr><td colspan="6">1</td><td colspan="6">0</td></tr>
<tr><td colspan="3">1</td><td colspan="3">0</td><td colspan="3">0</td><td colspan="3">0</td></tr>
<tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

In this solution, first row determines which candidate is selected. Second row indicates which types of stations are selected to be constructed and finally last row shows in which time slots the stations are functioning.

2.4.1.2 Operators

In this section, we introduce the operators that are used in the TS algorithm. In order to find an immediate neighbor of a solution, there are two solution components that can be altered. First is the type of BSS and second is either adding or removing a BSS based on some performance measures. In the following section, we introduce the operators that we use in TS. First, we define the performance measures for adding operators.

2.4.1.3 Performance Measures

Different performance measures can be considered in placing a station. First is the distance from the demand locations. A populated area with multiple demand points, supplier locations and

other charging stations is a good choice since they are in a close proximity. The new station can be used for delivering goods to demand points in its coverage area, or to other parts of the network or even to improve the coverage area of other charging stations. This performance measure can be defined as follows:

$$PM_i^S = \sum_{j \in FS(i)} |b_j| \tag{11}$$

Equation 11 is a summation of absolute values of all supplier and demand points accessible from station i. We call it *Performance Measure Station* since it calculates the impact of the station on the other stations.

The other impact of adding a new station is on improving the coverage of the entire network. The selection process favors the stations that make the entire network more connected. For example, if selecting station i connects two unconnected clusters of the network, it is more preferable than selecting a station that solely covers some demand points and is disconnected from other parts of the network. This performance measure seeks to incentivize the creation of a coherent and well-connected network. Let set N_s the biggest connected component of a network before adding station i and N_s' be the biggest connected component of the network after adding station i. Thus, we can define this performance measure as follows:

$$PM_i^N = |N_s'| - |N_s| \tag{12}$$

Equation 12 calculates the number of stations connected to the network ($|N_s|$ is the cardinality of set N_s). We call it *Network Performance Measure.*

So far, we discussed the positive impacts of adding a new station to the network. The other aspect is the cost of adding a new station.

Finally, we introduce performance measures for selecting new stations. The benefit-cost ratio (BSR) is a reasonable representation thus we propose the *Total Performance Measure* of station i

is defined as follows:

$$PM_i^T = \frac{PM_i^N * PM_i^S}{f_i} \tag{13}$$

Equation 13 is the BCR indicator for station i. Its numerator is the potential benefits of adding station i (Connectivity and Coverage). Since PM_i^N does not increase with a rapid slope, it is more representative of the impact when is multiplied by PM_i^S. The denominator is the required cost of constructing the station (f_i).

Given the aforementioned performance measures, we can define the following insertion operators:

- Performance Measure Insertion (PM Insertion):

  This operator constructs the BSS candidate that has the highest value of PM^S. If the BSS is already constructed, the next station will be selected.

- Network Performance Insertion (PN Insertion):

  The station with the highest value of PM^N is selected to be constructed.

- Total Performance Insertion (TP Insertion):

  The station with the highest PM^T value will be inserted into the solution.

- Random Insertion:

  A random station will be selected to be inserted into the solution.

The other types of operators are removal operators. These operators help to refine remove some extra stations in order to reduce the cost of a solution. In other words, they evaluate to see if the saving on the construction of a station is justifiable by the lost sale it might cause. Here is the list of them:

- Smart Removal:

  We calculate a utilization value for each constructed station. The utilization value is equal to the number of used batteries divided by the total number of batteries at that station. In

other words, utilization of a station shows how many UAVs is served. Smart removal strategy removes the station with the least utilization value.

- Random Removal:

  A random station will be removed from the solution.

We also developed operators that combine both the insertion and removal strategies. The first operator is the *Swap Operator*. Swap operator randomly selects a station (C_1) to be removed from the solution. Then among all stations that are reachable by the removed station C_1, we randomly select one station to be inserted to the solution. Similar to the swap operator is the *Broading Operator* which chooses two stations to be inserted instead of one station. Thus, this operator will improve the coverage area of the network.

So far all the operators are altering the set of selected stations. There are two other operators that can change the type of the selected stations: *Upgrade* and *Downgrade*. This operators change the type of a station to include more batteries (Upgrade) or less number of batteries (downgrade). Finally, it is worth noting that a station is only operational when an UAV passes from it otherwise the station will not be functioning.

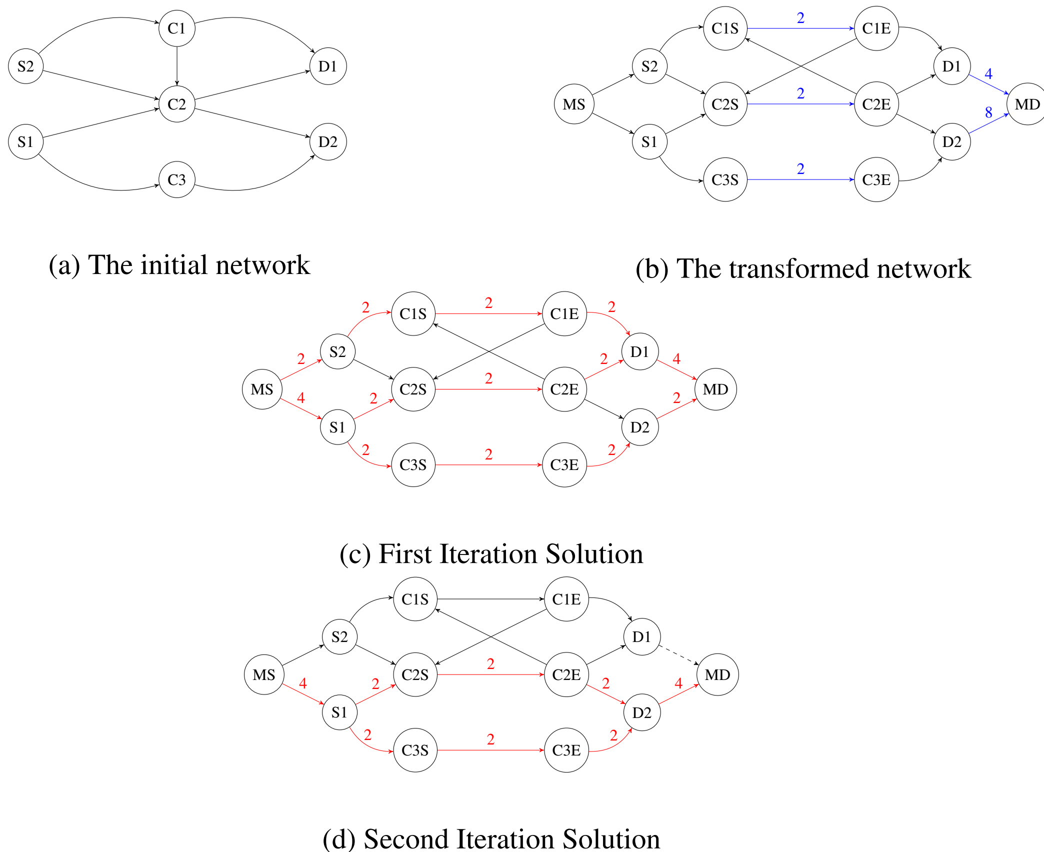
2.4.1.4 Objective function evaluation

The objective function of the problem consists of three terms. The first and the third terms are associated with the construction and operating costs. Since its unit cost parameters are deterministic they can be calculated based on the solution. On the other hand, the second term, which is associated with the lost sale, cannot be calculated directly. The problem that arise for this term is, we need to the maximum flow possible between all suppliers and demand points via the constructed network. The maximum flow problem solution will determine the number of demand that the network is capable of satisfying. Therefore, lost sale can be calculated by deducting the solution of maximum flow problem from total demand.

In order to develop a network, two key components of nodes and arcs should be defined. All the locations including demand locations, suppliers and selected BSS are considered as nodes. In or-

der to calculate the maximum flow of the constructed network, We have to add a *MainSupplier (MS)* node as well as *MainDemand (MD)* node. MS is connected to all of the suppliers, while all the demand locations are connected to MD. With respect to the arcs, we define set A as the set of all the arcs. The arcs have no capacity except for those that are connecting the demand locations to MD. The capacity of those arcs are equal to the demand of the corresponding demand locations.

Using the described network, we need to solve a maximum flow problem via CPLEX for repeatedly. At each iteration, the maximum flow problem solved and using its solution, we can see which demands are satisfied. Then, we update the demand. In other words, we will reduce the capacity of those arcs that connect the demand locations and MD based on the amount of satisfied demand. Then, we repeat this procedure for the next time slot. We explain the procedure in more detail for the following network:



(a) The initial network

(b) The transformed network

(c) First Iteration Solution

(d) Second Iteration Solution

Figure 2: An example of different steps of calculating the lost sale for a network.

The network as shown Figure 2a, consist of two suppliers ($S1$, $S2$), three BSS ($C1$, $C2$, and $C3$) and two customers ($D1$, $D2$). We are planning for two time slots and each BSS has 2 batteries in each time slot. $D1$ and $D2$ demands are equal to 4, 8 respectively. Current structure of the network do not allow us to solve the maximum flow problem. First, there is a capacity on the nodes and second we do not have source-sink nodes. As a result, we need to transform the network to the network in Figure 2b. There are three steps for the transformation. First, adding aforementioned nodes MD and MS. Then, we break each BSS node to two nodes ($Start$, End) connected with an arc in order to eliminate the capacities on the node. The arc between the two nodes has a capacity equal to the number of batteries at the BSS. Finally, we need to limit the capacity of arcs going into MD from each demand point. This makes sure we do not deliver more than we need to for each demand point. Blue arcs in figure 2b are the capacitated arcs. Now, we solve the maximum flow problem for the first time slot. Figure 2c illustrate the flow on the transformed network. In this solution, 4 and 2 unit of demand of $D1$ and $D2$ is satisfied. Therefore, we change the capacity of ($D1$-D) arc to zero and ($D2$ - D) arc to 6. In the second iteration, 4 units delivered to $D2$, 6 in total. So there is 2 unit of lost sale associated with $D2$. Note that the capacity of ($D1$-D) is zero in second iteration since we satisfied all demands in the previous iteration.

Note that by executing this procedure on all the time slots, not only we will have the exact number of lost sales but also the stations that are operating within each time slot. Hence the exact value of the objective function can be calculated.

We discuss the NP-hardness of the problem in the following. The capacitated fixed charge network design problem (CFCNDP) is a well known NP-hard problem [53]. Here is the definition for a general CFCNDP problem. A given network consist of (N,A). Each arc ($a_{ij} \in A$) has fixed charge (f_{ij}), variable charge (c_{ij}) and a capacity (u_{ij}). The problem is to minimize the cost of

flow in the network to satisfy different demands. Here is the formulation of CFCNDP problem:

$$Z = min \sum_{(i,j)\in A} c_{ij}x_{ij} + \sum_{(i,j)\in A} f_{ij}y_{ij} \tag{14}$$

Subject To:

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = \begin{cases} d & \text{if } i = O(k), \\ -d & \text{if } i = D(k), \\ 0 & \text{Otherwise} \end{cases} \tag{15}$$

$$0 \leq x_{ij} \leq u_{ij}y_{ij}, \qquad \forall (i,j) \in A \tag{16}$$

$$y_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A \tag{17}$$

$O(K)$ represent the set of depots and $D(K)$ represent set of demand location. We can reduce this problem to our problem with the following transformation steps:

- Each Arc transforms to a BSS station (ij) with a construction cost of f_{ij}, operational cost of c_{ij} and battery capacity of u_{ij}.
- BSS Station (ij) is reachable from all adjacent nodes of node i and node j.
- All Nodes with a negative demand act as demand node and the ones with positive demand as supplier.
- The Lost sale cost is equal to zero.

Here is an example of how each arc in CFCNDP problem reduce to a BSS in our research problem:

(a) General Arc of CFCNDP

(b) Transformed Arc

Figure 3: Transformation of each arc of fixed charge network design problem to our research problem.

Each new arc in figure 3b has unlimited capacity and zero cost associated with it. This transformation can happen in a $O(|A|)$ which is polynomial. Now that we show that a general form of CFCNDP can be reduce to our research problem, we can say if there is an algorithm that solves our problem to the optimality which leads to solve the CFCNDP to optimality by using the reverse of the transformation (i.e. Each constructed BSS in the solution is equal to use of crossponding arc in CFCNDP). This proves that our problem is NP-Hard.

2.4.1.5 Initialization

In this step, a feasible solution is identified for the problem. Although we can start with no selected station, this is not a initialization condition because, given the defined operators, the algorithm is not capable of adding multiple number of stations in one iteration. Therefore no product can be delivered to demand locations, which in turn increases the objective function. As a result, the algorithm will find the initial solution (i.e., no stations) more attractive than adding a station to the solution. This necessitates an initial solution that contains a network of BSS to cover all the demand locations. In order to obtain such a network, first the farthest demand location to the depot is selected. Then, using the dynamic programming (Markov decision process), a chain of connected BSS's is constructed to which the farthest demand location is connected. In the next step, the farthest unconnected demand location to the network is selected and using same approach it will become connected. This will continue till we have a connected network that covers all demand locations. In the following we explain the dynamic programming procedure that we used to find a connected route between two nodes.

In order to find a route between two nodes (S, M), we start from one node (S) and using equations 18 and 19 to find the route with the highest value. R_S is all demand locations that are reachable from node S. R is the set of all demand locations covered by a node on the route. N_S is the set of nodes adjacent to Node S. v_i is the demand value of demand location i and d_{iS} is distance between i and S.

$$V_M(S,R) = \begin{cases} D, & \text{if } S = M \\ F_S(R_S,R) + Max_{S' \in N_S} V_M(S', R \cup R_S), & \text{otherwise} \end{cases} \tag{18}$$

where:

$$F_S(R_S,R) = \sum_{i \in R_S/R} v_i + \sum_{i \in R_S \cap R} 0.5 * v_i + \sum_{i \in R'_S/R} v_i/d_{iS} \tag{19}$$

Equation 19 is the score function. This function consists of 3 terms. First term is equal to summation of all the demand nodes that are not covered by the route so far. Second term is a partial value for the covered demand. The last term gives stations that are closer to uncovered demand higher score. Equation 18 is the recursive function to find the best connected chain of station from node S to node M.

Here, we explain the three terms in the value function in more detail. The first term (Covering Demand Term) is making sure that among adjacent stations to S, a station gets selected that covers demand nodes that are not already covered by the network. This will make sure that a route will be created that covers most of the demand nodes. The second term (Back-Up Station Term) is trying to give credits to a station that is near a covered demand. Although these demand nodes are already covered by other stations, this will assure that if the magnitude of demands are significant, the obtained network has several stations covering that area. The last term (Direction Term) is trying to select a station that gets the route near demand nodes that are not reachable from the network currently i.e. as the distance of station S gets closer to demand node i, the term v_i/d_{iS} gets larger.

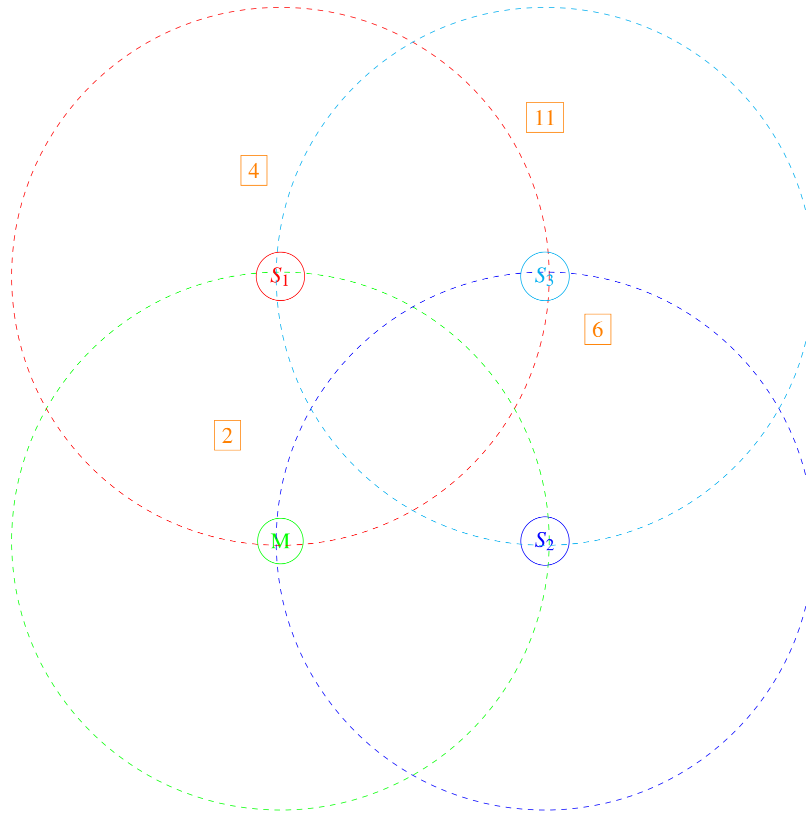


Figure 4: Example of how value function of dynamic programming works.

In Figure 4, the route is at node (M) (green node) and now it should select among S_1 (red node) and S_2 (blue node). The dashed circle shows the coverage of each node. Orange rectangles are demand points on the network. The demand node with a value of 2 is already covered by the network. Now we will compare S_1 and S_2 in terms of the value function. The Covering Demand Term for S_2 is higher than S_1 since it is covering a demand point with the magnitude of 6. On the other hand, the back up station term for S_2 is equal to zero but S_1 is reaching the demand that is already covered by M. Finally, S_1 is closer to the demand node with the value of 11 in comparison with S_2. As a result, the value function returns S_1 as the selected station for the next step. The other obstacle of using dynamic programming to find an initial solution is the curse of dimensionality. This means if the two nodes that we are trying to find a route between are far from each other, the computational time of calculating the best route between them grows exponentially. In order to over come this issue, we break down routes longer than ten time units of flying

range of UAV. It means if the rectangular distance of two nodes are more than ten time units of flying range of UAV, we will find several routes with the length of ten or less. The following figure represents an example of initial solution for the TS using the dynamic programming. The grid is the planning area and each intersection on it is represent a candidate location for a BSS. Red circles on the left represent the location of the demand points while the number inside them indicate the magnitude of demand. The black circles on the right is the output of dynamic programming. Each circles show the location of BSS on the grid. The first digit inside the circle is the ID of each BSS and the second one shows the type of BSS.

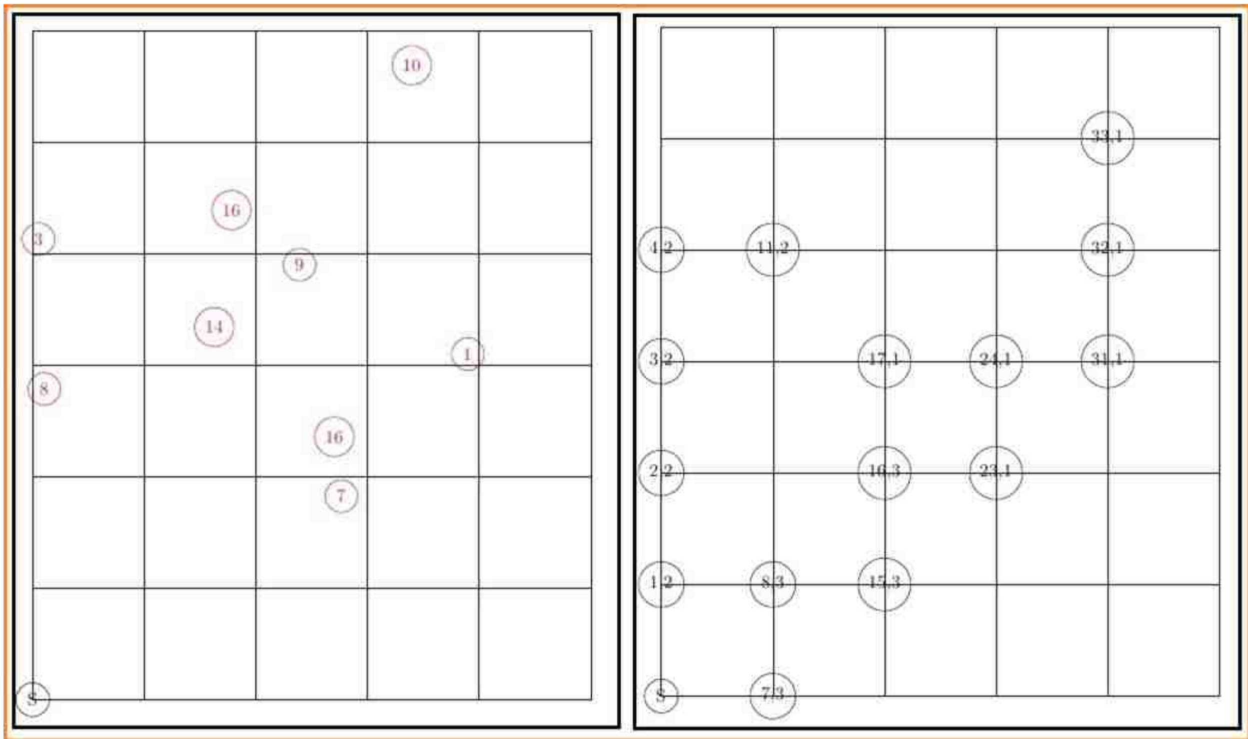


Figure 5: Example of an initial solution for Tabu Search

2.4.1.6 Tabu Search steps

A brief description of the developed tabu search is as follows:

- **Step 1. Initialization**

  Using the aforementioned dynamic programming, the initial solution of the tabu search is obtained.

- **Step 2. Creating Neighborhood Solutions**

  Using the defined operators, we generate a list of solutions that are immediate neighbors of the current solution. In order to calculate the objective function for each solution, we have to solve the maximum flow problem for each time slot. This would determine the demand magnitude that can be satisfied during each time slot.

- **Step 3. Finding the Best Neighbors**

  A solution with the minimum objective function value which is not forbidden by the tabu list is chosen to be the next solution. There are two types of tabu lists. The first one avoids adding a recently removed station and the second one does not allow to remove a recently added station. Note that if any of the solutions is better than what TS has identified so far, the algorithm will choose that solution regardless of the tabu lists.

- **Step 4. Update the tabu lists and Termination**

  The tabu lists are updated accordingly. If the algorithm reaches a predetermined number of iterations, it will terminate and the best solution will be reported. Otherwise, we repeat steps 2 and 3.

2.5 Computational Result

In this section, numerical tests are conducted for the proposed model. We first present the characteristics of instances followed by a discussion on the computational results.

2.5.1 Characteristics of Instances

There are four different parameters for each instance. Area of coverage, number of demand locations, magnitude of demand at each location, and number of available time slots. The instances are categorized in three different groups based on the number of demand points and BSS locations. Size of the region has a direct impact on the number of BSS candidates, while the number of demand locations impact the complexity of the BSS network. The following table shows how

the instances are categorized.

Table 2: Characteristics of instances

Scenario	Distribution
Low Demand Nodes (L)	*Uniform*(5, 15)
High Demand Nodes (H)	*Uniform*(15,20)
Small Region (S)	*Uniform*(20,30)
Medium Region (M)	*Uniform*(50,60)
Large Region (LA)	*Uniform*(80,90)
Demand Size	*Uniform*(1, 15)

For each category, the demand locations are generated based on a *Uniform* distribution function. Each location is randomly placed throughout the region. Also, BSS candidates are generated using a grid created in the region. Each square inside the grid has a side length equal to UAV flying distance. Each BSS Candidate is located at the intersection of the grid. In these instances four types of BSS with different attributes are considered (Table 3). For these instances, only one depot is considered and the penalty cost associated with the lost sale is $50,000$. The reason for using a large lost sale value is to justify the high cost of constructing a path from a depot to a customer.

In addition to the aforementioned parameters, we utilize the cost presented in table 3 in our instances. There is limited information available about the price of constructing a new BSS. Based on our research, the smallest station cost at least $20,000$. As a result, we estimate the other cost with respect to this number and assume the number of batteries is a driver for the cost. This assumption means that more number of batteries needs a bigger station which will cost more. In respect to operational cost, this is the cost to recharge the batteries and maintain the BSS. As a result the bigger station will have a bigger operational cost since it needs to recharge more batteries.

In total, 300 instances are generated and solved using the developed TS and CPLEX. There is

Table 3: BSS attributes

BSS Type	BSS Attributes		
	Construction Cost	Operational Cost	Number of Batteries
A	$20000	$10	5
B	$30000	$20	10
C	$35000	$40	15
D	$40000	$80	25

a 60 minute time limit for the CPLEX algorithm. This time limit allowed us to only solve 46% of the instances optimally, which gives us enough instances to evaluate the performance of the TS. We run the TS for the following number of iterations: 100, 200, 500 and 1000. ANOVA test at 5% significance level shows there is a significant difference between the results of different iterations. Tukey HSD shows that the results of 1000 iterations out performed others.
All of these approaches are coded in JAVA and executed on Intel Core-i7 3.6 Mhz, 16 GB RAM under Windows operating system.

2.5.2 Results

The results are represented in Table 4. The first column define the instance characterstics. For example LS_1 is a low demand scenario in a small region with 1 time slot.

Table 4: Tabu search results for 300 instances

	CPLEX			Tabu Search		
	Time (s)	# Optimal	Gap	Time (s)	CPLEX Solution Gap	Out Perform CPLEX
LS_1	1.35	10	0%	0.11	6%	0
LS_2	18.03	10	0%	3.03	2%	0
LS_3	1093.72	7	2%	217.70	10%	0
LS_4	798.72	8	0%	31.49	10%	0
LS_5	2973.77	2	36%	182.34	10%	0
LM_1	388.52	9	0%	29.64	18%	0
LM_2	2966.88	2	5%	555.50	10%	2
LM_3	3600.12	0	39%	622.01	16%	1
LM_4	3600.31	0	67%	28.52	8%	1
LM_5	3600.17	0	87%	602.92	9%	0
LLA_1	3140.88	1	4%	35.14	11%	2
LLA_2	3600.35	0	32%	224.52	10%	1
LLA_3	3600.27	0	62%	90.66	9%	4
LLA_4	3600.29	0	78%	451.99	12%	0
LLA_5	3600.29	0	92%	86.40	10%	2
HS_1	0.02	10	0%	0.00	10%	0
HS_2	0.06	10	0%	0.01	1%	0
HS_3	13.80	10	0%	2.42	9%	0
HS_4	464.89	10	0%	73.49	10%	0
HS_5	547.87	8	0%	83.62	3%	0
HM_1	0.33	10	0%	0.06	8%	0
HM_2	160.39	9	0%	131.20	10%	0
HM_3	864.13	8	0%	97.39	22%	0
HM_4	2731.11	2	1%	268.51	8%	1
HM_5	3150.97	1	4%	622.98	8%	3
HLA_1	221.31	9	0%	179.10	10%	0
HLA_2	2650.51	3	2%	411.67	6%	0
HLA_3	3346.79	0	6%	326.74	11%	2
HLA_4	3600.32	0	19%	387.56	11%	1
HLA_5	3600.27	0	22%	397.76	9%	4

It is apparent that by growing the complexity of the instances, the performance of TS with regards to the objective function value improves. In the small instances in terms of region, TS cannot outperform the CPLEX because it is trivial to solve the instances. However, in the medium and large region, the number of instances in which TS outperforms the CPLEX increases. The other benefits of using TS to solve the problem is the consistency of algorithm with the running time. The average running time for TS is about three minutes, while CPLEX uses 32 minutes on average. Table 5 summarize the result of 300 instances based on the region size.

Table 5: Summary of results of 300 instances based on the region size.

	Small	Medium	Large
CPLEX Gap (%)	4%	20%	32%
TS and CPLEX Gap (%)	7%	12%	10%
# of Instances TS out perform CPLEX	0	8	16
# of Optimal solutions by CPLEX	85	41	13
Average CPLEX running time (s)	591.22	2106.29	3096.13
Average TS running time (s)	59.42	283.36	259.15

Figure 6 is an example of the final solution of TS for 4 time slots. The initial solution of this instance is shown in Figure 5. Each black arc shows the flow between two BSS, while red arcs show the delivery to customers. The numbers on each arc is the total number of UAV trips. In each period, the depot will serve the nearest customer until one of the BSS run out of battery, in which case it waits until the next period for making new deliveries. This pattern is shown in Figure 6.

In this part, we provide a detail analysis of the impact of dynamic programming as an initial solution of tabu search. Fifteen instances are created based on LS_1, LM_1, and LLA_1 categories parameters explained in table 2.

Table 6: Result of 15 Instances

	Objective Value			Time (s)		
	DP	TS	CPLEX	DP	TS	CPLEX
1	240480	150090	140080	1.3	27.9	604.2
2	200400	140120	140120	2.4	108.1	210.3
3	360720	220230	215200	1.3	79.6	395.1
4	360720	260300	240290	3.6	113.1	533.7
5	280560	255450	235330	4.0	107.8	188.2
6	320640	180100	170090	18.0	211.9	1456.6
7	280560	225300	220280	19.0	238.4	914.7
8	520960	360380	355360	19.3	387.5	776.1
9	480960	325300	315260	82.6	211.6	1781.3
10	480960	400540	320360	38.7	223.1	771.3
11	601200	440480	390300	15.2	330.7	447.1
12	761520	480320	435280	110.5	559.1	3600.2
13	921840	720820	605690	54.6	482.1	3600.0
14	841680	585500	510480	14.6	388.9	3600.0
15	721360	530650	495620	38.3	185.4	539.2

Table 6 shows that dynamic programming on average accounts for 9% of running time of TS. Also, TS on average improves the result of dynamic programming initial solution by 29%. Still, TS solution is on average for these 15 instances, 8% worse than CPLEX solution but 69% faster. Lastly, we select 5 instances where the best solutions are obtained by CPLEX (*C_Instances*) and 5 instances where CPLEX outperformed by TS (*TS_Instances*) (Instances selected among 300 instances presented in table 4). These 10 instances are solved by CPLEX with an initial solution provided by TS and Dynamic programming.

When instances are solved using TS as an initial solution, *C_Instances* solution does not improve

and in one case leads to a worse solution and for $TS_Instances$, 3 out of 5 instances is improved compare to current best solution. The reduction in optimality gap is on average 3.18%. The dynamic programming initial solution does not impact the solution obtain by CPLEX in any cases.

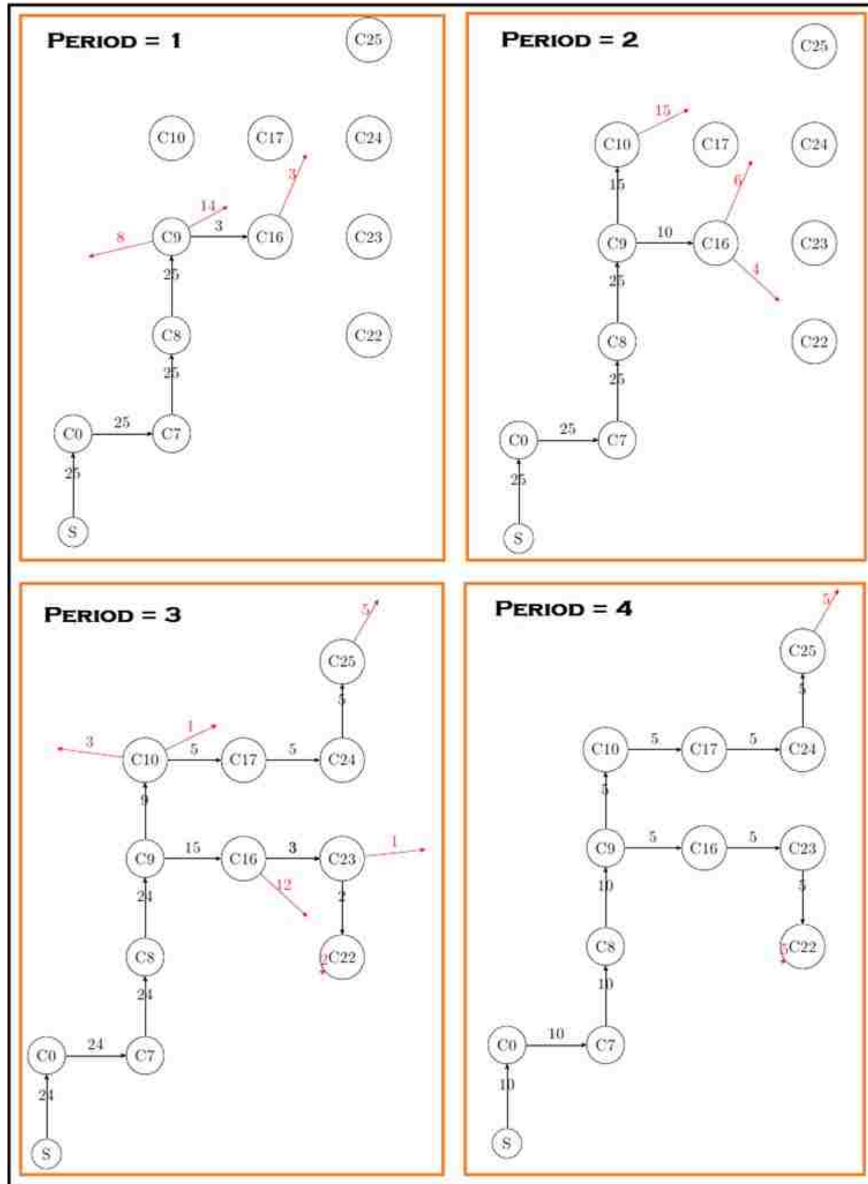


Figure 6: Final Solution from Tabu Search for an instance

2.6 Conclusion and future study

In this paper, we focus on locating battery swap stations for UAVs that transport commodities from suppliers to customers. The problem is designing a network of battery swap stations in the planning area with limited number of batteries in order to enhance the flying range of UAVs. The customer demands are assumed to be uncertain while a penalty cost is considered for lost sales. We developed a scenario-based stochastic programming model to satisfy the customer demands that allows constructing different types of BSS in the planning network.

To address the computational complexity of this problem, a Tabu Search heuristic is developed as the solution methodology. The initial solution is generated using dynamic programming. Multiple numerical tests are conducted using CPLEX and TS on a deterministic model (by considering only one scenario for demands) due to the complexity of the problem. The results show that, compared with CPLEX, TS generates significant improvement in running time and quality of solutions, especially for large problems.

As a future work, other solution approaches can be explored to solve the stochastic programming model with a large number of scenarios. Adding the time dimension to the problem, a dynamic model can be developed to construct battery swap stations over time.

3 Empty Container Management in an intermodal Network

In today's world, the transportation of goods is an essential part of supply chains. One of the mostly used way of transporting goods are third party logistics (3PL) transportation company. These companies are responsible to move goods from one point such as plant, warehouse to another point such as retailer. Usually company's request a container from transportation company and after loading it, the transportation company is responsible to move it to its the destination. One of the cost elements for transportation companies is the cost of repositioning their containers. These companies make a profit by moving a loaded container for a customer, but to provide them with a container, they need to reposition their assets accordingly. These moves are necessary due to an imbalance in the demand and supply of customers. For example, transportation companies move a loaded container through their network and deliver it to a customer. The customer empties the container and no longer needs to move goods. It is the transportation company's responsibility to move that container to a different location to satisfy another company's request. Companies remedy this cost inducing moves by utilizing different transportation modes such as inland, rail and maritime. There are advantages and disadvantages regarding intermodal moves. One advantage is that some modes are generally cheaper than the other ones. For example moving a container on a train is usually cheaper than moving it by truck due to fuel efficiency and labor. On the hand, the complexity of planning this move makes it harder for the companies to benefit from them to the best of their ability. If they are able to successfully reduce the number of empty moves, it will positively impact the operational cost to the company. In this chapter, we investigate this problem in collaboration with a major transportation company in the US. We investigate the challenges of planning empty container moves in the intermodal setting and develop a mathematical and forecasting model based on the needs of our collaborator.

3.1 Intermodal Freight Networks

Freight transportation networks deliver containers filled with goods from one point (its origin) and deliver them to another point (its destination). Freight transportation networks can employ any number of vehicles: large container-carrying ships, transcontinental railroads, barges, short-haul railroads and tractor trailers can all be used to transport freight in containers. However, due to a variety of factors (start-up cost, operational complexity, regulatory barriers to entry, etc.), usually just one mode is employed by any single freight transportation provider. Recently though, transportation providers have recognized the competitive advantage that can be achieved by working together. For example, from a customers perspective, its easier to deal with a single transportation entity rather than multiple entities, thereby making an integrated service operation more attractive to customers. Additionally, by pooling transport resources, transportation providers are able to provide a higher level of service, over a larger geographic area, at a lower cost.

The main issue being addressed in this research is that of network rebalancing. When not every node in a network has the same number of containers going into it as it has going out of it, surpluses develop at some nodes while shortages occur at others. If nothing were done to mitigate this behavior, all of the containers would eventually accumulate at a few nodes in the network and no more freight would be able to move. To prevent this from happening, network operators must perform rebalancing moves of empty containers. Rebalancing moves reposition empty containers from where they accumulate to where they are needed. Rebalancing moves are beneficial in that they allow the network to continue operating, but they are detrimental in that they cost money and do not directly generate revenue. As such, transportation network operators would like to make as few rebalancing moves as possible while still being able to move freight throughout their network. This makes it a minimum cost network flow problem. This research will examine how to best achieve this goal in an intermodal transportation network. This chapter examines the tactical planning problem of how to rebalance resources in an intermodal transportation network at a minimum cost.

The intermodal freight networks considered in this research are those consisting of trucks and trains. Intermodal freight networks exist to move their customers freight. Most customers usually have at least one empty container on site at all times. So, customers fill empty containers as needed and when the customer has a shipment that is ready to be moved, the customer contacts the intermodal freight coordinator and requests a container pickup. The intermodal freight coordinator then sends a truck to the customers site to pick up the laden container. In an intermodal regime, all containers are treated as full and no consolidation takes place. The truck then takes the container from the customers site to a railway ramp. The railway ramp is where the transport mode transfer occurs. A ramp is a rail depot where containers can be transferred from trucks to trains and vice versa. The container may be put on a train immediately or it may wait in the rail yard for a few days to be assigned to a train. When the train is ready, it departs the ramp and heads to other ramps. Eventually, the container gets to a ramp where it is transfered from a train onto another truck for delivery to the containers recipients site.

3.1.1 Comparison to Over-the-Road Networks

What is the benefit of intermodal freight transport? Intermodal freight transport is more logistically challenging than over-the-road transport, as it involves more parties and vehicle and schedules, and it is also more labor intensive as there is significantly more container handling involved. So why would anyone ever choose to use an intermodal carrier as opposed to a point-to-point over-the-road carrier? The answer is because of a significant cost savings.
Moving freight by rail is significantly cheaper than moving freight by truck for two reasons: moving freight by rail uses significantly less fuel than by moving freight by truck and there is significantly less driver labor involved. One truck can haul one container or occasionally two; whereas one train can haul two hundred containers. A train is significantly heavier than a truck, but it is not one hundred times heavier than a truck. So there is less non-revenue generating weight being moved on a train than on a truck. Alternatively, one can think of a train as being a more cost-efficient source of thrust. Another reason for the fuel efficiency of trains is that they can

maintain a constant speed and do not have to speed up and slow down as much as trucks (although trains do have to slow down or stop for other trains from time to time). Additionally, long-haul trucking requires one or two drivers for each container being transported, whereas a train requires a crew of operators while hauling possibly two hundred containers. This reduction in driver labor more than offsets the increase in container handling.

The advantages that over-the-road networks have over intermodal network are speed and flexibility. With an over-the-road network, the primary delay is waiting for a container to be picked up at the customers site. In an intermodal network, the container waits to be picked up at the customers site, waits at the origin ramp, the destination ramp, and possibly intermediate train stations as well. Additionally, the railroads in the United States west of the Mississippi River are not well-connected north-south. This makes travel by rail infeasible or impractical.

3.1.2 Explanation of Typical Problems that Arise

There are many logistical challenges that arise in an intermodal transportation network. Drivers have to be scheduled, trucks have to be maintained, containers have to be organized and allocated onto an appropriate train at the rail yard, locomotives have to be schedule and maintained, etc. All of these problems are rich and challenging in their own right. This chapter will only examine a subset of the problems associated with resource management in intermodal freight transport: empty container management, i.e., container stacking and repositioning. The reason these problems are challenging is due to the stochastic nature of supplies and demands of empty containers and the variable travel times of containers through the network. These issues will be discussed further in the next section.

3.1.3 Empty Container Management

Most realistic transportation networks are unbalanced, i.e., some nodes have more containers going into them than out of them and vice versa. In locations that have more containers going into them than out of them, e.g. Florida and the west coast, empty containers build up and must ei-

ther be stored or else relocated to where they are needed. In locations that have more containers departing than arriving, empty containers must be shipped in from elsewhere to satisfy demand. Some degree of network balancing can be achieved through price signals, but market forces usually require some amount of empty container rebalancing to occur. Rebalancing flows of containers can either be small scale, i.e., consisting of just a few containers and cost nothing, or else they can be large scale (a locomotive can pull 200) and incur a significant cost. Due to pricing structures, it is unlikely that a rebalancing flow of 5-180 containers would ever occur. The empty container management problem then becomes a matter of how to most cost-effectively rebalance empty containers while satisfying customer demand, which is itself not known with certainty until the period in which it occurs.

Seasonality adds another layer of complexity to the problem. There are peak demand seasons in the spring and in the fall during which the network is unable to meet demand in a timely manner in some locations. This requires the build up (pre- positioning) of empty container inventory in high-demand locations in anticipation of meeting the surge in demand.

3.2 Empty Container Management Literature Review

The empty container management literature can be divided into two different domains: maritime transportation and inland transportation. In this section, we summarize relevant literature, identify synergies and potential gaps associated with Empty container management. In the following, we first review the existing literature for both maritime and inland transportation domains and then identify how this research addresses the existing gaps in the literature.

3.2.1 Maritime Transportation

The use of containers for transporting goods between two continents has increased dramatically in the past years and is a reason for significant interest in the empty container management studies. In 1995, Shen et al. [74] studied this problem as a planning problem. They developed a decision support system for a shipping company to tackle the empty container management problem.

The problem is modeled as a mixed integer program (MIP) and used to answer these strategic questions: how to determine the number of leasing containers, how to reposition the containers and how to preposition the containers (i.e., how to move them to another port when we expect demand to appear there). Their DSS interface is developed in Excel with an AMPL solver. In their paper, they do not solve any instances therefore no performance efficiency is reported. Cheung et al. [17] considers a similar problem but proposes a dynamic empty container allocation model. They formulate the problem as a two-stage stochastic program and model it as a min-cost network flow problem. They solve these models using a stochastic quasi-gradient method and stochastic hybrid approximation procedures. In addition to these methods, hybrid solution techniques are proposed as well. They have three instances to test their model, 8, 10, 15 ports over a seven-day planning horizon.

Choong et al.[19] are the first to formulate the problem differently by studying the effect of the planning horizon length for shipping companies. Unlike previous studies, they formulate the problem to minimize the total cost of moving empty containers and illustrate the effect of planning horizon on empty moving cost. Container-on-barge operations within the Mississippi River basin is studied in their research. The size of their problem is 3 ports and 12 customer locations (along the river). They show that a more extended planning horizon provides shippers with the opportunity to use slower, less expensive modes of transportation. Olivo et al. [63] model the same problem with an hourly time-step but weekly planning horizon. They implement two integer programming (IP) formulations. A case study with 9 ports and 35 customers on the Mediterranean basin network is proposed, and ports act as storage pools with some maritime links existing between them. CPLEX is used as the solver for their problem. Jula et al. [52] add a new element of supporting retailers for each port. These retailers will help the port with its empty containers needs. They formulate the problem as a deterministic mixed integer program and use two-phase optimization to solve the problem. The first step transforms the model into a bipartite transport network and then matches the demand and supply. They then use simulation to evaluate their approach for Los Angeles and Long Beach ports using current data and projected data for a

20 year period. The projected data is obtained from a technical report submitted to the Gateway Cities Council of Governments and not a specific forecasting model.

Shintani et al. [75] add routing and deploying ships as a new element to the model. They formulate the problem as a two-stage problem. The first stage chooses the best set of demand ports while the second stage determines the sequence that a ship should visit them. In addition to empty containers, they considered the loaded moves as well. The objective function in this problem is to maximize company profit, because of loaded moves, they may not necessarily minimize the cost of empty ones. They use a genetic algorithm to solve the problem and utilize a data set from companies in Japan. Their largest instance considers 20 ports.

Song et al. [79] take a new approach to the problem and consider the empty container repositioning problem as a flow balancing problem. They consider worldwide shipping service routes in the maritime environment for their model and develop a simple heuristic to minimize the repositioning cost. The novel element of this study is that they use their approach for point-to-point rebalancing versus whole system rebalancing. The point-to-point rebalancing is a better representation of the real world because there is no central system for rebalancing moves. Their work includes sensitivity analysis under deterministic and stochastic situations. They expand their work in [80] to consider multiple service routes, multiple vessels, and multiple voyages with a shortest path based integer program and a two-stage heuristic. They have tested their solution methodology on two small instances and a realistic large scale network instance. The small instance consists of 5 ports and 3 shipping service routes, and the large-scale network has 24 ports and 5 service routes.

Chang et al. [14] study a different aspect of the problem. For a vessel, containers compete for a limited space. Naturally, companies prefer to perform profitable loaded moves rather than incur the cost associated with empty container moves. However, to continue their operations, they are forced to do empty container moves. Chang studied the allocation of these moves with loaded moves and its trade-off. A LP model is developed to solve the problem, as well as a decision-assist tool for the intra-Asia route. They solved several instances from which the largest one includes 20 ports. Westarp et al. [89] expand Chang study to consider the effects of seasonality,

constant change in market change as well. They test this approach on random instances with five ports.

Another approach to these types of problems is to use an inventory approach. We briefly summarize the literature associated with an inventory approach. Li et al. [60] explore the problem from the inventory perspective. They formulate the problem for a single port as a (U, D) allocation inventory policy. (U, D) policy means empty containers are exported down to D when the number of empty containers exceeds D, and empty containers are imported up to U when the number of empty containers falls below U. This policy is different from a traditional (s, S) policy in inventory replenishment. The level of inventory is strictly non-increasing except in the event of a replenishment. However, the inventory of empty containers at a port may fluctuate with the arrival/departure of vessels. Li extends his study in [59] to minimize the average cost of such a policy and develop a heuristic model to find a (U, D) policy. Song et al. [78] extend Lis work and add uncertainty elements to the model for a cyclic shipping route. They developed a three-phase threshold policy. This policy estimates the total number of containers needed to be moved at the current time, then allocates the empty containers to moves and assigns them to the vessels. They evaluate their approach using simulation models against several heuristic policies.The studied instance consists of 3 ports. They show that their approach is on average 20% better than other approaches.

Song et al. [81] continue their effort to find the optimal policy for empty container repositioning by formulating the problem as two-state Markov process. They assume the flow of containers as continuous fluid flows and find an optimal policy's threshold in closed-form in terms of parameters. Yun et al. [94] study the problem to find a traditional (s, S) policy for a port. In addition to the traditional source of empty containers for a port (i.e. other ports), they consider another source of empty containers as well. A leasing company with a zero lead time. Using Arena and its optimization tool OptQuest, they have found near optimal (s, S) policies for their randomly generated problems. In 2014, Zhang et al. [95] expanded Li's model in [59] to include operating cost, idle cost, back order cost, and importing/exporting cost. They developed a (U, D) policy us-

ing a polynomial-time algorithm to develop a policy for a multi-port problem, and evaluate the performance of it using simulation. The largest instance consists of 49 ports and their solution. It is worth mentioning another possible view point to this problem. Bernat et al. [71] study the impacts of pollution and repair options. They develop a heuristic algorithm to find a near optimal policy for inventory management system parameters. The total cost increased 10% with the emission cost included.

There are several other studies that consider additional aspects of the problem. Imai et al. [49] study the shipping network design in addition to the empty containers distribution. The problem is formulated as a two-stage problem; locating the ports and creating the whole network and then distributing the empty containers along that network. Their largest instance consists of 19 candidates. Di Francesco et al. [31] study the uncertainty in the model for several parameters which cannot be forecasted using historical data such as demand of empty container. They claim the in the demand is significant enough that leads to new observed value that cannot be predicted. They formulate the problem as an integer programming multi-scenario optimization model.

Chang et al. [15] study the concept of interchangeability of containers. There are several types of containers in terms of length that usually cannot be used in substitution. They study an approach to substitute the containers in Los Angeles and Long Beach ports to reduce the cost of satisfying demand. They develop a heuristic algorithm to obtain the near-optimal solution.Their computation results shows 4% to 41% cost reduction. Moon et al. [61] consider another type of container in the study; foldable containers. These containers consume a smaller space and are easier to transport. The downside of using these containers is that not all ports have the equipment to fold/unfold them. They consider an environment where several ports have the capability of using them and developed a hybrid genetic algorithm to solve their linear programming problem. The largest instance consists of 25 ports.

Hariga et al. [45] investigate the coordination between ports and their customers and assume uncertain return time for the customers when they receive a loaded container. They also consider the capability of renting containers. Their research uses an exact solution procedure to find an op-

timal replenishment cycle and the required number of containers and trucks for the mixed integer non-linear model. They solved the model for one vendor and cluster all of its customers to one customer.

The literature on maritime transportation of empty containers is quite extensive. Researchers have looked at different approaches such as mathematical modeling and inventory policies. There are two aspects that we would like to point out. First, there is a limited number of DSS developed in this literature. Second, the size of instances does not exceed 40 ports across current studies. We believe this is due to the nature of the maritime shipping problem.

3.2.2 Inland Transportation

In this section, we summarize the literature on inland empty container management studies. Dejax et al. [30] provide a literature review on empty vehicle movements, the lack of any models for empty container management for inland transportation is pointed out. They note that the first study on empty container management was completed by Florez [33] in 1986 where they model the problem as a network optimization problem. Using network optimization techniques, Florez explores the impact of a rolling horizon. To the best of our knowledge, the first attempt to address the problem was made by Crainic et al. [23] in 1993. They model the problem using a variant of a location-allocation problem. They develop a heuristic solution as a solution methodology and evaluate their methodology on random sets of instances where the largest instance consists of 44 depots and 219 customers. They extend their study in [22] by developing a Tabu Search (TS) approach to the problem. In this study, the goal was to plan for three elements. First, where to locate the depots in the network, then how customers should be assigned to the depots, and finally how should containers move between depots. Note that in this study, movements of containers between customers are not considered. They perform extensive computational testing on random data sets where the largest instance was 44 depots and 220 customers. Bourbeau et al. [12] develop parallelization strategies for a typical branch and bound algorithm for the same problem. Eleven instances are generated to evaluate the new approach and the largest instance consists of

130 depots. Unlike Crainic in [22], they did not consider customers.

Bandeira et al. [8] develop a DSS for a multi-commodity variant of the problem. Loaded and empty containers are the two commodities in their study. In their research, the depots' locations are input, and their objective is to minimize total transportation, storage, and handling costs. They formulate the problem as a MIP and develop a two-stage heuristic to solve it. The first stage of the heuristic, prioritizes the capability of the company to answer all demands. In the next stage, they minimize the costs. The largest instance they study is eight customers and 216 containers. This is still a small instance by which to evaluate the performance of a DSS in this industry. Glock et al. [41] studied a different aspect of the problem. They formulate the problem to find the optimal number of returnable transport items (RTI) (e.g., container, pallet). Their instances consist of one vendor and multiple retailers. A heuristic approach is developed based on different parameters such as lead time and demand. They claim that the demand of each retailer and the capability for unloading a container are crucial to determine the number of RTIs.

There are a limited number of studies that address the intermodal environment for this problem and the resulting complexity. Dang et al. [26] study the repositioning of empty containers in the port area with several depots. In their research, the sources of empty containers are three points: maritime ports, inland depots, and leasing companies. They minimize the total cost of repositioning empty containers via a simulation-based genetic algorithm. Their problem instance consists of 4 depots. Jiele in [51] considers two types of containers and SteadieSeifi et al. in [82] consider perishable commodities. For perishables, the empty container management is more important because of the small lead time for an empty container provision. Their network consists of maritime moves and rail transport. Their MIP model is solved using an adaptive large neighborhood search. The largest instance in their study consist of only 20 hubs. Xie et al. in [91] point out an essential aspect of the problem. Authors advise that in the real world several companies should be in coordination with each other to perform the empty container management. Unlike most of the studies which assume a central planner, their study focuses on the decentralized intermodal environment which consists of a railroad and a maritime company. They approach the

problem as an inventory sharing game. Their designed solution is a buy-back contract to develop coordination between the two companies.

To sum up, we reviewed 32 studies related to container management. The most commonly used models for this class of problem are mixed integer programs (in the deterministic case) and two-stage stochastic programs (in the stochastic case). The mixed integer program approach uses binary variables to represent the decision of which storage facilities to use while the continuous variables are used to represent network flows ([24], [74], [19], [63]). In the two-stage stochastic programs, the first stage typically decides which set of facilities to use based on expected network flows, while the second stage evaluates network realizations [17], [75], [8]. Many of the models are executed on a rolling horizon basis, i.e., planning is updated every period based on the newly available information. Numerous papers noted the importance of end-of-horizon effects on their models. Figure 7 shows the journal sources and their corresponding number of articles for the 32 papers.

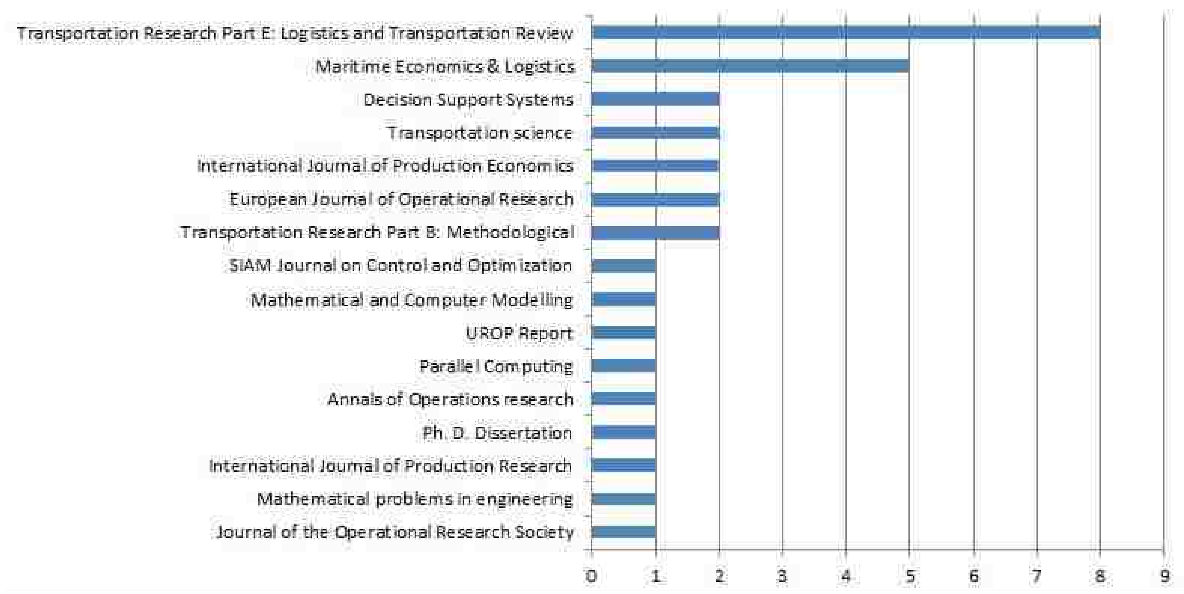


Figure 7: Number of Empty Containers Management by Publisher

Fig 8 shows the number of papers in the maritime versus inland transport domains.

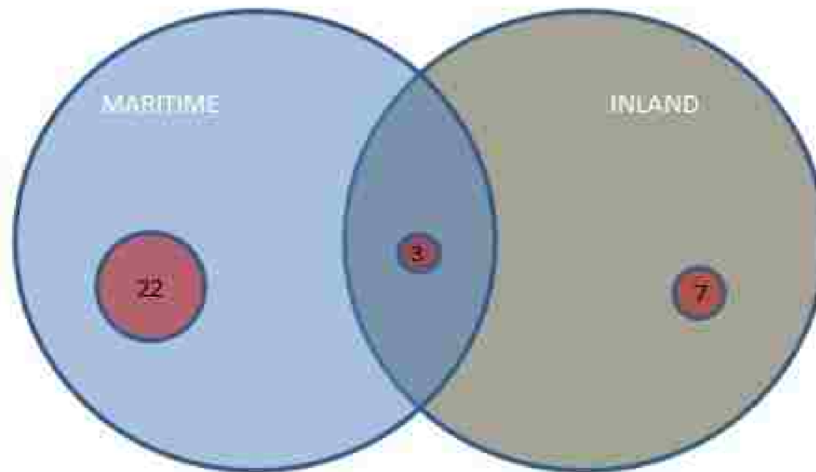


Figure 8: Number of Empty Containers Management Research in each domains

To conclude, we focus on three studies that pointed out the importance of our research. In the 2017 literature review conducted by Glock [40], they pointed out the lack of applicability of decision support systems for use on real cases in Returnable Transport Items (RTI). Therefore, he suggests that future DSS studies should be industry specific to improve the applicability of them. Theofanis et al. [87] review the major factors affecting the empty container management problem in different levels. In their detailed history of containerization since the 1970s, they have identified three different levels for the problem, global, interregional and regional-local levels. The global level is the maritime movement which is not the focus of our study. However, interregional and regional-local levels include the intermodal moves. The main decisions for the land environment (interregional and regional-local) are shown in Fig 9. As depicted, the interregional level, forecast, repositioning choice and routing are three out of four main decisions that we try to address in our modeling effort.

Finally, in 2017, Sze et al. [86] state that the containerization of inland transportation is more complex than maritime. These complexities arise due to several uncertainties such as customer's unloading time. As they state, limited research had been done on empty container management for inland transportation. Sze et al. [86] develop a simple heuristic to address the problem for random instances as big as eight depots and 15 customers. Significantly smaller than what is needed

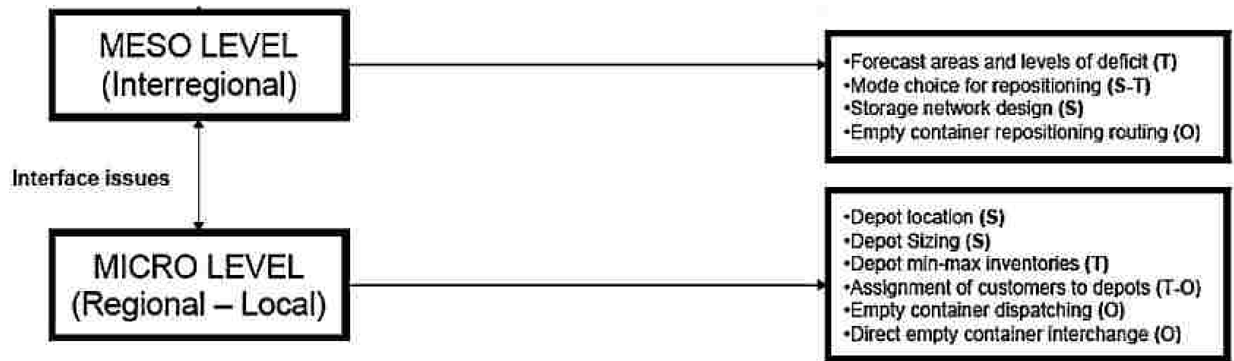


Figure 9: Empty container flows- regional and local level (adapted from [87])

by industry. We summarize the studies on empty containers management in table 7.

The contribution of our research is supported by these three studies. In our research, we address three of the known deficiencies in empty container management modeling. First, developing a new DSS targeted for a major transportation companies in the trucking industry by considering its specific characteristics. Next, developing a model to integrate the decision making process for three out of four major factors of the domain, forecasting, repositioning choice, and routing the containers. Finally, we solve a large scale representative instance which is orders of magnitude larger than those formulated in the literature.

Table 7: Classification of the literature of empty containers management modeling

	Domain	Model	Type	Solution	Size
Florez [33]	Inland	MIP	OR	-	10
Dejax et al. [30]	Inland	-	-	-	-
Crainic et al. [23]	Inland	MIP	OR	Heuristic	44
Crainic et al. [22]	Inland	MIP	OR	Tabu Search	44
Shen et al. [74]	Maritime	MIP	OR	Exact	N/A
Jiele [51]	Maritime	LP	OR		
Bourbeau et al. [12]	Inland	MIP	OR	Branch/Bound	130
Choong et al. [19]	Maritime	Planning Horizon	OR	Exact	12
Li et al. [60]	Maritime	(U, D)	Inventory	Heuristic	4
Olivo et al. [63]	Maritime	IP	OR	Exact	12
Jula et al. [52]	Maritime	MIP	OR	Exact	20
Shintani et al. [75]	Maritime	Stochastic	OR	Heuristic	15
Shintani et al. [75]	Maritime	Two Stage	OR	Genetic	20
Li et al. [59]	Maritime	(U, D)	Inventory	Heuristic	3
Chang et al. [15]	Maritime	LP	OR	Heuristic	20
Song et al. [78]	Maritime	Three Phase Policy	Inventory	Policy	3
Imai et al. [49]	Maritime	Two Stage	OR	Heuristic	19
Di Francesco et al. [31]	Maritime	Multi Scenario	OR	Heuristic	5
Bandeira et al. [8]	Inland	MIP	OR	Heuristic	8
Song et al. [81]	Maritime	Markov Process	Inventory	Policy	1
Song et al. [79]	Maritime	Deterministic/Stochastic	OR	Heuristic	N/A
Yun et al. [94]	Maritime	Policy	Inventory	Exact	1
Song et al. [80]	Maritime	Deterministic/Stochastic	OR	Two-Stage Heuristic	24
Dang et al. [26]	Inland/Maritime	(U, D)	Inventory	Genetic	4
Zhang et al. [95]	Maritime	(U, D)	Inventory	Heuristic	49
Westarp et al. [89]	Maritime	LP	OR	Exact	5
Bernat et al. [71]	Maritime	Policy	Inventory	Heuristic	1
Moon et al. [61]	Maritime	LP	OR	Genetic	25
Hariga et al. [45]	Maritime	MIP	OR	Exact	1
SteadieSeifi et al. [82]	Inland/Maritime	MIP	OR	ALNS	20
Xie et al. [91]	Inland/Maritime	Policy	Inventory	Policy	1
Glock [40]	Inland	N/A	N/A	N/A	-

3.3 Minimum Cost Network Flow Problems

This chapter considers a stochastic minimum cost network flow problem in the context of an intermodal freight transportation system. The first section describes the minimum cost network flow problem generally and the next section discusses the specific application dealt with in this work. The simplest version of the minimum cost network flow problem is to find the cheapest set of feasible flows through a network that satisfy all demands. Given a directed graph $G = (V,A)$ where V is a set of nodes and A is a set of arcs, the network flow problem is defined mathematically as [68]

$$\min \sum_{(i,j)\in A} c_{ij} * X_{ij}, \tag{20}$$

s.t.

$$\sum_{(i,k)\in A} X_{ik} - \sum_{(k,j)\in A} X_{kj} = b_k \qquad \forall k \in V \tag{21}$$

$$0 \leq X_{ij} \leq u_{ij} \qquad \forall (i,j) \in A \tag{22}$$

where $i, j \in V$ and $(i, j) \in A$. x_{ij} is the flow along arc (i, j) and c_{ij} is the cost per unit flow along arc (i, j). u_{ij} is the maximum capacity of flow along arc (i, j) and b_k is the demand at node $k \in V$. When b_k is positive, node k has a net demand and when b_k is negative, node k has a net supply. The objective function (20) is to minimize the sum of the cost of all flows through the network. Constraints (21) are flow balance constraints, ensuring that the flow into a node is equal to the flow out of a node and Constraints (22) ensure that arc capacities are respected and that there are no negative flows. In this simplest form, there is no dependency on time and all costs, demands and capacities are assumed to be known. This is a deterministic linear program and it can be solved very easily by any commercial solver, using either the simplex method or an interior point algorithm.

Unfortunately, most real-world problems are not this straight forward. To begin with, the assumption that all costs, demands and capacities are known may not be valid and frequently it is not.

Additionally, most real-world planning problems take place over periods of time (called planning horizons), which means that the linear program presented above must be modified to be indexed over time. Finally, in any sort of physical distribution environment, flows will not be instantaneous which means that there will be a lead time between when a flow begins and when it ends. This lead time too is frequently not known with certainty. All these sources of uncertainty make it necessary to consider the use of stochastic optimization methods, which will be discussed at the end of this chapter.

3.4 Empty Container Management Model

This chapter deals with the planning problem of repositioning containers in an inter-modal freight network such that demand is satisfied. We want to find the minimum cost solution to such a problem. The following sections of this chapter describe the problem and the modeling approach, the model validation process, and an analysis of the results. This section deals with the tactical planning problem.

3.4.1 Specific Problem Instance

This section will describe the network being studied, its operational characteristics and some baseline statistics to give the reader an idea of the scale involved. In the transportation network considered in this research, a rail operator has partnered with a trucking company to form an inter-modal transportation network, i.e., more than one mode of transportation is used within the same network. In this network, when a customer needs a container picked up, the customer calls the trucking company which then has a driver go to the customers site and pick up the container. The driver then takes the container to a railyard (or ramp) where the container is loaded onto a train. Once the train is satisfactorily loaded, it travels to another ramp where the container is then unloaded and delivered by a truck to its destination. [It is worth noting that in this regime, even though most of the distance the container travels is by rail, most of the cost incurred is by the truck. Due to economies of a scale (a train can carry 200 cars) and overall fuel efficiency, it is

much cheaper to use rail to ship containers whenever possible.] In this scenario the trucking company benefits by being able to provide a cost benefit compared with its competitors and the rail operator benefits from higher freight volumes (more business). If the network can be efficiently managed and coordinated, both transportation operators can mutually benefit from the partnership. There are, however, a few logistical challenges involved in operating such a network. The focus of this research is on that of empty container management.

The representative network under consideration consists of rail links and road links throughout North America. Rail links are typically cheaper than road links, but road links are economic for relatively short distances (i.e., less than 100 miles) and also have the advantage of being dense in the network (i.e., freight can go anywhere by road, but can only go where there are tracks by rail). The network we studied, mainly operates in the United States, but also provides some service to Canada and Mexico. The rail network studied consists of 153 ramps. Figure 10 on page 50 shows the location of these ramps. Since 2008, the network transported 40,316,447 containers. In order to serve those 40.3 million revenue generating loads, they had to move 651,539 empty containers. On average each empty moves costs $ 130 for the company. Due to the high cost of attracting and retaining drivers, rebalancing moves are only done via rail. The cost of hiring new drivers prevents the use of road rebalancing movements, even though it may be more cost effective over relatively small distances, e.g. the northeast. Any reduction in the number of required rebalancing moves would decrease operating expenses and would result in more profit for the partnership. That is the goal of this research.

Rebalancing moves (for the network under consideration) are currently centrally planned by a small team (3 or 4 people) at the network operators corporate headquarters. The central planning team faces multiple challenges:

- **Stochastic demand**: The biggest challenge facing central planners is that 45% of the companys demand is same day, i.e., a customer calls and wants a container picked up that same day. Such high variability necessitates high inventory level, i.e., having many empty containers on hand.

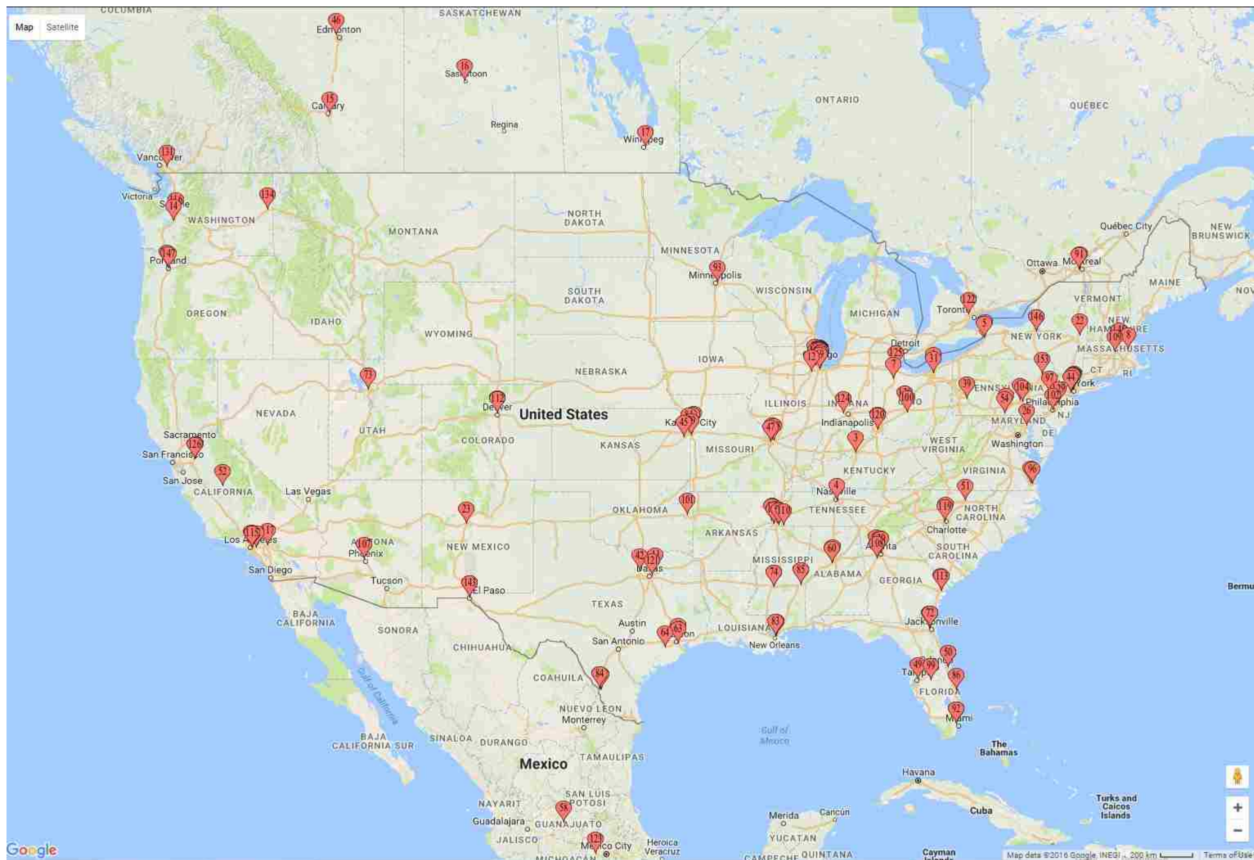


Figure 10: Ramps Location throughout USA, Canada and Mexico

- **Stochastic travel time**: This is less of a problem, but it still needs mentioning. While the average travel time for individual rail lanes is well known, any particular shipping time may vary by a few days.

3.4.2 Modeling Approach

Consider an inter-modal transportation network. The sets, costs, parameters and decision variables are described below. Note that *Location* means either a customer or a ramp.

- **Sets**
  - Customers (C): Set contains all customers. (Represented by n).
  - Ramps (R): Set contains all ramps. (Represented by k, q).
  - Location (L): Union of Customers set and Ramps set. (Represented by i, j)

– Time Line (T): set of time slots. (Represented by t)

- **Parameters**
  - C_{ij}^{T}: Cost of driving between a location i and a location j.
  - C_{kq}^{R}: Cost of rail movement between the ramp k and ramp q.
  - C^{I}: Idle cost associated with an empty container stored at a location.
  - C^{N}: Purchase cost of each new container.
  - S_{i}^{t}: Supply of empty container at location i at time slot t.
  - D_{i}^{t}: Demand of empty container at location i at time slot t.
  - Cap_{i}: Yard capacity of location i.

- **Decision Variables**
  - Y_{ij}^{t}: denotes the number of truck movements between location i and location j.
  - X_{kq}^{t}: denotes the number of rail movements between ramp k and ramp q.
  - I_{i}^{t}: denotes the number of containers stored at location i at time slot t.
  - Z_{q}^{t}: denotes the number of new containers delivered at ramp q at time slot t.

The mathematical model for our problem is given as

$$\min \sum_{t\in T}\sum_{i,j\in L} C_{ij}^{T} * Y_{ij}^{t} + \sum_{t\in T}\sum_{k,q\in R} C_{kq}^{R} * X_{kq}^{t} + \sum_{t\in T}\sum_{i\in L} (C^{I} * I_{i}^{t} + C^{N} * Z_{i}^{t}), \tag{23}$$

s.t.

$$\sum_{i\in L} Y_{iq}^{t} + \sum_{k\in R} X_{kq}^{t-1} + I_{q}^{t-1} + S_{q}^{t} = \sum_{i\in L} Y_{qi}^{t} + \sum_{k\in R} X_{qk}^{t} + I_{q}^{t} + D_{q}^{t} + Z_{q}^{t},\ \forall q \in R \forall t \in T, \tag{24}$$

$$\sum_{i\in L} Y_{in}^{t} + I_{n}^{t-1} + S_{n}^{t} = \sum_{i\in L} Y_{ni}^{t} + I_{n}^{t} + D_{n}^{t},\ \forall n \in C \forall t \in T, \tag{25}$$

$$I_{i}^{t} \leq Cap_{i},\ \forall i \in L, \forall t \in T, \tag{26}$$

$$X_{qk}^{t} \leq 200,\ \forall q,k \in R, \forall t \in T. \tag{27}$$

Table 8: System status since 2008 to 2015.

Year	Customers	Ramp	Total Moves	Load	Empty
2008	20680	86	824815	824815	0
2009	23971	100	977028	898709	78319
2010	25612	109	1150675	1044181	106494
2011	31056	114	1587055	1192158	29919
2012	36901	121	1824522	1371531	33988
2013	39799	126	1944778	1547965	39090
2014	43310	124	8981681	1665534	158988
2015	40969	125	10936469	1740037	204741

The objective function equation (23) consists of three main elements. The first term is the transportation cost via road. The second term is the cost of rail movements. The last term is the cost of idle containers and new containers. The constraint set (24) is the flow balance for each ramp. The inflow consists of incoming road and truck movements, inventory from previous time slots and empty containers supplied at the ramp. The out flow is the outgoing road and truck movements, inventory at the end of the time slot and the demand required at the ramp. Constraint set (25) is the same as the flow balance at a ramp just without the rail movement terms. Constraint set (26) assures that the number of containers stored at a location does not exceed its capacity. Constraint set (27) limits the number of containers on the rail movement to 200. The capacity of the rail line is determine by the system experts.

3.4.3 Data Analysis

The company has provided us with a representative sample data set of 11 millions rows for the period from 2008 to 2015. Each row of the data provides us with the shipper and receiver code, origin and destination ramp, revenue, pick up and delivery time, time stamp at each ramp and type of moves (Empty or Loaded). Table 4 presents a summary of the data. Note that the data for the empty moves in 2008 is missing and 2011 to 2013 is partially missing as well.

In order to get an initial starting point for our model, the location of each container in the system needs to be known. Therefore, a simulation model is developed using similar data to establish a

representation of the current system status and behavior.

3.4.3.1 Simulation

In order to obtain the current state of the system, a monte-carlo simulation has been developed. In this section, the detail of the simulation model used is explained.

Each row of the provided data represented a move. A ”Move” is defined to be the transport of a container (Loaded/Empty) between two locations (Customer/Ramp). Each row has several columns such as ”pick up date”, ”time stamp at origin stamp”, ”time stamp at destination ramp” and ”delivery date”. Using this information, three events can be defined. Scheduling a Move, Initializing a move and Arrival of a container. In the mathematical model, truck delivery is considered to be within the same day of the container arriving at the ramp. As a result, the time that the container arrives at the origin ramp is assumed to be the date that the move started (In_Gate time stamp) and the date that the container leaves the destination ramp is considered the delivery date (Out_Gate). Each move is flagged as loaded or not as a result of when a move is concluded and we assume the container will be available for use after seven days. This timeline lets the customer unload their products.

3.4.3.1.1 Scheduling a Move

Scheduling a move is the act of planning an initialization of a move. The logic of this event is presented as follow:

Algorithm 1 Logic of Scheduling

```
if In_Gate Available then
  Start at In_Gate
else
  if Pick_up date available then
    Start at pick_up date
  else
    Schedule arrival at Out_gate
  end if
end if
```

In this psudo code, we assume if the In_Gate data is available we schedule the move to be started at that time stamp if not, we check to see if the data for truck pick up (Pick_up) is available or not. If we have a pick up date we can schedule the move at that time. In the cases where we have no information about when the move is initiated we only schedule the arrival of the container at the destination. Note that, all rows have an Out_Gate date.

3.4.3.1.2 Initializing a Move

Initializing a move is the act of picking up a container from a location and starting its move through the system. The logic of this event is presented as follow:

Algorithm 2 Starting a move logic

```
if Container is Free at Ramp then
  Use that container and schedule arrival
else
  Create a new container and schedule arrival
end if
Read a new Line and schedule a move
```

The logic consist of evaluating the status of containers at the pick up location. Since we do not

have any information about the state of the system, all locations are considered empty. With each move, if the system cannot find a free and idle container it will assume that there is a container at that location and create a new container to start a move. Upon finding the container, the system will schedule arrival using the Out_Gate time stamp.

3.4.3.1.3 Container Arrival

The last step of each move is the arrival of the container at the destination site. When a move concludes at a destination site based on the type of the move (load/empty) the container will become available immediatly (empty move) or 7 days later (loaded move)

Algorithm 3 Ending a move logic

```
if Empty Move then
    Set the container free
else
    Schedule a release after 7 day
end if
```

In order to better illustrate the process of simulation, we will walk through a couple of sample data rows presented in table 9.

Table 9: Sample Data for Simulation

	In Gate (day)	Out Gate (day)	Status	From	To
1	1	5	L	1	2
2	2	5	E	1	2
3	5	9	E	2	1

The Simulation first reads the first row, it will schedule a move at the ingate time stamp (Day 1). The only event in the model currently is the start move that is created by the first row of data. The model is advanced to day 1, the move is initiated from Ramp # 1 to Ramp #2. Since ramp #1 does not contain any container, we create a new container and schedule its arrival at ramp #2 at

day 5. Now, we read the second line of the data and schedule a move for day 2. Now, the model advances to second day and schedules an empty move between ramp #1 and #2. then reads the third line and schedules a move from ramp #2 to ramp#1. The Model now advances to day 5, first we have two arrival and one start move. The model will always process the arrivals and then schedule the moves because it might be able to use the containers without creating a new one. The first arrival is the container from the first line of the data which is a loaded move. The system schedules the container availability for 7-days later which is day 12. The second arrival is the container from the second line. This is an empty move so the container becomes available immediately. Now it is time to schedule the move associated with third line of data. Since we have a container available at ramp # 2, we use that container to send to ramp #1. Figure 11 shows the flow diagram for one move in the system.

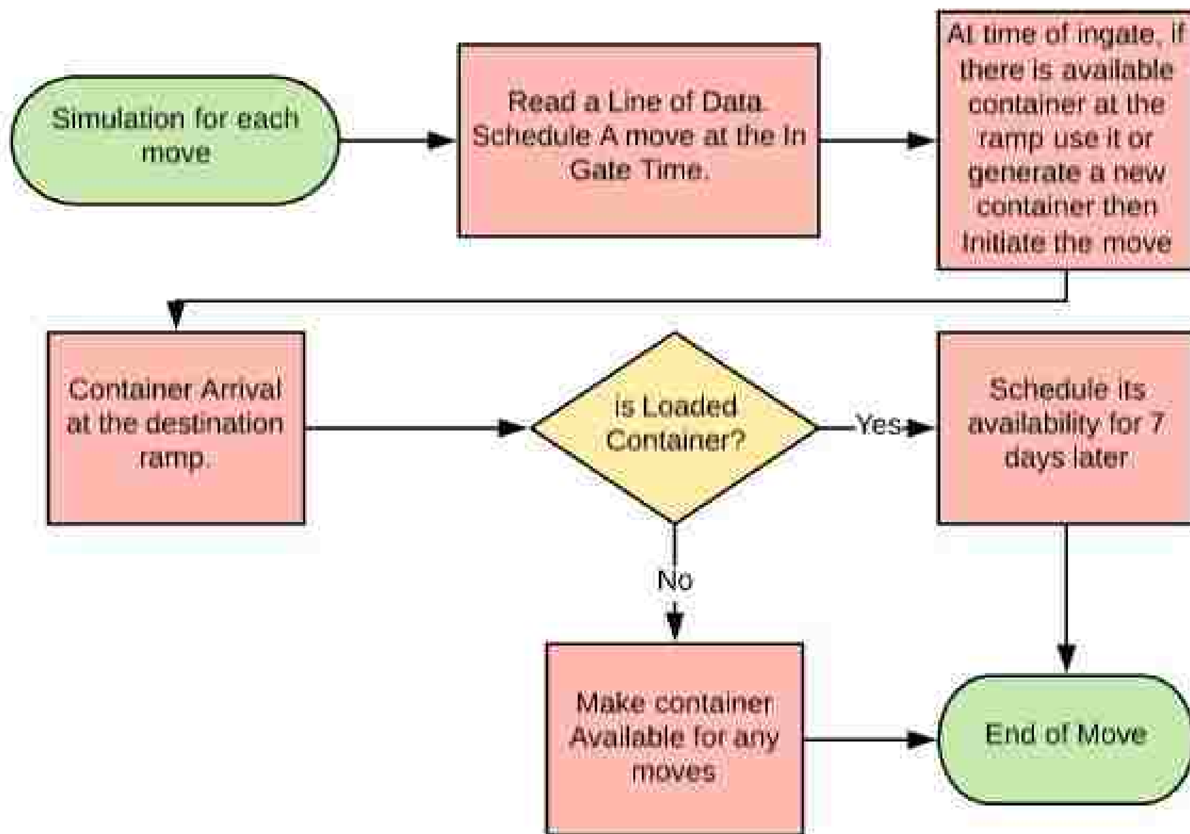


Figure 11: Flow Diagram for a move in the System.

In the next section we will present a summary of results obtained via the simulation model.

3.4.3.2 Simulation Result

In this section we analyze the results that we obtained from the simulation to give us some better insight to the system's performance and the impact of our data set.

3.4.3.2.1 Move Analysis

In this section, we focus on the behavior of the empty and loaded moves using our simulation and dataset.

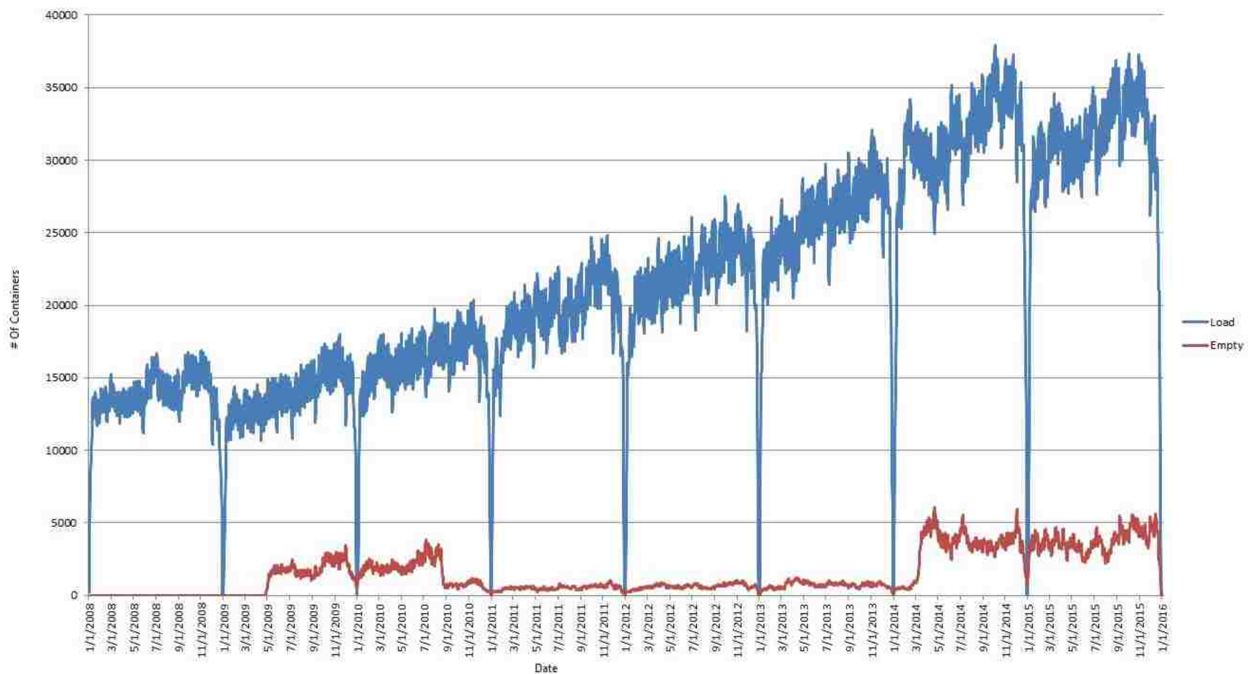


Figure 12: Number of loaded and empty moves

Figure 12 represent the number of empty moves (red) and loaded moves (blue) since 2008. Based on the increasing trend of loaded moves, the empty moves are expected to increase as well but during 2010 to 2014 there seems to be a drop. Based on discussion with the company experts, this might be due to their inability to track the empty moves during that time period. As a result, our analysis will focus on years 2014 and 2015.

3.4.3.2.2 Containers Analysis

In this section, we analyzed the status of the containers in the system. Figure 13 represents the status of each container at each point in time. Free and ready to use containers (orange), busy containers at a customer location (light blue) and on a train (dark blue) are shown. Also, the total number of containers are estimated to be $100,000$. We can observe the behavior of the system which tends to collect all of its containers at the end of the year. Company experts confirmed this behavior which provides some face validity for our simulation model.

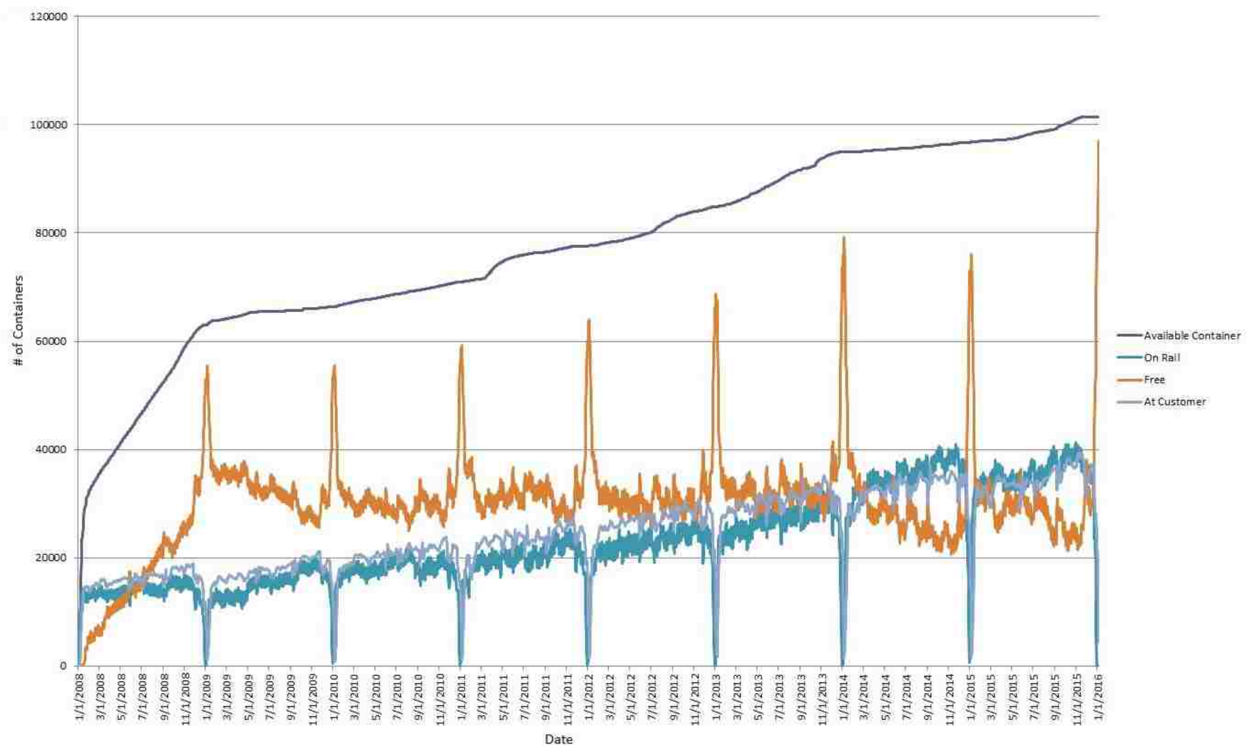


Figure 13: Status of containers

3.4.4 Forecasting Model

In this section we describe briefly our preliminary forecasting method for obtaining the future empty container demand for the company. The current method for forecasting the demand in the company is purely based on human judgment. A group of employees who work in different parts of the country use their experience to make a recommendation about how many containers their

region needs in the upcoming week. At the end of the week, they will decide on how they should make the moves in order to satisfy all demands. This approach provides us with an opportunity to reduce their error margin by using a forecasting model. For the first step of our research effort, we used the Seasonal Autoregressive Integrated Moving Average (SARIMA) model. The demand for empty containers is seasonal in its nature. For example, companies need to move more products during the holiday season.

Seasonal Autoregressive Integrated Moving Average model is a well-known forecasting method consisting of three different elements. First, the autoregressive (AR) model, this specifies that the value of the variable is dependent on its p previous values and a random (stochastic) term. The general form of $AR(p)$ is as follow:

$$X_t = c + \sum_{i=1}^{p} \varphi_i * X_{t-i} + \varepsilon_t$$

where c is a constant value, φ_i is the coefficient of autoregressive model while p is its order (number of time lags). X_{t-i} is the time series lagged i periods. ε_t is an error term following normal distribution with its mean as 0 and standard deviation as 1.

Second, the moving average (MA) model, the model predicts the value of the variable based on several past stochastic terms. The general form of $MA(q)$ is as follow:

$$X_t = \mu + \varepsilon_t + \theta_1 * \varepsilon_{t-1} + \cdots + \theta_q * \varepsilon_{t-q}$$

where μ is the mean of the series, θ_i is the coefficient of the moving average model while q is its order. $\varepsilon_t, \varepsilon_{t-1}, ..., \varepsilon_{t-q}$ are error terms.

Last, is the seasonality element. Seasonality in a model incorporates the behavior of the variable over different time periods. The SARIMA model utilizes autoregressive and moving average models to create an accurate forecasting model. SARIMA model is determined by four parameters. p and q which as defined above, are AR and MA parameters (non-seasonal part) and also P and Q which are defined as seasonality parameters of AR, MA models. P and Q are very similar to the non-seasonal elements of the forecasting model with the only difference that they involve backshifts to the seasonal periods. As an example, if a SARIMA($p = 1$, $q = 2$)($P = 1$, $Q = 2$) is

utilize to forecast January 2018 demand, the model uses historical data for December 2017 ($p = 1$ one period back), December and November 2017 ($q = 2$, two periods back) for non-seasonal part of the model and January 2017 ($P = 1$ one seasonal period back), and January 2017 and 2016 ($Q = 2$), two seasonal periods back) for seasonal part of model.

Each customers in this system has a unique behavior in terms of its demand request. The computational time to estimates all the coefficient of optimal SARIMA model is directly related to the magnitude of p, q, P, and Q. Since the company has thousands of customers, Finding the best forecasting model for each customer is not computationally feasible. Therefore, we need to come up with an approach to minimize the amount of computational time spent on the forecasting model. To this end, we limit the different parameters values to 0 and 1. Using these small values for the four parameters of SARIMA, made it possible to find suitable forecasting model for each customer. The combination of different value for p, q, P, and Q creates 16 different configurations for the SARIMA parameters (i.e. SARIMA(1,1)(1,1), SARIMA(1,1)(1,0), SARIMA(1,1)(0,1) and so on). Among these models, the best model is chosen using the Bayesian Information Criterion (BIC) [73]. The BIC is a criterion for selecting among a finite set of models. The most important benefit of using the BIC is the ability to automating the process.

The other concern about the forecasting model is the time unit. Although all of the data provided is daily, the amount of computation for finding a decent daily forecasting model is not reasonable or necessary at this time. As a result, for now the model is forecast monthly and the result of each month will be distributed uniformly over the 4 weeks which is consistent with our mathematical model. The company will provide us with the historical percentage of daily distribution of the demand over a week so we can calculate daily forecast as well.

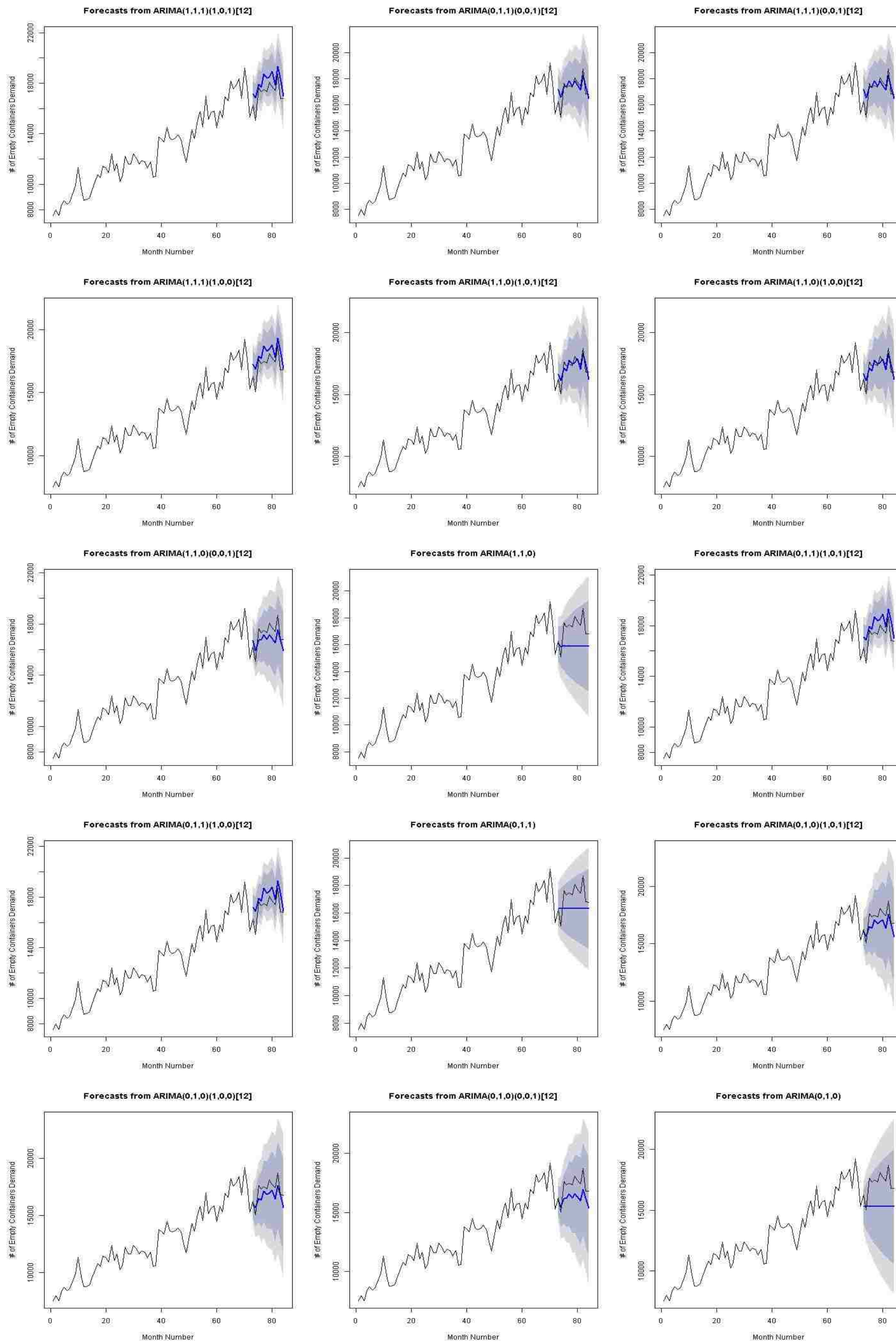


Figure 14: 15 Different Configurations for forecasting model of "Customer A"

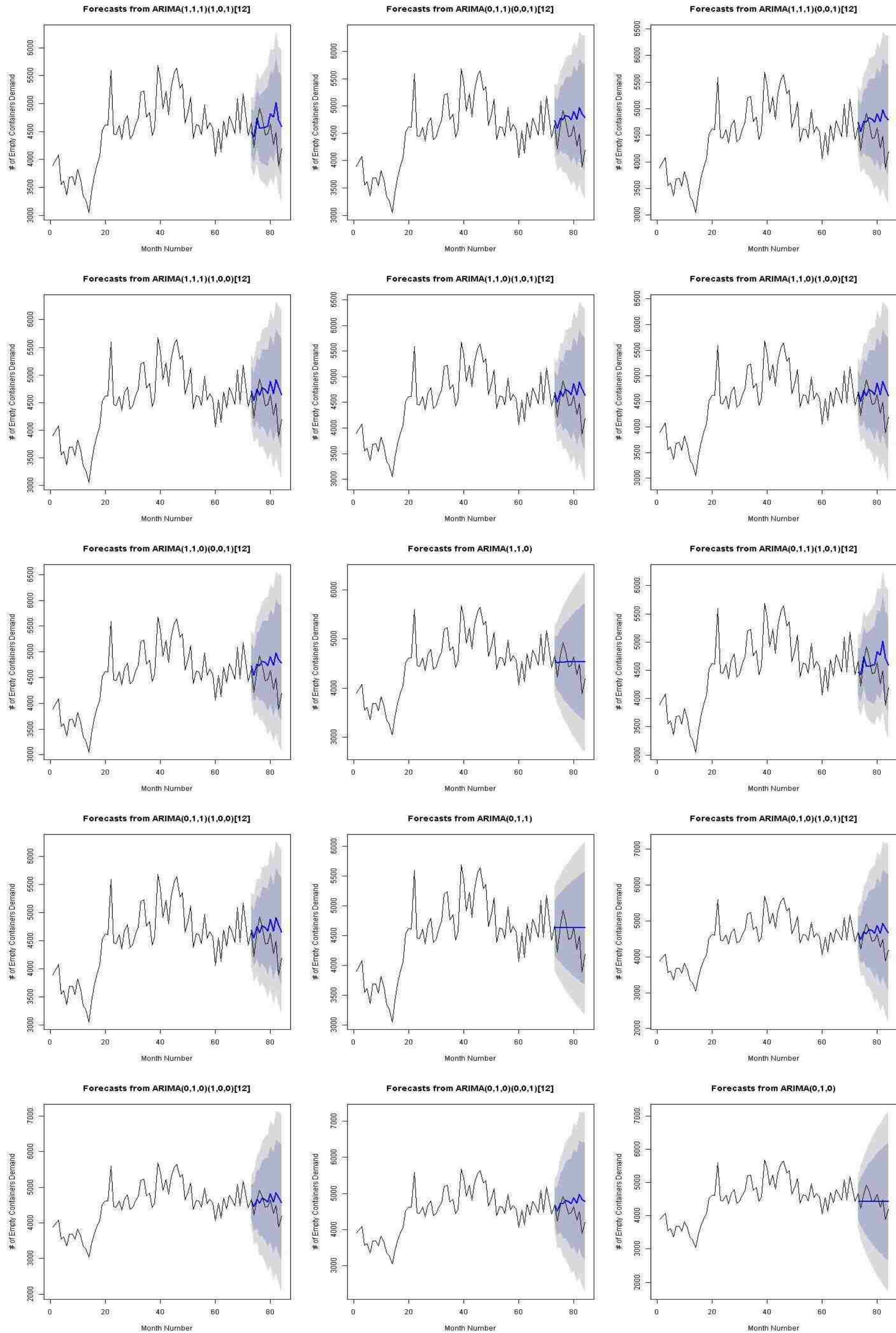


Figure 15: 15 Different Configurations for forecasting model of ”Customer B”

In figures 14 and 15, the forecast for two different customers are shown under 15 different configuration. The blue line shows the forecasted demand versus black line the actual demand and the dark shaded area represent the 95% confidence interval on the forecast and light shaded represent the 99%.

Among different configuration, SARIMA(1,0)(1,0) and SARIMA(0,0)(1,0) is the best fit model for customer A and Customer B respectively. For customer A, the model over estimates the demand and for customer B it underestimates the demand. In order to address this issue, safety stock (SS) is presented to support the forecasting model. For each customer there are three different SS levels (0% , 5%, and 10%). Based on the company's expert, the cost ratio of underestimating the demand (shortage) to overestimating the demand (inventory cost) is 2 to 1. Using this information, the back order cost and inventory costs are assumed to be $10 and $5 respectively.

Table 10: The forecasted value against the actual value and cost associated with SS calculation for a customer.

	Forecasts		Difference			Inventory & Lost Order Cost		
	Actual	Model 1	0% SS	5% SS	10% SS	0% SS	5% SS	10% SS
Jan	15800	14279.2	-1520.76	-806.798	-92.836	7603.8	4033.99	464.18
Feb	15266	15015.42	-250.58	500.191	1250.962	1252.9	5001.91	12509.62
Mar	16923	15872.96	-1050.04	-256.392	537.256	5250.2	2437.99	17882.18
Apr	16597	15355.46	-1241.54	-473.767	294.006	6207.7	1149.84	20822.24
May	18200	16336.54	-1863.46	-1046.63	-229.806	9317.3	5233.165	18524.18
Jun	17565	16775.41	-789.59	49.1805	887.951	3947.95	491.805	27403.69
Jul	17878	15983.35	-1894.65	-1095.48	-296.315	9473.25	5231.51	24440.54
Aug	18381	17557	-824	53.85	931.7	4120	538.5	33757.54
Sep	16783	16347.66	-435.34	382.043	1199.426	2176.7	4358.93	45751.8
Oct	19208	16692.34	-2515.66	-1681.04	-846.426	12578.3	6225.75	37287.54
Nov	17631	16796.34	-834.66	5.157	844.974	4173.3	51.57	45737.28
Dec	15297	15911.4	614.4	1409.97	2205.54	6144	14151.27	67792.68
Total	205529	192923.1	-12605.9	-2959.72	6686.432	72245.4	**48906.23**	352373.47

Table 10 shows the safety stock calculation. First, the difference between forecasted model with different level of SS and actual data is calculated. Then, based on the mentioned cost, total cost for three different safety stock levels is calculated. The model with the lowest cost is selected. For example for Customer A, the model with 5% SS has the lowest total cost.

To sum up, in order to find the best forecasting model for each customer, the following algorithm is followed:

Algorithm 4 The best SARIMA model for a customer

Step 1. Iterate through all models with different configuration of p, q, P, and Q.

Step 2. Calculate the BICs and select the model with the lowest value

Step 3. Forecast the values for the last year.

Step 4. Calculate the cost fuction for different safety stock levels scenario using forecasted values and actual values

Step 5. Select the SS level with the cheapest cost function.

3.4.5 Decision Support System

In this section, we developed a Decision Support System (DSS) for the trucking company. The goal of this DSS is to facilitate the process of decision making and making it efficient. In this section, the current process of decision making is explained and then the architecture of DSS is outlined.

There are several experts who are responsible for making the decision of empty moves across the company. Each expert is responsible for a region. These experts gather field information about their region during the week such as the performance their forecast to the realized demand of that week, holiday of customer's factory, etc. For example if they under estimate the demand the previous week they will over estimate this week to catch up with back orders from the last week, or if they know a company's employees are on strike they deduct that demand from their estimated values. Using relevant information, each expert updates their spreadsheet. These spreadsheets are built by the experts themselves independent from each other. At the end of a working week, They will tweak their results based on their experience and hold a meeting. During this meeting, each expert announces their demand/supply of nodes in their region and some other expert will fulfill those request by either using the supply or satisfying the demand. The only degree of optimization observed is they usually estimate the distance in order to fulfill a shortage. After the meeting they will put in a "Moving Order" i.e. how many empty containers should reposition from one location to another one.

This process is vastly inefficient and time consuming. The forecasting process is mainly manual and the quality of it, is based on the experience of the expert. In the developed DSS, we tried to address several of these issues.

3.4.5.1 DSS Architecture

In this section, the DSS structure and its communication channels are explained. In order to have a maintainable and scalable DSS, object oriented concepts are utilized.

In order to define a network, the first class that is developed is ”Node”. This class is representative of each node in the network. At the time of initializing each node, the program will automatically forecast its demand/supply for the planning horizon. Two sub-classes of this class are ”Ramp” and ”Customer”. These two classes contain information specific to that type of ”Node”. ”Adjacency” is another class that defines all of the arcs on the network. There are two types of adjacency, it can be a Truck or Rail Adjacency. Truck adjacency is defined by the distance between two nodes. If the distance is less than a predetermined parameter, two nodes are accessible from each other by truck. If the company provides us with the cost of rail move between two nodes we assume those two nodes are adjacent. ”TimeLine” and ”TimeSlot” are responsible for holding the forecast value and types of the value (i.e. Demand or Supply).

”Network” is the main class that contains all of the company’s network. In addition to the elements of the network, this is the class that holds the mathematical model and its connection to CPLEX. ”Forecaster” class is responsible for all of the connections between the DSS and the R libraries and contains the forecasting methods. Figure 16 on page 67 shows the process flow of the DSS.

The User Interface (UI) of the DSS is developed in Visual Studio 2017. This part is coded in VB.NET and after getting all of the inputs from the user it will use the engine that is developed in JAVA. The different snapshot of the UI is shown in Figures 17, 18 and 19 on pages 67 and 68.

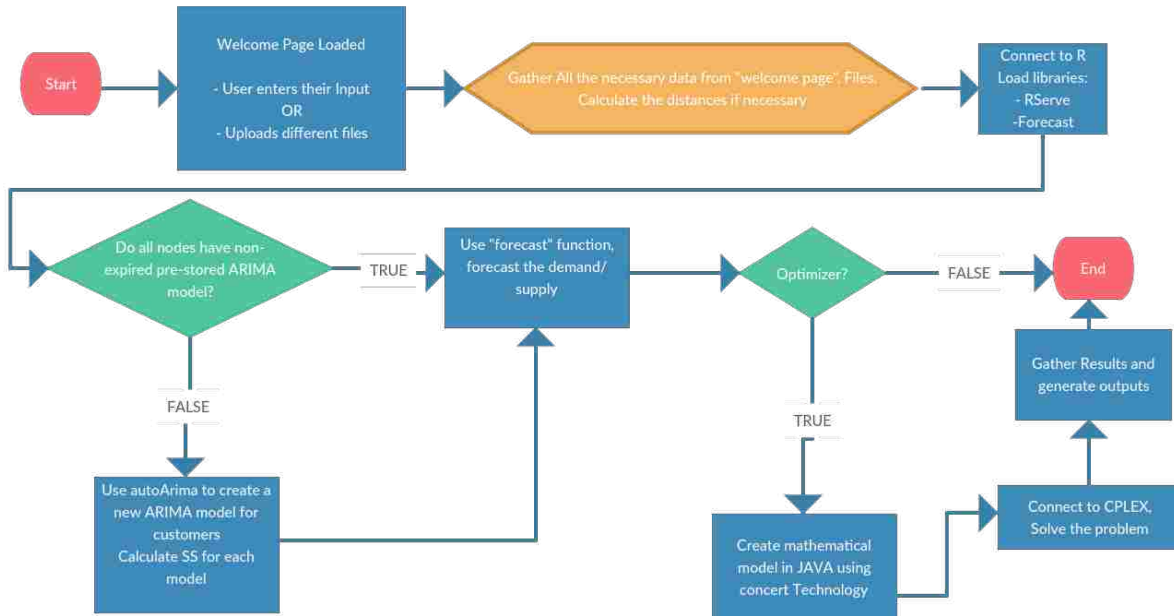


Figure 16: Process Flow of developed DSS.

Figure 17: First window of DSS before entering/uploading any input file.

In figure 17, Ramp Info, Customer Info includes the location of all rail ramps/customers, their demands and their yard capacities. Rail Cost and driving data can be uploaded or enter the cost per mile for each mode. In parameters section, the user could select to include customers or not. The planning horizon is determine by "# of Weeks". Run Forecaster will just run the RStudio to find the best ARIMA model for all the customers and ramps. Finally, The result of "Run Optimizer"

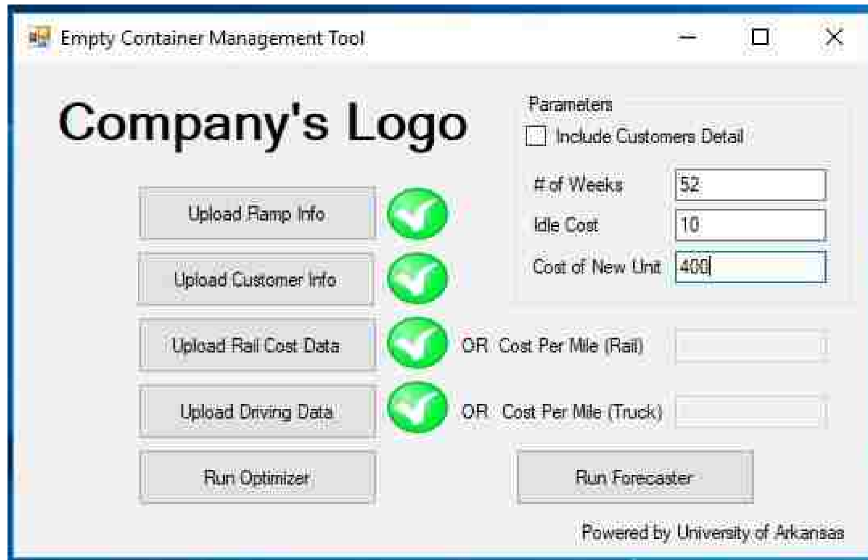


Figure 18: Window of DSS after entering/uploading any input file.

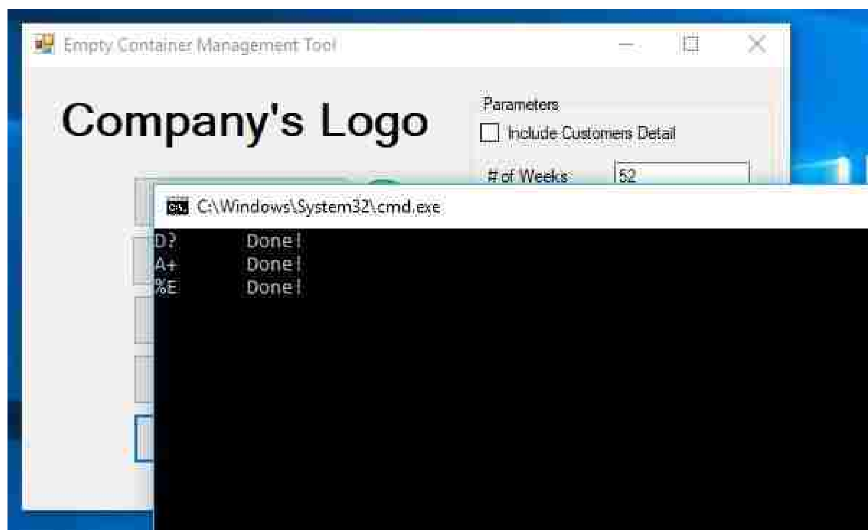


Figure 19: DSS Window while it is processing.

is obtained by CPLEX is placed in a spreadsheet for ease of use of the end user. The result is the number of moves between two nodes and the move's mode (i.e. Rail or Truck).

3.4.6 Case Study

In this section, we create three test cases for our DSS using the representative data obtained from the company. The following assumptions are made for all of the test cases.

- Range (2011 - 2013) used for forecasting 2014
- Yard capacity selected based on the data from the simulation data
- Cost of rail movement
  - Direct distance between two ramps is used
- Order cost added to the model to prevent sending small batch of containers.(i.e. economy of scale is used for different order size of container movement)

In the following we discuss the three test cases and DSS performance.

3.4.6.1 Rail Movement Only

In the first the test case, only rail movement is considered. As a result, the demands and supplies of all of the customers associated with a ramp are summed over and all truck movements removed from the mathematical model. In this case, the top 41 ramps from the representative dataset are selected (Ramps associated with the Top 25 customers in the representative dataset). Figure 20 shows the location of selected ramps. Note that some of the ramps appear to be out of the borders that is because of lack of precision associated with the representative dataset that we were provided with.

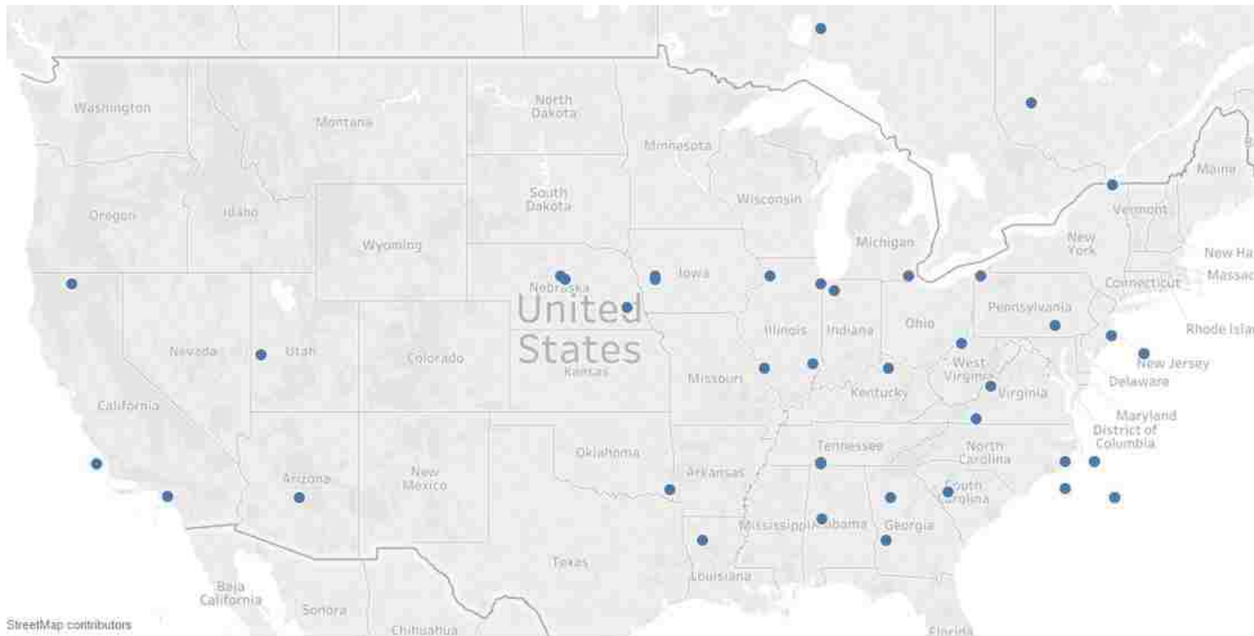
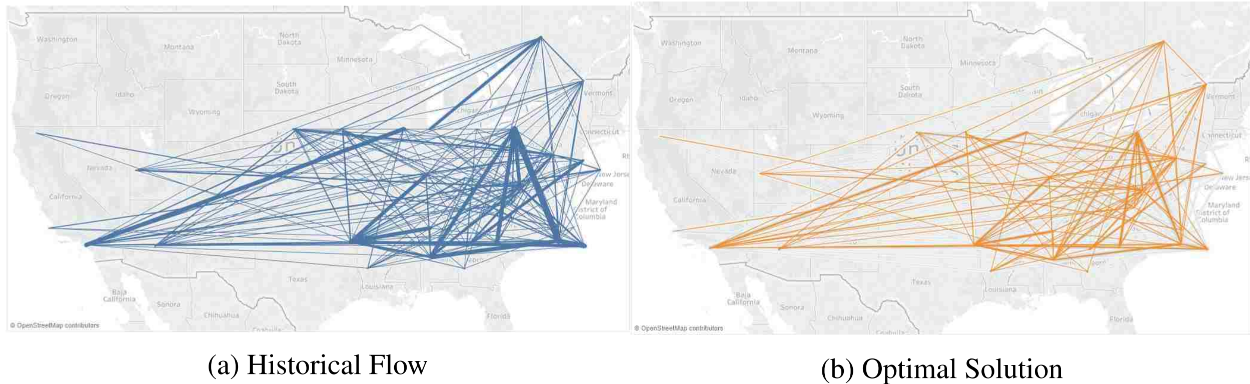


Figure 20: Location of 41 Ramps.

The model is solved optimally using CPLEX software. The historical data shows that 29247 empty moves happen between these 41 ramps. Our optimal solution maintain the balance of each ramp with only 6110 moves which is a 79% decrease. Based on the data, assuming the average cost of moving an empty container is $130, there is an opportunity to save $3M in operational cost. Although the result shows a large potential savings opportunity, the actual savings is significantly affected by the initial conditions. The initial condition is provided by the simulation and it does not reflect the exact situation encountered by the company. As a result, the company provided us with a representative snapshot of our initial date. Using that as an input for the model. Using this as an input for the model, CPLEX solved the model optimally with 17251 moves. This still shows 41% reduction (i.e. $1.55M) in empty moves. The other advantage of this test case is even if the size of problem reaches a point that the DSS cannot solve the problem optimally we can still solve it by aggregating nodes together.



(a) Historical Flow (b) Optimal Solution

Figure 21: The comparison between the historical flow of empty containers between ramps to optimal solution obtained from DSS.

3.4.6.2 Rail Movement and Truck Movement based on Zip Code

In this test case, we include the truck movements from the ramp to the customers' locations. In addition to all of our nodes from the previous test case, the smaller customers, in terms of volume, are also added to the problem. One of the obstacles of having smaller customers in the test case is the quality of the forecasting function. Since these customers are not ordering a container for most of the days, the forecasting function is less likely to predict their demands. Company experts suggest that we aggregate the customers based on their location into zip codes. As a result, each zip code will have several customers and the demand/supply of that zip code is the summation of all its customers. All the distances are calculated to the center of each zip code. In total, for the 41 Ramps, 110 Zip codes are considered. Based on the representative dataset, $94,472$ empty moves happen between these ramps and zip codes.

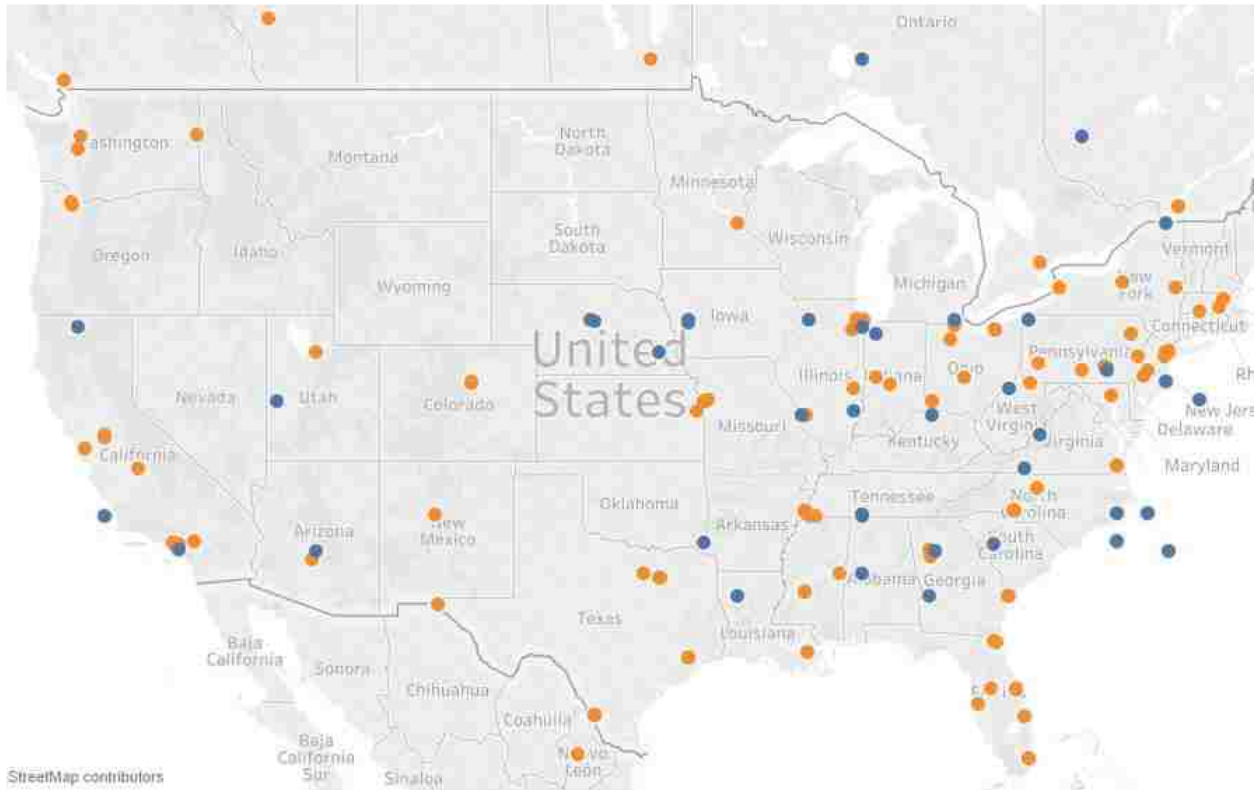


Figure 22: Location of 41 Ramps and 110 center of zip codes.

Note that some zip codes are far from a ramp. This happen because we are only considering the top 41 ramps and omit some of the ramps that are feeding those zip codes. This will affect our solution. The DSS can still solve the problem optimally using the snap shot provided by the company. The results suggests $67,918$ empty moves. This shows a potential reduction of 28% in the number of empty moves required to balance the network.

3.4.6.3 Rail Movement and Truck Movement with no zip codes In the last test case, we wanted to test the performance of the DSS with a larger dataset. In this test case, we consider each customer as a node. The previously mentioned problem of small customers in the earlier test case is an issue. Since our approach considers a SS for each customer, these customers will have some SS to avoid a back order. This test case considered 41 Ramps and 480 customers. Figure 23 illustrate the location of nodes.

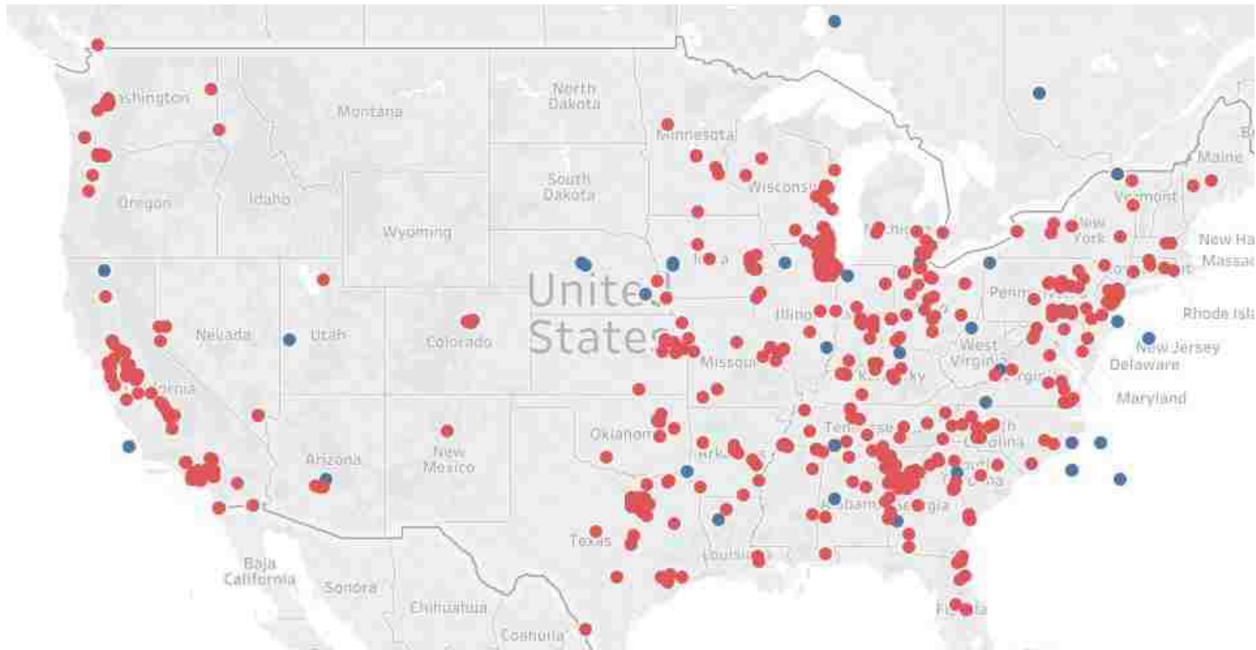


Figure 23: Location of 41 Ramps and 480 customers.

Since the planning meeting takes an hour in the company, we set an hour as time limit for CPLEX. The DSS terminates by reaching the time limit. The optimality gap reported by CPLEX at 1 hour time limits is 31.17%. The solution consists $115,072$ empty moves. The historical data shows $134,960$ moves, which means a potential 15% reduction. In addition to the size of test case, which inhibits the results not being optimal, the small customers lead to more empty moves due to their SS. We also increased the time limits of CPLEX to be 2 hour and 4 hour to see how much improvement can be obtain by more computational time. The optimality gap for these two new time limits are 24.58% and 22.16%. Based on our test cases, we believe the approach of clustering customers based on zip codes performs best, it is worth mentioning that our collaborators were interested in this case as well. Figure 24 shows the relative volume of each node of the system for the year 2014 in our representative dataset.

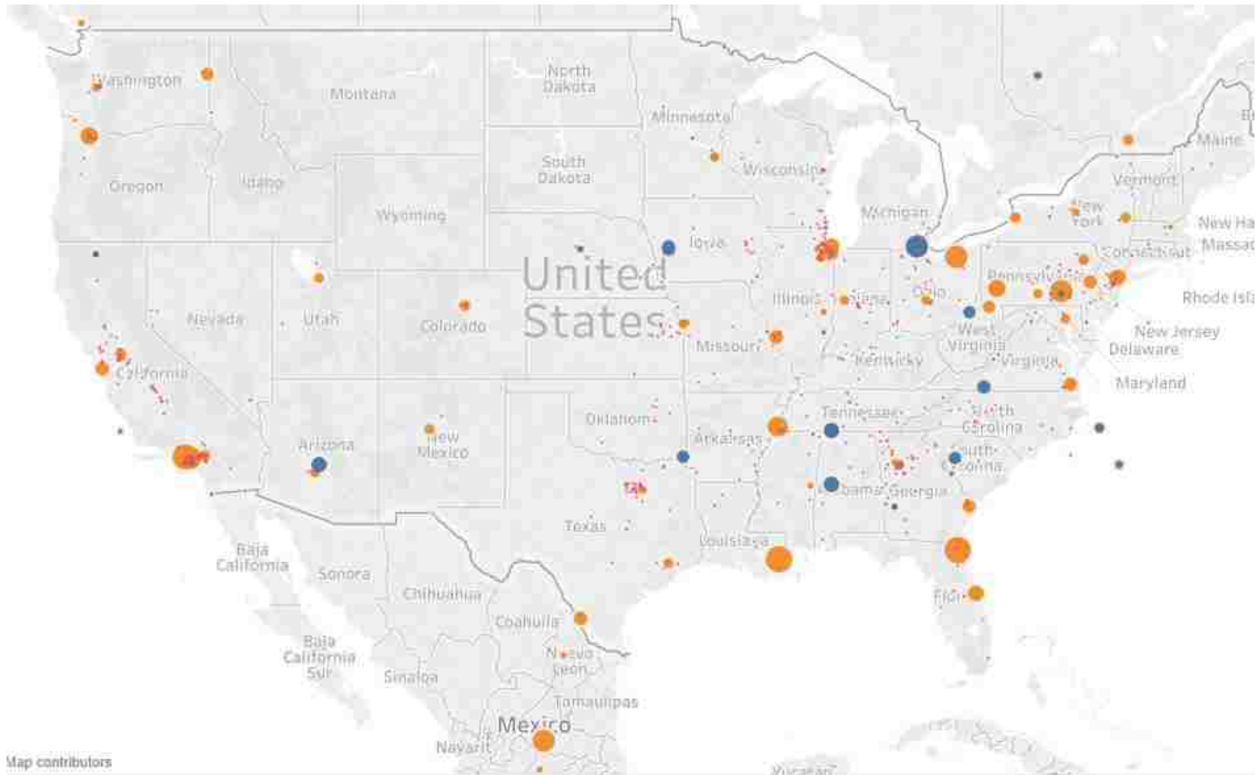


Figure 24: Empty Container Demand Volume of each node on the system in 2014.

3.5 Future Work and Contribution

In this chapter we try to address a traditional problem which is widely studied in the literature. However, in our literature review, we showed that the problem of inland empty container managements is not studied as much as maritime empty container management. In addition to that, the problem is not studied in an intermodal environment. Also, based on our observation and interaction with our industry partners process, the companies are not utilizing this knowledge in their daily decision making. Right now, this process is completely manual. Our analysis illustrates the impact of utilizing this knowledge and customizing it for this company which highlights the importance of this chapter. There are several other approaches that can be an extension of this study. First, develop a unique solution methodology or heuristic for this specific problem to provide a better/faster solution for larger instances. Second, consider other aspects of the real world

problem such as chassis and bobtail moves. Finally, explore the use of some newer AI and Machine Learning techniques on the large data set available.

4 Designing a Supply Chain Network for TB response in high-risk infected countries

Currently, the diagnostic process for TB takes several weeks. In a high-risk infected country, this lengthy process leads to higher mortality rates caused by TB. In this research, we are investigating several approaches to redesign the logistics of the diagnostic process utilizing a new technologically advanced device. These new methods could reduce the process from weeks to several hours. The following section is an overview of TB and its current diagnostic process.

4.1 Introduction

Tuberculosis (TB) is a disease caused by a bacteria called Mycobacterium tuberculosis (Mtb). More people die by active TB in comparison to other infectious illnesses.It is estimated that more than 1 billion people are infected with Mtb [64].

Quick diagnosis as well as getting a fast suitable treatment is of importance in treating Mtb, since there is a higher probability of mortality during the first few weeks of diagnosis [4], [32], [9], [62]. An obstacle in diagnosis of this disease is that conventional laboratory identification may take more than 10 weeks. Another obstacle is that the high risk infected countries do not have the necessary laboratory infrastructure to support rapid evaluation and response. They must use a solid culture method instead of liquid method which leads to even slower growth of bacterium. As a result the final drug susceptibilities are diagnosed way after the important early months [35]. On the other hand, applying empirical treatments may cause unpredictable health issues.A new test has been developed in which the diagnosis of TB and drug resistance are determined based on studying single nucleotide mutations (SNPs) [96], [25], [69]. Some of these methods require expensive instruments and complex laboratory processes (MTBDRplus) [96], while others prepare results in about two hours (Xpert MTB/RIF) [65], [16], [83]. Infrastructure issues in potential countries often force Xpert MTB/RIF or MTBDRplus laboratories to be far from the infected population. Considering the issues above, Point-of-care testing (POC) is of importance in order to dignose and propose the best treatment on the first visit [66]. A new device, GeneXpert

Omni, is a portable device capable of diagnosing on the first visit. With its simplicity, rapidness, and accuracy, the GeneXpert Omni device is supposed to be applicable for a POC test as it does not require that the test be done in a TB Microscopy Center which is often far from the patients. There are three main delivery scenarios for use with this new technology. POC devices at clinical centers, a mobile laboratory with the device in it (TB-OnDemand) and locate them in the existing TB Microscopy Center. The second option works by having a fleet of delivery vehicles carrying many of the GeneXpert Omni devices. A fleet gets notified when a clinical patient is detected, goes and picks the sample and uses the device for the required test, sends the test results to the clinic. For such systems, the vehicle position, devices readiness, and number of devices in each fleet needs to be determined. In order to do this several influencing parameters must be considered: TB incidence rate in a city, location of clinics, network and road pathways. Based on these, we look to explore the required number of vehicles, number of devices per each vehicle, whether to choose between moving fleets or waiting vehicles. Additional scenarios would not use a fleet but instead deploy the device either at clinics or at TB Microscopy Centers. Thus, the model dependency to the above parameters should be investigated.

4.2 Emergency Medical Service Literature review

The problem that we are studying in this chapter has similar characteristic to the Emergency Medical Service (EMS) and Vehicle Routing Problem (VRP). EMS studies the location and relocation of medical vehicles which have attracted special attention during decades of research. The EMS vehicle location problem deals with locating the available vehicles in some specific stations to cover all of the emergency service demands when the number of vehicles is restricted (Cordeau et al. [21]). There are three main categories for this problem: deterministic, stochastic and dynamic models.

On the other hand, the VRP is a well-studied problem. The only new characteristic that our problem introduces is the concept of time capacity on the vehicle. In the traditional VRP, the capacity is only considered as a physical capacity. Although in our problem the number of devices sets a

physical capacity on the number of samples that we can process, the capacity will be replenished by passing a specified period and not arriving at a depot.

4.2.1 Deterministic models

The first article in this area was introduced by Toregas et al [88]. They formulated a location set covering problem as an integer programming model. The objective minimizes the number of facilities to cover all of the demands in the network. Another early model for emergency vehicle location problem is Church and ReVelle [20]. They introduced the maximal covering location problem which maximizes coverage (demand points covered) within a stated service distance when the number of facilities are fixed. These two problems are static models, because they do not consider the stochastic nature of an EMS system. When a vehicle is dispatched to the demand location, it is not available anymore for other demand points. Another problem is when simultaneous demands occur and each of these should be covered. Daskin and Stern [29] introduced a hierarchical objective set covering problem (HOSC) for locating EMS vehicles. The objectives minimize the number of vehicles to cover all of the demand zones and maximize the multi-coverage of demands. Hogan and ReVelle [48] introduced two models for backup coverage in the location problem for emergency vehicles. Backup coverage handles high demand areas by covering demand points using two vehicles. Gendreau et al. [36] proposed a double standard model (DSM) that uses multiple ambulances to cover each demand point. In this model, any ambulance must cover all demands located within r_2 time units and a proportion α of the total demand within r_1 time units. This model is based on the standards defined in North America. This problem is solved using a Tabu search algorithm for randomly generated instances and real data obtained from the Island of Montreal. Tabu search was shown to provide near optimal solutions in a reasonable time.

4.2.2 Stochastic models

Stochastic models consider uncertainty arising from availability of the vehicles (when they are busy, the time of availability is unknown) or the time of service requested in the demand point. Daskin [27] proposed an expected covering location model by considering the probability of being busy for each vehicle to cover a demand. One year later, Daskin [28] formulated the maximal expected covering location model given a fixed number of vehicles. This model considers the possibility that all vehicles are busy when a new demand arrives. A heuristic algorithm is proposed to solve the larger problems in a reasonable time. Beraldi [11] proposed a robust model for locating emergency medical services. The problem deals with locating the vehicles in the demand points and finding the number of vehicles at each location in order to satisfy the demand using a chance constraint and minimizes the total cost. Reliability in this problem is defined as probability of covering an uncertain level of demand by a sufficient number of emergency vehicles. Cho et al. [18] studied the problem of locating trauma centers and helicopters as an emergency medical service vehicle simultaneously. The availability of helicopters is considered as a random parameter, thus the objective function maximizes the expected demand covered in less than one hour. Helicopters are assumed to be a substitute for a trauma center. Therefore, the problem is challenging because of the uncertainty and interdependent demands for trauma centers and helicopters. Two solution methodologies are considered in this article: a shifting quadratic envelopes method and a generalized Benders decomposition approach and they are applied to a real data set from Korea. The computational results revealed that these methods perform reasonably well in comparison with the solutions obtained from several benchmark heuristics. An et al. [5] proposed a scenario-based stochastic programming model as a mixed-integer linear program for emergency service facility location. They assumed that each emergency facility is subjected to disruption which is the uncertain parameter in the model and scenarios are generated base on the number of disrupted facilities at each location. The model determined the optimal location of facilities, patient to facility assignment in each scenario and traffic routing decision. The objective is minimizing the total expected cost of the system including facility set-up cost, travel cost and waiting

cost for the patients to receive a service in a facility. Then, they found lower and upper bounds to approximate the objective value which facilitated the development of a Lagrangian Relaxation approach to solve complex problems.

4.2.3 Dynamic models

The most recent problems in this area are Dynamic (multi-period) models that consider the changes in the system over time. Gendreau [37] considered the redeployment problem of ambulances which is relocating available ambulances to the potential sites when the new demands are requested. They proposed a dynamic programing approach to maximize the coverage of demand points and minimize the cost of movement of ambulances. A parallel tabu search heuristic is developed as a solution methodology and it is successfully applied on the EMS system of Montreal. Andersson and Varbrand [6] developed a new algorithm for a dynamic ambulance relocation problem. The objective is to minimize the maximum time of movement for the relocated vehicles between demand points. They introduce a new concept of *Preparedness* which states that there should be at least a certain level (P_{min}) at all demand points. This parameter measures the capacity of the EMS system to cover the demands at each zone and the vehicles are relocated when one or more zones have a preparedness less than P_{min}. A simulation model is used to demonstrate the effectiveness of the model in covering the demands with less waiting time. Rajagopalan et al. [67] formulated a dynamic model where the demand is uncertain and changes significantly over time. Their objective is to minimize the number of ambulances when a specified level of reliability for coverage of demand points should be satisfied. A reactive tabu search heuristic is proposed and a look ahead policy is used to reduce the computational time. The numerical results revealed that high quality solutions can be obtained in a reasonable time. Schmid and Doerner [72] model the ambulance location and relocation problem as a dynamic programming model in which ambulances need to be located in the specific locations in order to cover the requests in a reasonable time. Travel time is uncertain and time-dependent in this model. A variable neighborhood search (VNS) is proposed as the solution methodology and is tested on a

real data set from Vienna (Austria). The results show that the VNS method works very well in comparison with CPLEX.

4.3 Simulation

The complexity of the model and its stochasticity suggest the use of simulation. Simulation will helps us calculate different metrics under different scenarios and compare the effectiveness of each approach in terms of those metrics such as average diagnostic period. In the first section we define the different scenarios. These scenarios consist of a base model and suggested alternative models. The base model is the current implementation. The scenarios are explained as follow:

4.3.1 Proposed Scenarios

4.3.1.1 Base Model

In this scenario, the current situation of the country is simulated to be used as a base model for all of the other comparisons. We describe the system briefly; patients (people who are suspicious of TB) are going to the nearest clinic, and a doctor examines them and gets a sample for further analysis. The clinic will collect all the samples and at the end of the day, send all the samples to the assigned microscopy center. The microscopy center is responsible for processing the samples and reporting back to the clinics with the result. The samples with the negative results will be sent to the major hospital for further processing which can take an extensive amount of time. Figure 25 shows the flow of information in this scenario.

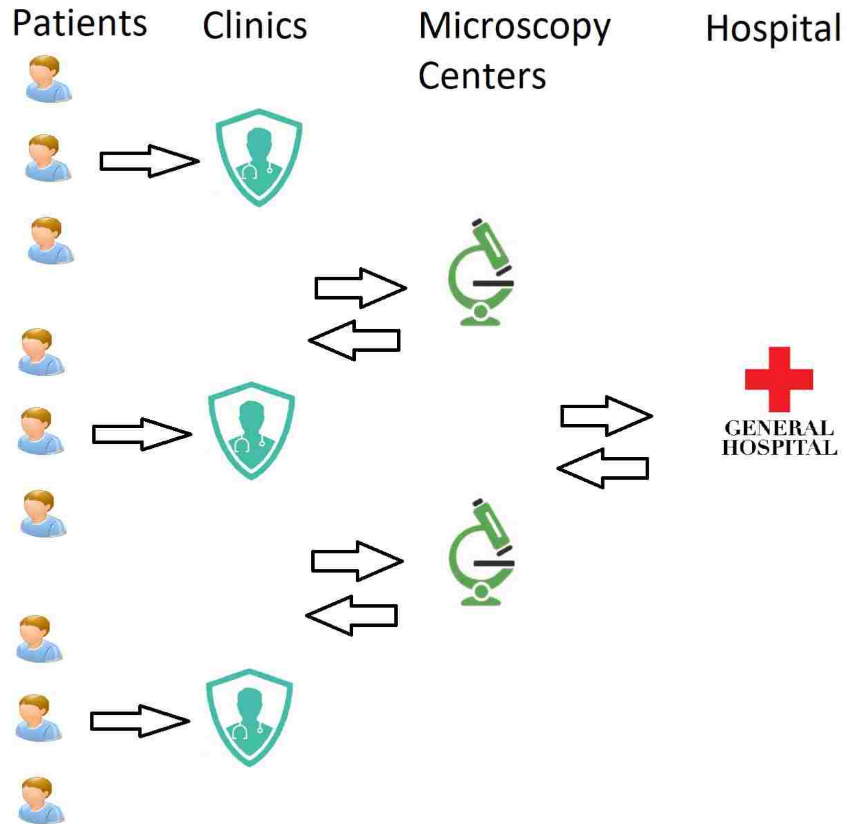


Figure 25: Current process of TB diagnostic

4.3.1.2 POC device at TB Microscopy Center

This scenario and the other three are proposed configurations that our collaborator would like us to compare to the current situation. In this scenario, each microscopy center will have a specified number of devices to process the samples. In this scenario, patients will provide the clinics with the samples. The purpose of this model is to have higher utilization for each device and reduce the diagnostic period by eliminating the hospital from the process. The following figure shows the flow of information in this scenario.

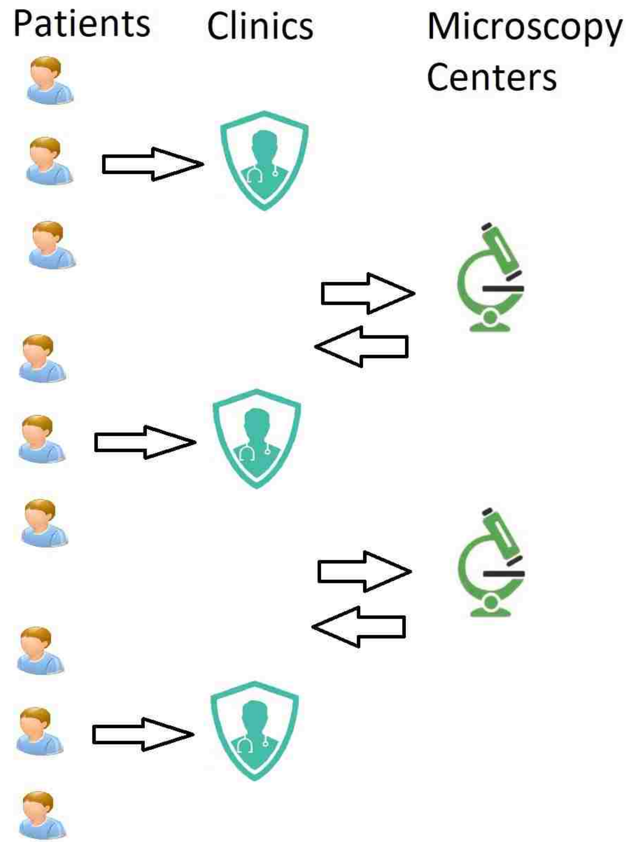


Figure 26: Current process of TB diagnostic

4.3.1.3 POC device at clinical centers

This scenario brings the device one step closer to the patients by having devices at the clinic level. The down side of this scenario is the low utilization on each device and the capital cost of providing each clinic with devices. On the other hand, this will eliminate the waiting time for samples to be sent to the centers. Figure 27 shows this scenario.

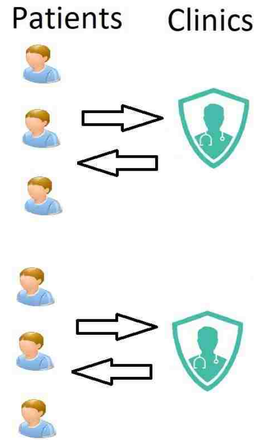


Figure 27: Current process of TB diagnostic

4.3.1.4 TB OnDemand

In this scenario, we examine the performance of the process by locating several vehicles on the planning horizon of the city. Each vehicle is equipped with several devices (The number of devices can be different from one vehicle to the other). As a patient requests a test, the nearest vehicle will be dispatched to the location of the patient to pick up the sample. The vehicle assignment can be done in several different ways. The following approaches will be considered for assignment: a. Nearest Assignment: The nearest available vehicle is assigned; this will reflect the time travel as well. In this context, if a busy vehicle can become free and drive to that location sooner than the nearest free vehicle, the system will assign the busy one to save time. b. Zone Assignment: The vehicles are assigned to a zone to cover that area's requirements. Each zone can have a different number of assigned vehicles.

In the following section, details of the simulation model and then simulation experiments are discussed.

4.3.2 Simulation Model

In this section, we explain the simulation model for each of the scenarios.

4.3.2.1 Base Model

In this configuration, using incident rate and exponential distribution, the appearance of first patient is created. At appearance of patient, its location is generated uniformly across the region. Each patient will select the nearest clinic to their location and will go there. After arriving at the clinic location, they will wait if there is no doctor available to visit them. After doctor's visit they leave the clinics. Clinics gather all patients samples and send them to the microscopy center to process. The microscopy center will process the samples. If a sample is positive they will notify the patient about the result if not, they will send the sample to a major hospital for further processing. The major hospital will examine the samples and send the final result to the patient. Figure 28 shows the process of this simulation.

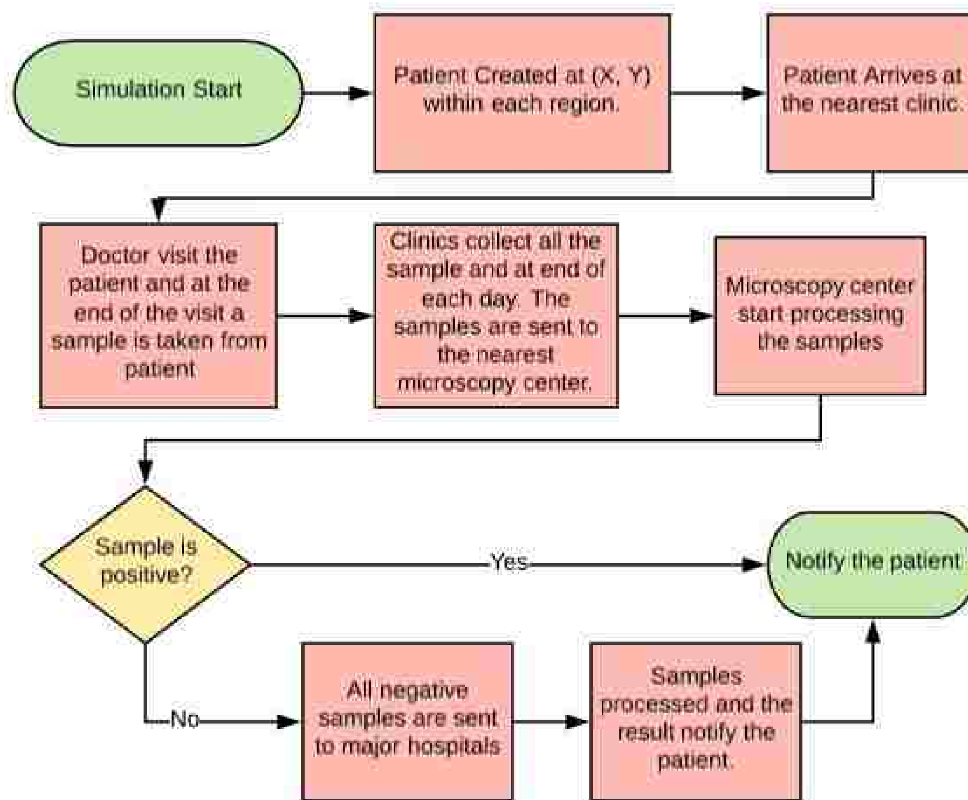


Figure 28: Simulation Process for the base model

4.3.2.2 POC at Microscopy Center

In this model, the process is identical to the base model except for when the samples arrives at microscopy centers, the TB device processes them and the final answer is sent to the patients. Figure 29 shows the process for simulation under this scenario.

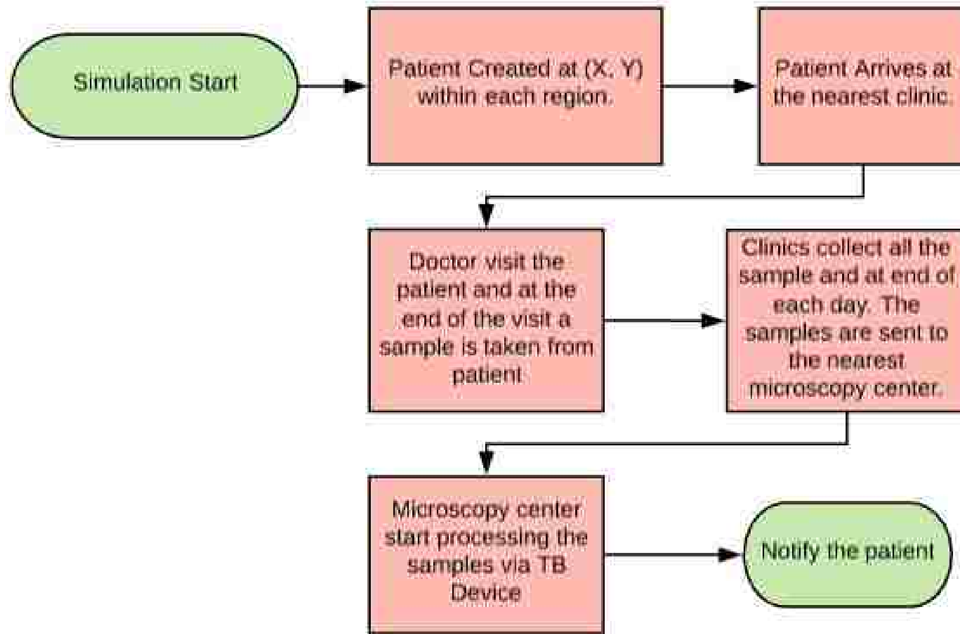


Figure 29: Simulation process for POC at Microscopy Center

4.3.2.3 POC at Clinics

In this scenario a patient visits the nearest clinic and after doctor visit. Their sample is processed using the TB device. Figure 30 shows the process for this simulation.

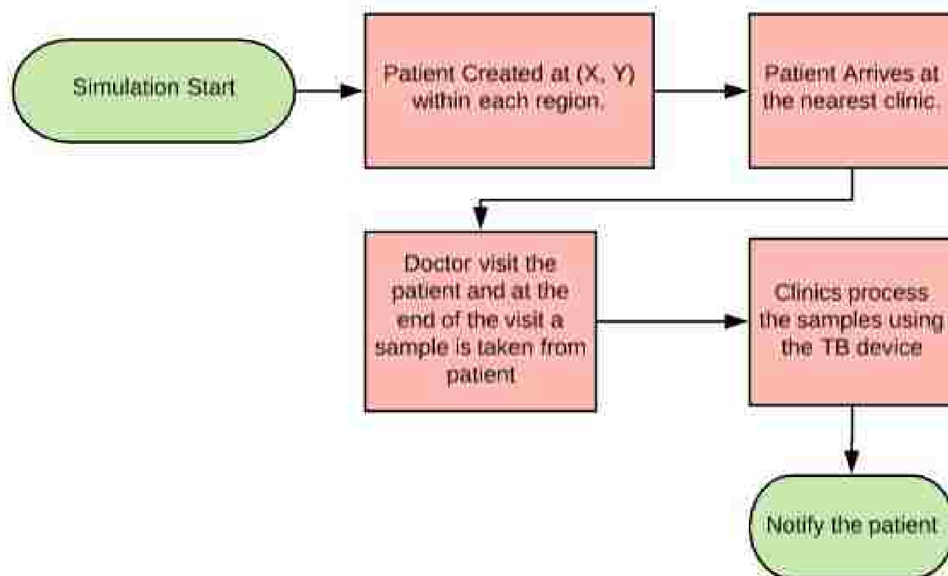


Figure 30: Simulation process for POC at Microscopy Center

4.3.2.4 TB On Demand

The last scenario is when the patient is not going to any clinics instead he/she will wait for a vehicle to collect its sample. In this scenario, patients notify the system and based on their location and assignment policy of vehicles, a vehicle is assigned to go to their place and collect their sample. As soon as their sample is collected, the TB device in the vehicle will process their sample and notify the patient.

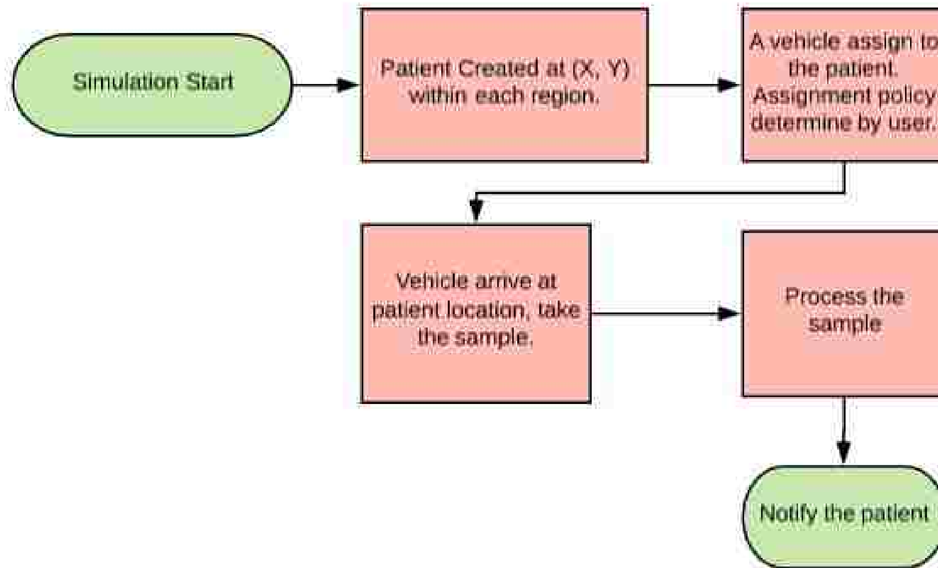


Figure 31: Simulation process for TB On Demand

4.3.3 Simulation Experiment

The simulation model is developed to compare the base model with the three different scenarios that our collaborator suggests. In this section, we will discuss our approach to model the problem and the different parameters that we explore.

The focus of the study is on Ghana, Africa, Fig 32 shows the general map of Ghana with its 10 different regions. The data that we obtained from our collaborator is aggregated to the region level and based on our discussion it will take years before we have the required data at the level of clinics, microscopy centers, and major hospitals. As a result, the simulation is developed at a

regional level.

Figure 32: Ghana's map with its different regions (adapted from [38])

4.3.3.1 Parameters

Four parameters are defined and their values must be determined for each region. In the following we will introduce these four parameters and elaborate how we evaluate them for each region.

- **Incident Rate**: The value of this parameter shows how frequent people develop TB symptoms in a region and go to a clinic for a doctor visit. This visit leads to providing the sample to the clinic and starting the diagnostic process. The exact value of this parameters is not available but based on the report "*The National Tuberculosis Health Sector Strategic*

Plan 2015-2020" provided by *Country Coordinating Mechanism of the Global Fund to Fight AIDS, Tuberculosis and Malaria*[1] (CCM Report), we know which regions reflect a higher rate of incident in comparison to the others. As a result, we have three different levels for the incident rate; High, Medium, Low. Fig 33 shows each region's incident rate. The distribution of the interval of two patients arriving at a clinic is assumed exponential with the rate of λ. The exact value of λ cannot be calculated with the current data set. We assume that the λ is in direct relation with the population of that region. Later, we will perform sensitivity analysis on the exact value of the λ.

- **Infrastructure Type**: As we discussed earlier, each clinic will send its samples to a microscopy center to retrieve results. Also, each microscopy center sends its negative result sample to a major hospital for further analysis. The time travel between each pair of facilities is another input to our model. This time travel is in direct relation to the distance and quality of infrastructure of that region. It is not possible to evaluate the infrastructure type for each region, but using the poverty level details, we can categorize the infrastructure type to three categories of Good, Medium, and Poor. Fig 34 shows the poverty level of each district. It gives us an idea about how to make assumptions about the Infrastructure type in each region. Based on the infrastructure types, three different vehicle speeds are defined. These numbers are based on the average speed of vehicles during different type of congestion in Ghana since it is impossible to conduct a survey in each region to find the average speed of vehicle for the purpose of this study. The high level of infrastructure assumes no congestion scenario, with an average speed of 44 MPH, the medium level is during normal traffic, with average speed of 16 MPH, and the low level is in the rush hour, with an average speed of 10 MPH. These numbers are obtained from a study of traffic congestion in Accra [3].

[1]"The Ghana Country Coordinating Mechanism (CCM) of the Global Fund to fight AIDS, Tuberculosis and Malaria was established in 2002 with an overall responsibility of managing Global Fund grants in the country. It is responsible for developing grant proposals, setting the policy framework for effective implementation of approved grants, exercising oversight of grants, monitoring progress towards meeting set targets and reviewing program performance"

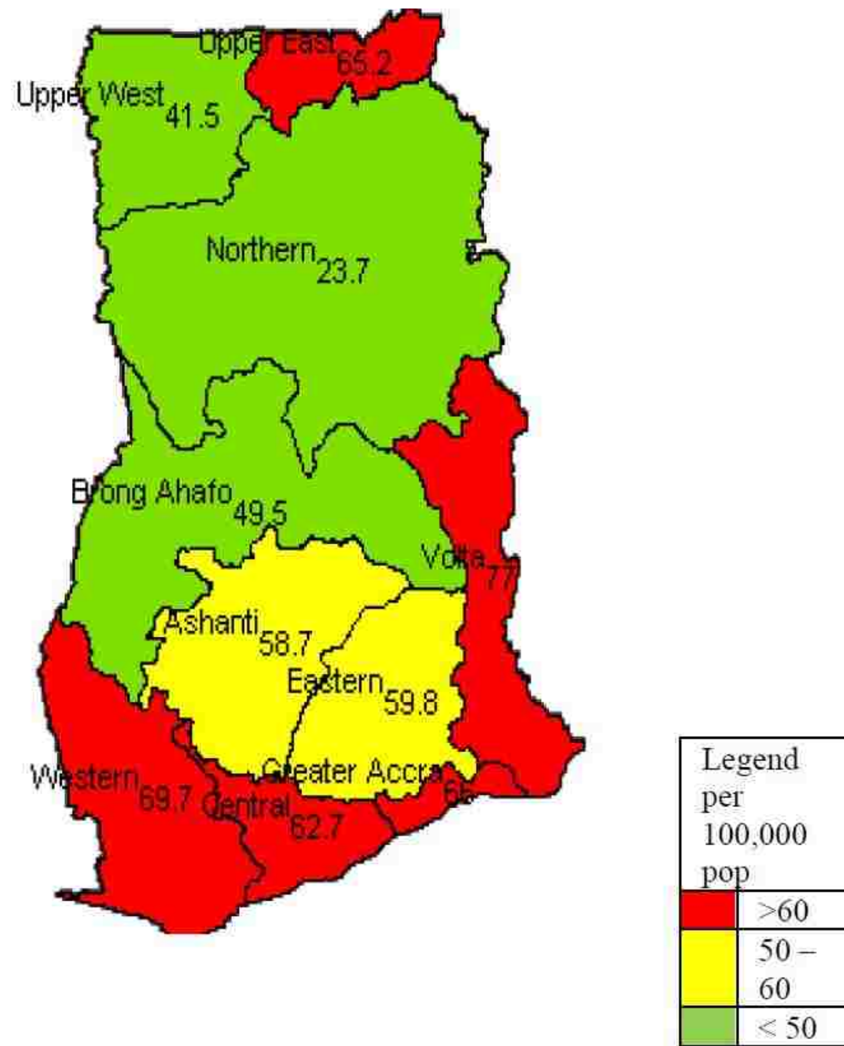


Figure 33: Ghana's map with its different regions (adapted from [38])

Table 11 shows Incident Rate and Infrastructure type of each region for our study. Each of them are an estimates based on figures 33 and 34.

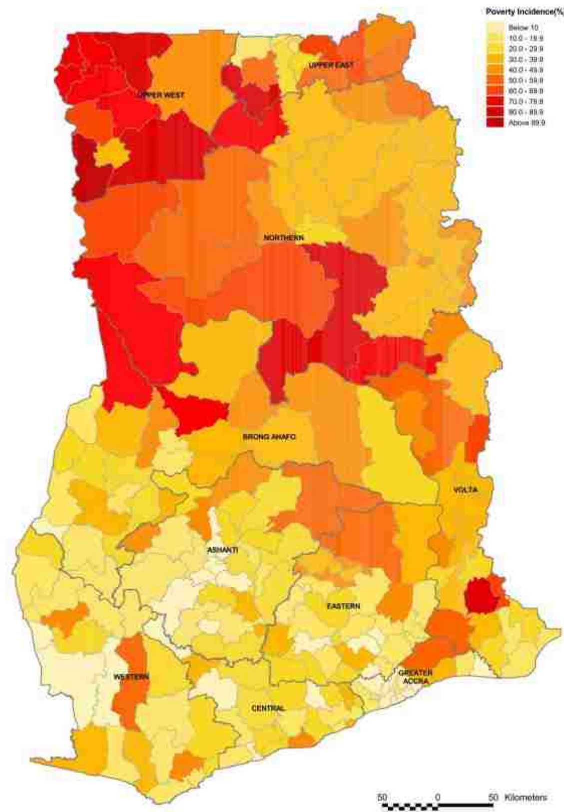


Figure 34: Poverty Level of Ghana's districts (adapted from [38])

Table 11: The incident rate and infrastructure type of each region

Regions	Incident rate	Infrastructure type
Ashanti	Medium	Good
Bring Ahafo	Low	Medium
Central	High	Medium
Eastern	Medium	Medium
Greater Accra	High	Good
Northern	Low	Poor
Upper East	High	Poor
Upper West	Low	Medium
Volta	High	Medium
Western	High	Good

- **Processing Time**: There are several categories of processing time in our model. We discuss each of them briefly. The first processing time is for a doctor visit at the clinic (Doctor Visit Process). This process is the time that a doctor spends to see if the patient has enough symptoms to be a candidate for providing a sample or not. Still, there is no concrete evidence about the length of this process, therefore in the next section we will perform sensitivity analysis to see the impact of such processing time. The next process is the time a microscopy center spends to get the first results of testing the sample (TB Center Processing). Based on the information we gained from our collaborator, it takes between 3 - 8 weeks for such testing results to get prepared. If the results of this test turn out to be negative, the sample is sent to a major hospital for further tests (Hospital Test). Again, this process takes between 2 - 3 weeks. These three processing times are the main ones in the traditional pathway to test TB in Ghana. In the newly proposed way, the new device can process the sample in only 90 minutes while there is no need for further testing of negative result samples.

- **Geographical Information**: This parameter consists of the number of clinics, microscopy, and hospitals and their locations. Based on the CCM Report, table 12 provides information on the number of clinics, microscopy centers, and major hospitals in each region.

Table 12: The geographical information of 10 regions.

Regions	Population	# Clinics	# Microscopy centers	# hospitals
Ashanti	4,725,046	79	50	0
Bring Ahafo	2,356,534	53	30	0
Central	2,107,209	55	33	0
Eastern	2,596,013	62	34	1
Greater Accra	3,909,764	123	60	2
Northern	2,468,557	37	28	0
Upper East	1,031,478	55	22	0
Upper West	677,763	21	10	0
Volta	2,099,876	64	33	0
Western	2,325,597	52	25	0
Total	24,297,837	601	325	3

Information on the exact locations are not provided except for the three major hospitals. In this part, we assume that all of these locations are uniformly distributed in their corresponding regions. Figure 35 demonstrates the location of microscopy centers in each location at the high level. Another difficulty in the simulation phase is defining the borders of each region and the country. As a result we simplify the shape of the country to a rectangle where each smaller rectangle corresponds to a region. The area of each region is in direct relation with the area of rectangles in the shape. Fig 36 shows the representative approximation of the country shape used for our model. The locations corresponding to patients are also generated using a uniform distribution since real data on that is not on hand. Note that the location of patients is only important for TBOnDemand scenario.

Our sensitivity analysis in the next section will also assess the level of dependency of our metrics to the location of clinics, microscopy centers, and hospitals.

Figure 35: Microscopy centers locations in Ghana

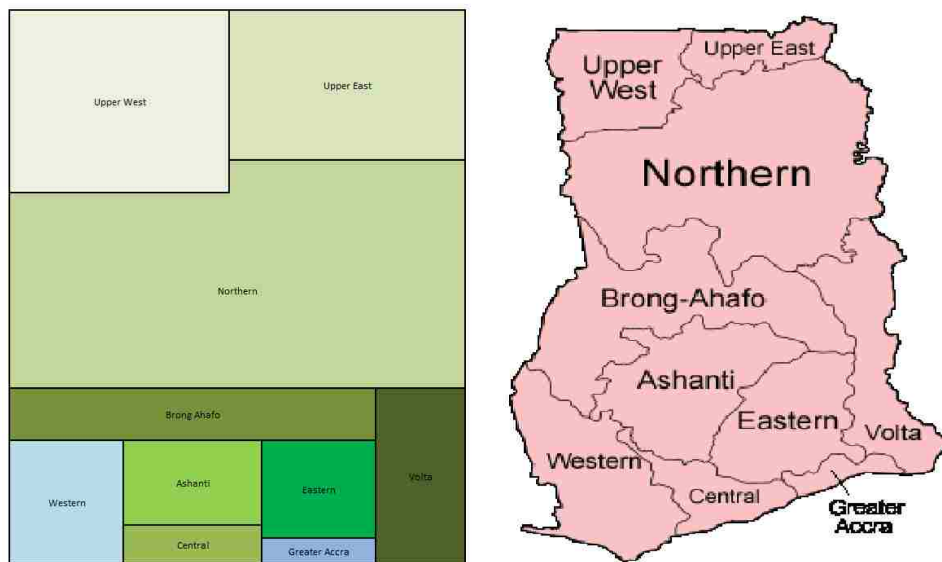


Figure 36: Approximation of Country Representation

Now, we define the two main outputs (KPIs) of our simulation. These KPIs are defined based on our discussion with our collaborator.

- **Average Diagnostic Period (ADP)**: This measures the average time in minutes from when a patient shows up at a clinic until the results are determined.
- **Number of diagnosed patient (NDP)**: This number measures the number of diagnostic

processes that have been completed during the simulation process.

4.3.4 Sensitivity Analysis of Simulation's parameters

In this section, we evaluate the impact of the parameters defined in the previous section on the KPIs and also determine the warm-up period and required number of replications for each experiment. Note that all of our experiments used a Replication Method for a period of 52 weeks. This means we will run N simulation with different random seeds and use those results to estimate the KPIs and their confidence intervals.

4.3.4.1 Warm up period analysis

In order to find the warm-up period, Welch's method is used. To create the Welch's plot, we used 5 different seed numbers to execute 5 runs for 52 weeks of simulation run length, and took ADP as the studied output. Every 10 minutes we calculated the ADP for the model and used the average of these runs for each observation to generate the Welchś plot. As Fig 37 shows, ADP cannot be calculated for the first phase of the simulation since none of the patients has completed the diagnostic process during that time.
Based on the Welchś plot, we select 20 weeks ($201,600$ minutes) as a warm up period for our experiments.

4.3.4.2 Number of Replications

In order to find the number of replication, we need to do a pilot study. For this study, we use 10 replications with 20 weeks as warm-up period.

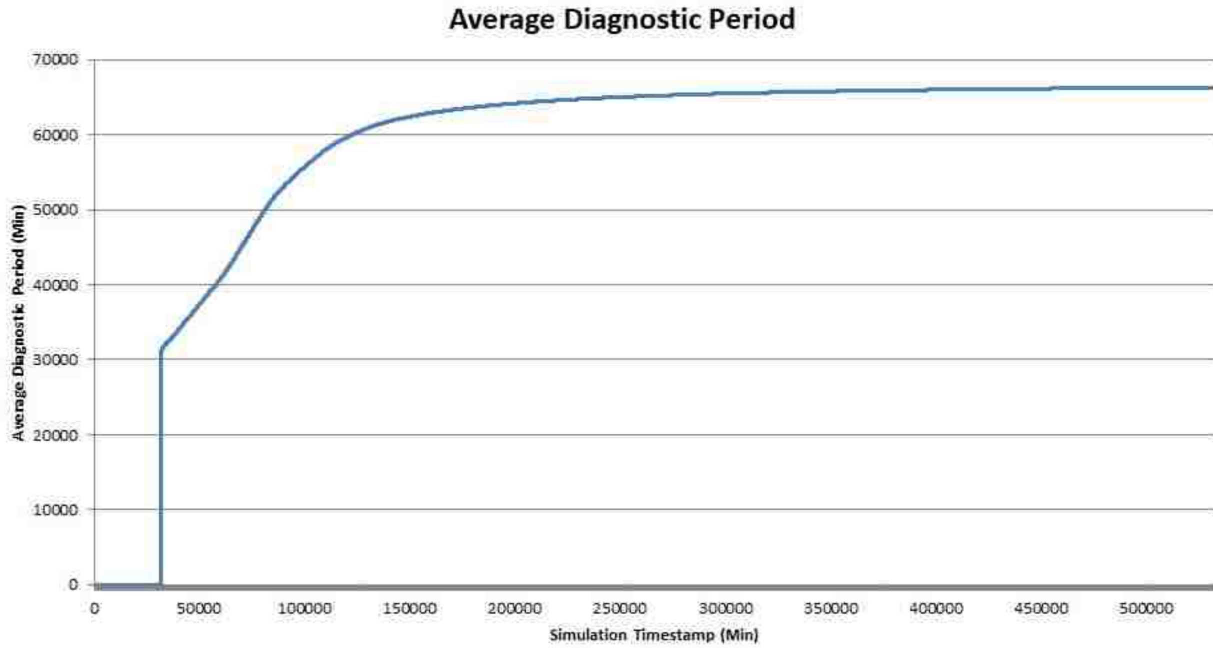


Figure 37: Welchś Plot based on 5 runs for Average Diagnostic Period (ADP)

Table 13: ADP Values in the pilot study.

Observation	ADP (minutes)
1	67094.59
2	67142.07
3	67170.81
4	67107.01
5	67114.92
6	67048.1
7	67155.27
8	67121.14
9	67145.05
10	67171.34

Using the confidence interval formulation, we determine the number of replications to construct the required confidence interval for ADP at 99% confidence level within ± 1 hour. It can be shown

that n can be obtained from the following formula:

$$n \geq (\frac{t_{\alpha/2;n-1} * S}{\varepsilon})^2$$
$$n \geq (\frac{3.249836 * 38.14263}{60})^2$$
$$n \geq 4.26$$

The number of replications is chosen to be 5. We will use 5 replications to construct any confidence interval. In order to find significant factors in our simulation, we will do a full factorial design with four factors and three levels. In the following we explain each factor and their levels.

4.3.4.3 Incident Rate Analysis

The incident rate is the rate that a patient shows up with TB symptoms at a clinic. This rate is directly related to the number of people in that region. Since we do not have actual incident rate by region, to calculate this parameter, we make a couple of assumptions. The arrival of patients has an exponential distribution with the rate λ. This rate is defined as a specific percentage of population of that region. Figure 33 on page 91 shows that the number of diagnosed TB in green region is less than 50 per $100,000$ which is 0.05% of the population. For yellow zone is between 0.05% and 0.06% and for red region is more than 0.06%. Therefore, Assume the diagnosed TB rate of 0.05%, 0.055%, and 0.06% for each type of region. We are testing if incident rate in each region is 5 times, 10 times, and 15 times higher than diagnosed TB rate. Thus, the incident rate (λ) for each region is calculated using the following formula:

$$\lambda_{Region} = X * DiagnosedTBRate_{Region} * Population_{Region}$$

where X is the magnifier and DiagnosedTBRate of each region is the percentage of actual TB cases in that region.

4.3.4.4 Infrastructure Type

The type of the infrastructure is another factor considered in our simulation model. A summary of the speed range for each region is provided in the following table. The speed for each trip is uniformly generated based on the region type.

Table 14: Speedś bounds for each region type in the simulation model

Region Type	Minimum Speed (MPH)	Maximum Speed (MPH)
Good	42	46
Medium	12	20
Poor	8	12

In this analysis, we evaluate what would happen to ADP if we increase the speed by 50% and reduce it by 50%.

4.3.4.5 Doctor evaluation process

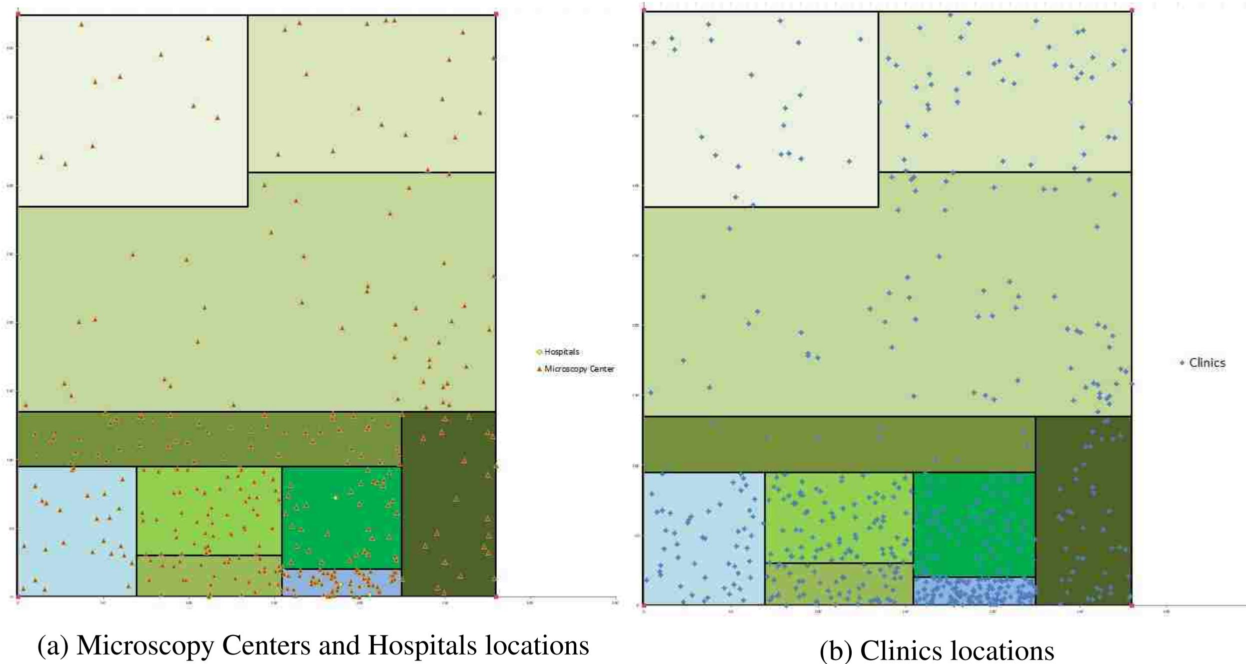
Another parameter which we consider the sensitivity of the model upon, is the time a doctor spends to visit a patient and order a test. Young in [93] the median of patient visit is 35.8 minutes. Based on this number, we introduce three different categories. These categories that we test for are represented in the following table:

Table 15: Different Doctor Visit bounds

Visit Type	Minimum Time (min)	Maximum Time (min)
Fast	5	20
Medium	20	40
Slow	40	60

4.3.4.6 Geographical Location

The last parameter that we will evaluate is the location of the clinics, microscopy centers, and hospitals. We used three different seed numbers to create three sets of locations. Fig 38 shows microscopy centers, hospitals, and clinics as an instance set.



(a) Microscopy Centers and Hospitals locations

(b) Clinics locations

Figure 38: An example of different facilities' locations in the simulation model

4.3.4.7 Factorial Design

In order to evaluate the significance level of each of the discussed parameters. We create a full factorial design with 10 replications. The random seed stays constant among different levels but varies for the same level tests. Before analyzing the result of the factorial design, we investigate the assumptions by evaluating the residual plots. Figure 39 shows residual plots.

It is apparent that there is no shift, trend, or cycle in the residuals and they follow normality assumption as well. Now, we can analyze the result of design. The results shown in figure 41 are achieved by doing factorial design in Minitab Software:

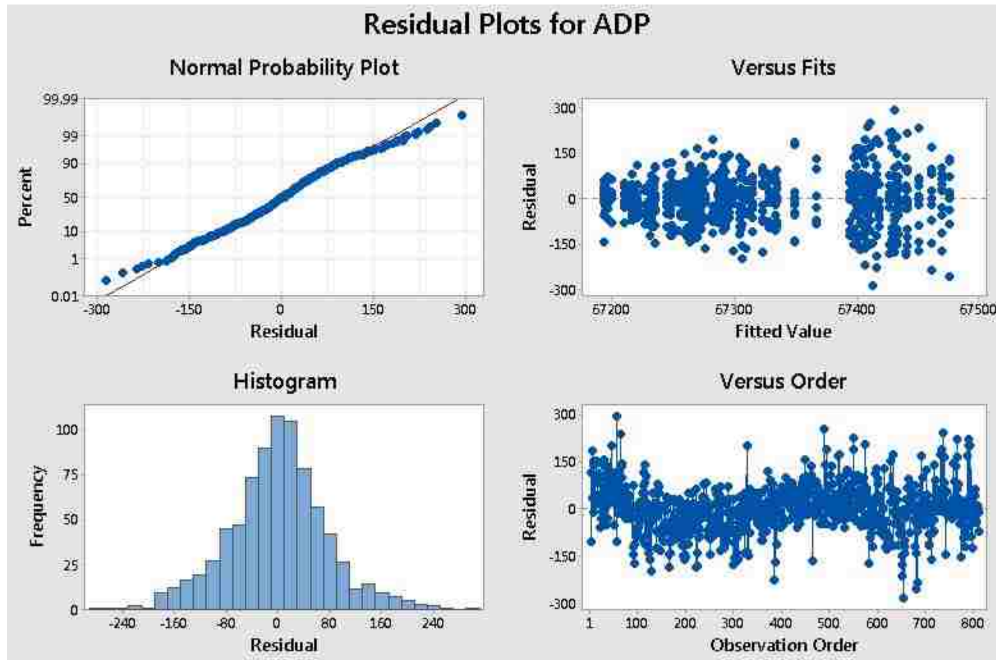


Figure 39: Different Plots on the Residuals

Factor Information

Factor	Levels	Values
Incident_Rate	3	0, 1, 2
Speed	3	0, 1, 2
Doctor_Visit	3	0, 1, 2
Location	3	0, 1, 2

Analysis of Variance

Source	DF	Adj SS	Adj MS	F-Value	P-Value
Model	80	5133792	64172	9.77	0.000
Linear	8	4851607	606451	92.37	0.000
Incident_Rate	2	4594090	2297045	349.89	0.000
Speed	2	11856	5928	0.90	0.406
Doctor_Visit	2	234011	117005	17.82	0.000
Location	2	11650	5825	0.89	0.412
2-Way Interactions	24	116222	4843	0.74	0.815
Incident_Rate*Speed	4	10490	2623	0.40	0.809
Incident_Rate*Doctor_Visit	4	18767	4692	0.71	0.582
Incident_Rate*Location	4	15541	3885	0.59	0.669
Speed*Doctor_Visit	4	32171	8043	1.23	0.299
Speed*Location	4	26799	6700	1.02	0.396
Doctor_Visit*Location	4	12453	3113	0.47	0.755
3-Way Interactions	32	129067	4033	0.61	0.955
Incident_Rate*Speed*Doctor_Visit	8	24200	3025	0.46	0.884
Incident_Rate*Speed*Location	8	27803	3475	0.53	0.835
Incident_Rate*Doctor_Visit*Location	8	36178	4522	0.69	0.702
Speed*Doctor_Visit*Location	8	40886	5111	0.78	0.622
4-Way Interactions	16	36896	2306	0.35	0.991
Incident_Rate*Speed*Doctor_Visit*Location	16	36896	2306	0.35	0.991
Error	729	4785966	6565		
Total	809	9919758			

Model Summary

S	R-sq	R-sq(adj)	R-sq(pred)
81.0254	51.75%	46.46%	40.44%

Figure 40: The Result of Minitab Software for full factorial design of 4 factors and 3 levels.

Based on the $p-value$, there are two significant factors, Incident Rate and Doctor visit time. In the next section, we want to compare the ADP of base model setting with TBOnDemand. In order to be conservative, we will choose the significant factors at a level where ADP is lower. We

will utilize such a strategy even for the insignificant factors. In this way, we consider the base model setting as competitive as possible. In order to see which levels has a lower ADP, the main effects plot is drawn.

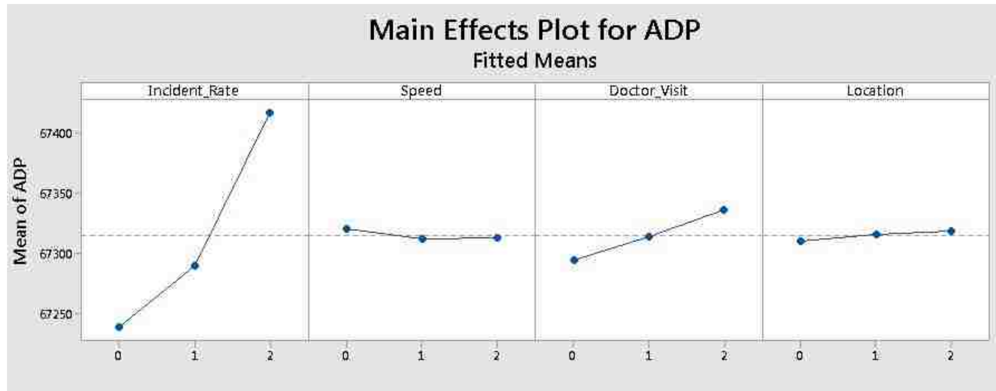


Figure 41: The main effects plot for the 4 factors.

The main effects plot suggests choosing the following parameter levels for the future analysis in this section:

Table 16: Parameters List for simulation model

Parameters	Value
Magnifier of Incident Rate	5
Doctor Visit Range	[5,20]
Speed Range Easy Region Type	[63, 69]
Speed Range Medium Region Type	[18, 30]
Speed Range Hard Region Type	[12, 18]
Location Random Seed	0

4.3.5 TBOnDemand

In this section, we investigate two factors for the TBOnDemand scenario; number of vehicles in each region, and number of devices per vehicle. These parameters mainly depend on the budget

of the government and the amount of investment. As a result, we provide our collaborator with different levels of service for different numbers of vehicles and devices in each region.

Another aspect of TBOnDemand that we should consider is the policy of assigning a vehicle to a patient (Assigning Policy). There are two policies suggested by the collaborator. First; the nearest Vehicle policy, which will assign a vehicle to a patient considering the travel time. Vehicle on the other hand, will choose a patient on its list depending on travel time and its devices' status. This means if the vehicle has at least one available device it will go to the nearest patient; otherwise, it selects a patient such that when it arrives at his/her location a device becomes available. The goal of this policy is to reduce the wait time at a patient's location. In the second policy, there is one difference. Vehicles are selected from the vehicle's pool corresponding to that specific region and not all regions. The average ADP of 5 replications for different number of vehicles and devices under different policies is provided below.

TBOnDemand Under the Nearest Vehicle Policy									
ADP Value	# Vehicle								
# Devices	1	2	3	4	5	6	7	8	9
1	28963.20	448.97	139.61	127.53	126.50	126.35	126.12	126.17	126.28
2	298.71	103.80	101.51	101.37	101.42	101.36	101.34	101.42	101.43
3	110.32	95.45	95.31	95.32	95.31	95.31	95.31	95.31	95.32
4	96.28	93.30	93.27	93.27	93.27	93.26	93.25	93.25	93.26
5	93.37	92.56	92.54	92.55	92.55	92.54	92.54	92.54	92.54
6	92.60	92.31	92.29	92.30	92.31	92.30	92.30	92.30	92.30
7	92.38	92.24	92.23	92.23	92.24	92.23	92.24	92.23	92.23
8	92.35	92.22	92.21	92.22	92.22	92.21	92.22	92.22	92.21
9	92.31	92.22	92.21	92.21	92.22	92.21	92.21	92.21	92.20

Figure 42: ADP Average over 5 replication for different number of vehicle and devices under Nearest Vehicle Policy

TBOnDemand Under the Region Base Vehicle Policy									
ADP Value	#Vehicle								
#Devices	1	2	3	4	5	6	7	8	9
1	66998.57	8360.48	132.70	125.71	125.11	125.08	125.03	125.04	125.03
2	8343.28	102.37	101.09	101.04	101.04	101.04	101.04	101.04	101.03
3	106.54	95.59	95.51	95.51	95.52	95.51	95.51	95.51	95.51
4	95.79	93.84	93.82	93.82	93.82	93.82	93.82	93.82	93.82
5	93.82	93.28	93.26	93.27	93.27	93.27	93.27	93.27	93.27
6	93.34	93.10	93.09	93.09	93.09	93.09	93.09	93.09	93.09
7	93.22	93.06	93.04	93.05	93.05	93.05	93.05	93.05	93.05
8	93.19	93.05	93.03	93.03	93.03	93.04	93.04	93.04	93.04
9	93.18	93.04	93.03	93.03	93.03	93.04	93.03	93.03	93.03

Figure 43: ADP Average over 5 replication for different number of vehicle and devices under region based Vehicle Policy

It is perceivable that the ADP value is more affected by the number of devices than by number of vehicles. The reason behind this, is each device can serve a patient and process a sample and it can free up while the vehicle is traveling to the next patient. On the other hand, if we have more vehicles and only one device on board, vehicles should wait at patient's location for a free device. The other observation from this experiment is ADP is much better under TBOnDemand. The other KPI in this analysis is NDP. Figures 44 and 45 shows the number of processed patient. As you can see when the number of vehicles is low, the two policies are almost performing identical in terms of NDP. The reasoning behind this behaviour is that the limited number of device/vehicle is the bottleneck for performance. Regardless of how we choose to get to the next patient, we should wait do to limited number of device. This leads to similar number of processed patients. We will analyze these observations more in the next section.

NDP Value	# Vehicle								
# Device	1	2	3	4	5	6	7	8	9
1	2867.4	5734.8	8601.8	11469.2	14336.2	17203.4	20070.8	22938	25805.2
2	5734.8	11468.8	17203.4	22938.2	28672.2	34407	40141	45875.4	51610
3	8601.8	17203.4	25805	34406.6	43008.6	51610.2	60211.8	68813.4	77414.6
4	11469	22937.8	34406.8	45875.6	57344.2	68813.4	80282	91750.8	91997.2
5	14336.2	28672.6	43008.4	57344.4	71680.2	86016.4	92228.2	92189.6	92188
6	17203.4	34406.8	51610.2	68813.2	86016.2	92350.4	92159.6	92124	92211.2
7	20071	40141.4	60212	80282	92259.4	92280.8	92420.8	92260.8	92213.6
8	22937.8	45875.8	68813.2	91750.6	92067.2	92215.8	92181.2	92068.8	92000.4
9	25805	51610.2	77414.8	92080	92202.6	92170	92277.2	92144.8	92076.8

TBOnDemand Under the Nearest Vehicle Policy

Figure 44: NDP Average over 5 replication for different number of vehicle and devices under Nearest Vehicle Policy

NDP Value	# Vehicle								
# Device	1	2	3	4	5	6	7	8	9
1	2867.8	5734.4	8601.8	11469.4	14336.6	17203.6	20071	22938	25805
2	5734.8	11469	17203.4	22937.8	28672.6	34406.8	40140.8	45875.4	51609.8
3	8601.8	17203.6	25805.2	34406.6	43008.6	51610.2	60211.6	68813.2	77414.6
4	11469.2	22937.8	34406.4	45875.6	57344.2	68813.2	80282.2	86398.8	86382.2
5	14336.2	28672.2	43008.6	57344.6	71680.6	85981	86465.2	86294.8	86561.6
6	17203.4	34406.4	51610	68813.2	86016.4	86274.8	86346.4	86638.6	86192.4
7	20070.6	40141	60211.4	80281.8	86419.2	86440.4	86259.6	86298	86274.6
8	22937.8	45875.4	68813.6	86207.2	86192	86401.2	86300.6	86529.8	86296.4
9	25805.2	51609.6	77414.8	86500.2	86175.6	86413.8	86777.8	86302	86310.8

TBOnDemand Under the Region Base Vehicle Policy

Figure 45: NDP Average over 5 replication for different number of vehicle and devices under region based Vehicle Policy

4.3.6 Comparison of Base Model Setting and TBOnDemand

In this section, we will evaluate different base model settings against the TBOnDemand scenario. All simulation runs are using the parameters from the previous section. As for the TBOnDemand scenario, number of vehicles and number of devices can take the value of 1 and 2. We select the lowest value to evaluate the TBOnDemand in its worst case against base model settings. For the two other scenario, POC at clinics and POC at Microscopy centers, also we use 1 or 2 devices.

In order to have a conservative approach in comparing the base model and TBOnDemand, we choose a TBOnDemand under a policy to result in longer ADP, as a result the region based policy is selected based on the numbers in Figure 42 and 43

The random seed number is constant for an observation across the settings which means the same set of patients are appearing in the system. Note that the random seed is different from an observation to another one. 25 replications are executed for each setting. The following table lists and explains different settings.

Table 17: Setting description for simulation experiment

Setting	Explanation
Base Model	No Diagnostic Device is used.
MS_1	1 Diagnostic Devices are located at each Microscopy Center
MS_2	2 Diagnostic Devices are located at each Microscopy Center
C_1	1 Diagnostic Device is located at each clinic
C_2	2 Diagnostic Devices is located at each clinic
T_1_1	1 Vehicle per region with 1 device on board under Region based policy.
T_1_2	1 Vehicle per region with 2 devices on board under Region based policy.
T_2_1	2 Vehicle per region with 1 device on board under Region based policy.
T_2_2	2 Vehicle per region with 2 device on board under Region based policy.

The following table summarizes the result of 25 replications for each setting along with the number of devices in each scenario.

Table 18: Number of devices and Result of 25 replications for each setting

Setting	ADP (mins)	ADP (days)	# of Devices
Base Model	677337.19	46.76	0
MS_1	869.52	0.6	348
MS_2	834.73	0.57	696
C_1	104.04	0.07	558
C_2	102.52	0.07	1116
T_1_1	66901.59	46.45	10
T_1_2	8044.2	5.58	20
T_2_1	8060.8	5.59	20
T_2_2	102.28	0.07	40

From these results we can see that base model setting and T_1_1 are not performing well. On the other hand, T_2_2, C_1, and C_2 has the smallest ADP among other settings but T_2_2 performed better since it has only 40 devices comparing to 558 and 1116 devices for C_1 and C_2.

4.3.7 Simulation Conclusion

In this section, we evaluated different settings for TB diagnostic process via simulation. Our results showed that using the new device will improve the average diagnostic process except for one scenario. Based on the number of devices and the ADP, our suggestion is T_2_2 setting which utilizes 2 vehicles for each region each of which having 2 devices on board. It is important to note that the final decision of which setting to utilize is dependent on the budget and service level that our collaborator needs. In the next section, we will focus on the routing of these vehicles for one particular region and compare the performance with the two assignment policies of this section.

4.4 Mathematical Model

In this section, we want to evaluate the assignment policies of the previous section against the optimal assignment and routing. Since the area that we are studying is suffering from poor infrastructure, we can assume that online assignment is not a suitable approach. Online assignment needs additional communication devices and a center to solve the routing and dispatch vehicles. Since the performance of TBOnDemand is significantly better than the base model setting (0.07 days against 46.76 days), we make an assumption that we collect patients' notifications with the request for a vehicle on a daily basis. All these requests are accumulated for a day and at the end of the day, a system solves the routing problem. Based on its suggested solution, the system lets vehicles to serve the patients in the morning. This will add one day to the process but is still significantly faster than the base model setting. This will improve the vehicle's travel distance and reduce total service time for all patients.

We model the TB on Demand scenario as a mathematical model. This model has similar characteristic to classic vehicle routing problem (VRP). The main difference in our formulation is that the capacity will be updated over time. As a result there is no need for vehicles to return back to the depot and they can wait at the patientś location until the device becomes available and then take the patient sample. In this section, we introduce the notations and parameters followed by problem formulation. The results of the model will be illustrated afterwards with several examples.

4.4.1 Notation and Parameters

In our mathematical model, we have N as the set of all patients that we need to visit and take their samples. Let q be the number of vehicles. Each vehicle has m number of devices. The processing time of each device on a sample is denoted as T minutes. After that the device can process another sample. Let t_{ij} be the traveling time between node i and j. The goal of this model is to find shortest tour while visiting all of the patients.

We require four sets of decision variables for our model formulation. Letting X_{ij} equal 1 if a ve-

hicle traverses from node i to node j and 0 otherwise, we denote Z_{ie} as the time capacity left on a device when the vehicle arrives at location i. Also we let Y_{ie} equal 1 if the sample of patient i is assigned to device e and 0 otherwise. Let S_i be the amount of time that a vehicle should wait at node i so that at least one of its devices becomes available, we formulate the problem and related constraints in the next section.

4.4.2 Model

In this section, we formulate the problem as a mixed integer programming problem.

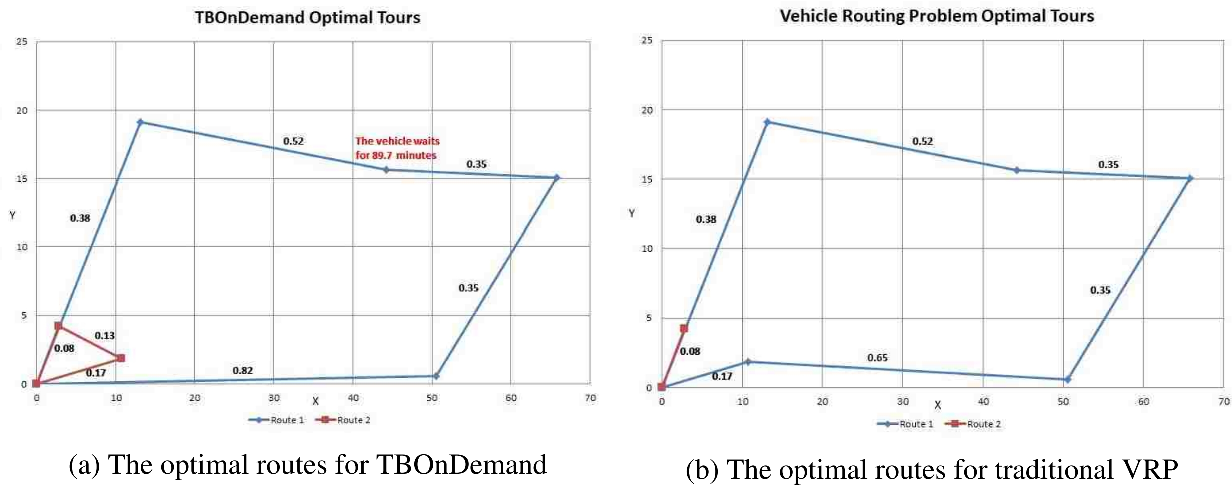
$$\min_{X,S} \quad \sum_{i,j} t_{ij} X_{ij} + \sum_i S_i$$

s.t.

$$
\begin{array}{rrcll}
(1) & Z_{ie} & \geq & T * Y_{ie} & \forall i \in N, \forall e \in \{1..m\} \\
(2) & Z_{ie} - T * Y_{ie} + t_{ij} + S_j & \geq & Z_{je} - M * (1 - X_{ij}) & \forall i, j \in N, \forall e \in \{1..m\} \\
(3) & \sum_j X_{ij} & = & 1 & \forall i \in N \\
(4) & \sum_i X_{ij} & = & 1 & \forall j \in N \\
(5) & u_i - u_j & \geq & d_j - C(1 - x_{ij}) & \forall i, j \in N/\{0\} \\
(6) & u_i & \leq & C - d_i & \forall i \in N/\{0\} \\
(7) & \sum_e Y_{ie} & = & 1 & \forall i \in N \\
(8) & Z_{0e} & = & T & \forall e \in \{1..m\} \\
(9) & Z_{ie} & \leq & T & \forall i \in N \\
(10) & X_{ij} & \in & \{0,1\} & \forall i, j \in N \\
(11) & Y_{ie} & \in & \{0,1\} & \forall i \in N, \forall e \in \{1..m\} \\
(12) & Z_{ie} & \geq & 0 & \forall i \in N, \forall e \in \{1..m\} \\
(13) & S_i & \geq & 0 & \forall i \in N \\
(14) & u_i & \geq & 0 & \forall i \in N
\end{array}
$$

The objective function in this formulation aims to minimize the total time of traveling and waiting at patients' locations of all vehicles. Constraint set (1) is to make sure that we assign a sample to a device if it has enough time capacity. Constraint Set (2) is calculating the amount of time capacity left on each device from one node to the other node. This is based on the assignment of a sample to a device at node i and the time travel distance between two nodes i, j. Constraint Set (3),(4) are making sure that we visit each patient only one time. Constraint sets (5), (6) are MTZ sub-tour elimination constraint. Constraint set(7) is making sure at each node only one device is assigned to the sample. Constraint Set (8) sets the initial time capacity to T while Constraint set (9) makes sure a device does not violate its maximum capacity. Constraint Set (10),(11) correspond to the binary decision variables whereas constraint set (12)-(14) represent the non-negativity conditions.

Figure 46 is a demonstration of an instance of mathematical model result. The instance is generated with the same parameters as our simulation analysis.



(a) The optimal routes for TBOnDemand

(b) The optimal routes for traditional VRP

Figure 46: An example of TBOnDemand optimal solution vs traditional VRP

In figure 46a, vehicle picks up the first two patients' samples and starts processing them. When the driver arrives at third patient's location, both of the devices are busy so he waits for 89.7 minutes to have at least one available device and then it continues its route. The traveling time in the optimal solution of TBOnDemand is equal to 2.81 while it is 2.58 for VRP. In the next section,

we will evaluate the model performance in terms of traveling time and waiting time at patient's location against the simulation result (i.e. online assignment).

4.5 Optimal vs Simulation

In this section, several instances are generated using the simulation model. In the simulation model, only the nearest vehicle assignment policy is used, the instances are considering only one region so in several cases the performace of these policies are the same. Since the nearest vehicle assignment performed better based on NDP and ADP in the previous section, we choose to use that. Three key performance indicators for each instance are reported; The amount of time that they are used to serve all customers and come back to the depot, the amount of time that they wait at a patient's location, and their idle time. After that each scenario is solved by means of the mathematical model formulated in the previous section and the same KPIs are reported and compared with the two policies.

The simulation instances are for the T_2_2 setting and for only one day. Note that in this part, we are only interested in the vehicle performance and won't be studying the ADP. The goal is to obtain the daily plan for each vehicle. Table 19 represents the results of 150 instances. The time limit on CPLEX solution is 30 minutes.

Table 19: Result of CPLEX runs against simulation runs

# of Patients	Nearest Vehicle Policy				CPLEX			
	Total	Traveling	No Device	Idle	Total	Traveling	No Device	Idle
5	1226.34	17.42	0	1208.92	101.78	13.01	88.77	0.00
6	1228.97	16.51	0.8	1211.66	102.3	13.01	89.29	0.00
7	1186.2	15.19	10.32	1160.69	191.61	12.74	178.87	0.00
8	1218.64	17.21	10.96	1190.47	192.04	13.35	178.69	0.00
9	1409.94	15.93	10.06	1383.95	281.55	12.85	268.7	0.00
10	1336.18	15.76	13.01	1307.41	282.44	13.52	268.92	0.00
11	1418.58	17.03	20.01	1381.54	371.21	13.66	357.55	0.00
12	1513.82	16.28	21.97	1475.57	371.66	13.76	357.9	0.00
13	1571.39	16.71	29.56	1525.12	461.13	14.83	446.3	0.00
14	1321.37	15.19	31.38	1274.8	458.77	13.56	445.21	0.00
15	1503.85	16.71	58.32	1428.82	545.92	13.81	532.11	0.00
16	1242.61	14.81	12.95	1214.85	543.88	13.06	530.82	0.00
17	1856.84	17.92	75.43	1763.49	639.06	13.95	625.11	0.00
18	1297.44	13.82	1.62	1282	634.87	13.3	621.57	0.00
19	1957.1	17.3	63.79	1876.01	725.43	13.59	711.84	0.00
20	1287.32	13.99	8.93	1264.4	726.43	13.04	713.39	0.00

Note that first column of each policy, ”Total”, corresponds to the total time required to serve all patients. The ”Traveling” column represents the total traveling time of vehicles. ”No Device” column denotes the time that a vehicle is at the patient location but all of its devices are busy. ”Idle” time is the sum of the times that vehicles wait for the next patient to appear in the system. There are three major differences between the results gained from CPLEX and the nearest vehicle assignment policy.

First, the idle time in the CPLEX solution is always zero since the list of all patients are available at the beginning of planning horizon and therefore, it’s possible to serve all of them in an optimal

back to back way which leads to elimination of idle time. On the other hand, the assignment policy is happening at the moment, i.e. online, and as a result a vehicle becomes idle if there exists no patient to serve at the moment. This elaborates one of the advantages of a CPLEX solution. The other observation is related to the ”No Device” time. The CPLEX solution has higher value in comparison with online assignments. This again goes to the fact that, in the online assignment, usually there is significant amount of idle time between two patients which leads to have at least one available device to serve with. On the other hand, offline assignment will serve patients back to back so it will run out of devices more frequently.

The most notable observation which distinguishes offline method from online policies, is the total time needed for serving all the patients. Despite having higher ”No Device” time, the offline method serves patients in less total amount of time compared to online plicies. This advantage provides the opportunity of letting the vehicles be utilized for other purposes for the rest of the day.

4.6 Conclusion and Future Research

The goal of our collaborator is to implement this in Accra, Ghana in Africa. Since there has never been a POC test for TB, the implementation/deployment of GeneXpert Omni must fit within the existing healthcare infrastructure, TB diagnostic pathways and economic systems of the deployed location.

First, we quantified the value of each of the scenarios for our stakeholder in order to help them make better strategic decisions for developing a new POC test infrastructure in Accra. As we investigated the different scenarios, the TBOnDemand plan turned out to bring significant improvement results over the current process.

Two online and one offline assignment policies are introduced and investigated with their advantages and disadvantages discussed. As a result, a new model that has similar characteristic to vehicle routing problem is formulated. For this new VRP problem we have vehicles with a predetermined capacity which is the number of samples they can process at a time. However, the oc-

cupied capacity will be available after a certain amount of time. This is a new twist on the classic capacitated vehicle routing problem. The solution for this problem is obtained using CPLEX and the results are discussed. As a matter of fact, we showed that using the offline assignment policy can significantly save time on serving all patients.

5 Contribution and Conclusion

In this dissertation, we investigate different transportation problems. In each chapter, a transportation problem is introduced and addressed. We briefly outline the contribution of each chapter.

In the second chapter, we focused on designing a battery swap station network to transport commodities from suppliers to customers via UAV. This chapter addressed the limited flying range of UAVs. We developed a scenario-based stochastic programming model to satisfy the customer demand that allows constructing different types of BSS in the planning network. To address the computational complexity of this problem, a Tabu Search heuristic is developed as the solution methodology. To achieve a better performance, a dynamic programming is developed to generate the initial solution for Tabu Search algorithm. Multiple numerical tests are conducted using CPLEX and TS on a deterministic model (by considering only one scenario for demands) due to the complexity of the problem. Results show that, compared with CPLEX, TS generates significant improvement in running time and quality of solutions, especially for large problems.

In third chapter, we performed a literature review on a traditional problem, Empty Container Management. We showed that the problem of inland empty container management is not studied as much as maritime empty container management. In addition to that, the problem is not studied in an intermodal environment and limited number of research studied a problem with the size of our problem. Also, based on our observation and interaction with our industry partner process, the companies are not utilizing this knowledge in their daily decision making procedures. Right now, this process is completely manual. In this chapter, we first develop a forecasting model to predict the demand of empty containers on their network. Later, a DSS is developed based on the forecasting model and mathematical model to forecast and optimize the problem. Three case studies are solved with the largest instance having over 500 nodes. We showed the impact of utilizing this knowledge and customizing it for a major transportation company in the US which highlights the importance and applicability of this chapter.

Finally, In fourth chapter, first we studied diagnostic process of TB in Ghana in collaboration with our academia partner. Their goal is to implement a novel process in Accra, Ghana in Africa. First, we determine the difference between scenarios for the stakeholder in order to help them make better strategic decisions for developing a new POC test infrastructure in Accra. A Simulation model is developed that quantifies the impact of each scenario. As we investigated, the TBOnDemand plan turned out to bring significant improvement over the current process. TBOnDemand is a setting in which patients will request a vehicle to take their sample at their locations. Three assignment policies are introduced and investigated with their advantages and disadvantages discussed. As a result, a new variant of vehicle routing problem is formulated. For this variant of VRP problem we have vehicles with a predetermined capacity which is the number of samples they can process at a time. However, the occupied capacity will be available after a certain amount of time. The solution for this problem is obtained using CPLEX and the results are discussed. As a matter of fact, we showed that using the offline assignment policy can significantly save time on serving all patients.

6 References

[1] Amazon prime air. http://www.amazon.com/b?node=8037720011. Accessed: 2014-10-01.

[2] Inside googles secret drone-delivery program. Online. Accessed: 2014-10-01.

[3] Albert M Abane. Tackling traffic congestion in accra, ghana: a road user's perspective. *Journal of advanced transportation*, 27(2):193–206, 1993.

[4] Shama D Ahuja, David Ashkin, Monika Avendano, Rita Banerjee, Melissa Bauer, Jamie N Bayona, Mercedes C Becerra, Andrea Benedetti, Marcos Burgos, Rosella Centis, et al. Multidrug resistant pulmonary tuberculosis treatment regimens and patient outcomes: an individual patient data meta-analysis of 9,153 patients. *PLoS Med*, 9(8):e1001300, 2012.

[5] Shi An, Na Cui, Yun Bai, Weijun Xie, Mingliu Chen, and Yanfeng Ouyang. Reliable emergency service facility location under facility disruption, en-route congestion and in-facility queuing. *Transportation research part E: logistics and transportation review*, 82:199–216, 2015.

[6] Tobias Andersson and Peter Värbrand. Decision support tools for ambulance dispatch and relocation. *Journal of the Operational Research Society*, 58(2):195–201, 2007.

[7] Anantaram Balakrishnan, James E Ward, and Richard T Wong. Integrated facility location and vehicle routing models: Recent work and future prospects. *American Journal of Mathematical and Management Sciences*, 7(1-2):35–61, 1987.

[8] Denise Lindstrom Bandeira, João Luiz Becker, and Denis Borenstein. A dss for integrated distribution of empty and full containers. *Decision Support Systems*, 47(4):383–397, 2009.

[9] Mayara L Bastos, Hamidah Hussain, Karin Weyer, Lourdes Garcia-Garcia, Vaira Leimane, Chi Chiu Leung, Masahiro Narita, Jose M Penã, Alfredo Ponce-de Leon, Kwonjune J Seung, et al. Treatment outcomes of patients with multidrug-resistant and extensively drug-resistant tuberculosis according to drug susceptibility testing to first-and second-line drugs: an individual patient data meta-analysis. *Clinical Infectious Diseases*, page ciu619, 2014.

[10] H Bendea, Piero Boccardo, S Dequal, Fabio Giulio Tonolo, Davide Marenchino, and Marco Piras. Low cost uav for post-disaster assessment. In *Proceedings of The XXI Congress of the International Society for Photogrammetry and Remote Sensing, Beijing (China), 3-11 July 2008*, 2008.

[11] Patrizia Beraldi, Maria Elena Bruni, and Domenico Conforti. Designing robust emergency medical service via stochastic programming. *European Journal of Operational Research*, 158(1):183–193, 2004.

[12] Benoıt Bourbeau, Teodor Gabriel Crainic, and Bernard Gendron. Branch-and-bound parallelization strategies applied to a depot location and container fleet management problem. *Parallel Computing*, 26(1):27–46, 2000.

[13] Joana Cavadas, Gonçalo Homem de Almeida Correia, and Joao Gouveia. A mip model for locating slow-charging stations for electric vehicles in urban areas accounting for driver tours. *Transportation Research Part E: Logistics and Transportation Review*, 75:188–201, 2015.

[14] Chia-hui Chang, Lawrence W Lan, and Mee Lee. An integrated container management model for optimizing slot allocation plan and empty container repositioning. *Maritime Economics & Logistics*, 17(3):315–340, 2015.

[15] Hwan Chang, Hossein Jula, Anastasios Chassiakos, and Petros Ioannou. A heuristic solution for the empty container substitution problem. *Transportation Research Part E: Logistics and Transportation Review*, 44(2):203–216, 2008.

[16] Kai Chang, Weiping Lu, Junji Wang, Kejun Zhang, Shuangrong Jia, Fake Li, Shaoli Deng, and Ming Chen. Rapid and effective diagnosis of tuberculosis and rifampicin resistance with xpert mtb/rif assay: a meta-analysis. *Journal of Infection*, 64(6):580–588, 2012.

[17] Raymond K Cheung and Chuen-Yih Chen. A two-stage stochastic network model and solution methods for the dynamic empty container allocation problem. *Transportation science*, 32(2):142–162, 1998.

[18] Soo-Haeng Cho, Hoon Jang, Taesik Lee, and John Turner. Simultaneous location of trauma centers and helicopters for emergency medical service planning. *Operations Research*, 62(4):751–771, 2014.

[19] Sook Tying Choong, Michael H Cole, and Erhan Kutanoglu. Empty container management for intermodal transportation networks. *Transportation Research Part E: Logistics and Transportation Review*, 38(6):423–438, 2002.

[20] Richard Church and Charles R Velle. The maximal covering location problem. *Papers in regional science*, 32(1):101–118, 1974.

[21] Jean-François Cordeau, Gilbert Laporte, Jean-Yves Potvin, and Martin WP Savelsbergh. Transportation on demand. *Handbooks in operations research and management science*, 14:429–466, 2007.

[22] Teodor G Crainic, Michel Gendreau, Patrick Soriano, and Michel Toulouse. A tabu search procedure for multicommodity location/allocation with balancing requirements. *Annals of Operations research*, 41(4):359–383, 1993.

[23] Teodor Gabriel Crainic and Louis Delorme. Dual-ascent procedures for multicommodity location-allocation problems with balancing requirements. *Transportation Science*, 27(2):90–101, 1993.

[24] Teodor Gabriel Crainic and Gilbert Laporte. Planning models for freight transportation. *European journal of operational research*, 97(3):409–438, 1997.

[25] Pedro Eduardo Almeida Da Silva and Juan Carlos Palomino. Molecular basis and mechanisms of drug resistance in mycobacterium tuberculosis: classical and new drugs. *Journal of antimicrobial chemotherapy*, 66(7):1417–1430, 2011.

[26] Quang-Vinh Dang, Izabela Ewa Nielsen, and Won-Young Yun. Replenishment policies for empty containers in an inland multi-depot system. *Maritime Economics & Logistics*, 15(1):120–149, 2013.

[27] Mark S Daskin. Application of an expected covering model to emergency medical service system design. *Decision Sciences*, 13(3):416–439, 1982.

[28] Mark S Daskin. A maximum expected covering location model: formulation, properties and heuristic solution. *Transportation science*, 17(1):48–70, 1983.

[29] Mark S Daskin and Edmund H Stern. A hierarchical objective set covering model for emergency medical service vehicle deployment. *Transportation Science*, 15(2):137–152, 1981.

[30] Pierre J Dejax and Teodor Gabriel Crainic. Survey papera review of empty flows and fleet management models in freight transportation. *Transportation science*, 21(4):227–248, 1987.

[31] Massimo Di Francesco, Teodor Gabriel Crainic, and Paola Zuddas. The effect of multi-scenario policies on empty container repositioning. *Transportation Research Part E: Logistics and Transportation Review*, 45(5):758–770, 2009.

[32] Dennis Falzon, Neel Gandhi, Giovanni B Migliori, Giovanni Sotgiu, H Cox, Timothy H Holtz, Maria-Graciela Hollm-Delgado, Salmaan Keshavjee, Kathryn DeRiemer, Rosella Centis, et al. Resistance to fluoroquinolones and second-line injectable drugs: impact on mdr-tb outcomes. *European Respiratory Journal*, pages erj01347–2012, 2012.

[33] Humberto Florez. Empty-container repositioning and leasing: An optimization model. 1986.

[34] Inês Frade, Anabela Ribeiro, Gonçalo Gonçalves, and António Antunes. Optimal location of charging stations for electric vehicles in a neighborhood in lisbon, portugal. *Transportation research record: journal of the transportation research board*, (2252):91–98, 2011.

[35] M Gauthier, A Somoskövi, JL Berland, O Ocheretina, MM Mabou, J Boncy, C Gutierrez, G Vernet, and JW Pape. Stepwise implementation of a new diagnostic algorithm for multidrug-resistant tuberculosis in haiti. *The International Journal of Tuberculosis and Lung Disease*, 18(2):220–226, 2014.

[36] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. Solving an ambulance location model by tabu search. *Location science*, 5(2):75–88, 1997.

[37] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. A dynamic model and parallel tabu search heuristic for real-time ambulance relocation. *Parallel computing*, 27(12):1641–1653, 2001.

[38] CCM Ghana. The national tuberculosis health sector strategic plan 2015-2020. 2015.

[39] Diego A Giménez-Gaydou, Anabela SN Ribeiro, Javier Gutiérrez, and António Pais Antunes. Optimal location of battery electric vehicle charging stations in urban areas: A new approach. *International Journal of Sustainable Transportation*, 10(5):393–405, 2016.

[40] Christoph H Glock. Decision support models for managing returnable transport items in supply chains: A systematic literature review. *International Journal of Production Economics*, 183:561–569, 2017.

[41] Christoph H Glock and Taebok Kim. Container management in a single-vendor-multiple-buyer supply chain. *Logistics Research*, 7(1):112, 2014.

[42] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & operations research*, 13(5):533–549, 1986.

[43] Roy Godzdanker, Matthew J Rutherford, and Kimon P Valavanis. Islands: a self-leveling landing platform for autonomous miniature uavs. In *Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on*, pages 170–175. IEEE, 2011.

[44] Roy Godzdanker, Matthew J Rutherford, and Kimon P Valavanis. Improving endurance of autonomous aerial vehicles through intelligent service-station placement. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 3179–3184. IEEE, 2012.

[45] Moncer Hariga, Christoph H Glock, and Taebok Kim. Integrated product and container inventory model for a single-vendor single-buyer supply chain with owned and rented returnable transport items. *International Journal of Production Research*, 54(7):1964–1979, 2016.

[46] Fang He, Yafeng Yin, and Jing Zhou. Deploying public charging stations for electric vehicles on urban road networks. *Transportation Research Part C: Emerging Technologies*, 60:227–240, 2015.

[47] Gerhard Hiermann, Jakob Puchinger, Stefan Ropke, and Richard F Hartl. The electric fleet size and mix vehicle routing problem with time windows and recharging stations. *European Journal of Operational Research*, 252(3):995–1018, 2016.

[48] Kathleen Hogan and Charles ReVelle. Concepts and applications of backup coverage. *Management science*, 32(11):1434–1444, 1986.

[49] Akio Imai, Koichi Shintani, and Stratos Papadimitriou. Multi-port vs. hub-and-spoke port calls by containerships. *Transportation Research Part E: Logistics and Transportation Review*, 45(5):740–757, 2009.

[50] Min Ji and Jun Xia. Analysis of vehicle requirements in a general automated guided vehicle system based transportation system. *Computers & Industrial Engineering*, 59(4):544–551, 2010.

[51] Z Jiele. Empty container distribution problem. *UROP Report*, 6019, 1999.

[52] Hossein Jula, Anastasios Chassiakos, and Petros Ioannou. Port dynamic empty container reuse. *Transportation Research Part E: Logistics and Transportation Review*, 42(1):43–60, 2006.

[53] Dukwon Kim and Panos M Pardalos. A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters*, 24(4):195–203, 1999.

[54] Jonghoe Kim and James R Morrison. On the concerted design and scheduling of multiple resources for persistent uav operations. *Journal of Intelligent & Robotic Systems*, 74(1-2):479–498, 2014.

[55] Jonghoe Kim, Byung Duk Song, and James R Morrison. On the scheduling of systems of uavs and fuel service stations for long-term mission fulfillment. *Journal of Intelligent & Robotic Systems*, 70(1-4):347–359, 2013.

[56] Gilbert Laporte. Location routing problems. 1987.

[57] Seunghyeon Lee and James R Morrison. Decision support scheduling for maritime search and rescue planning with a system of uavs and fuel service stations. In *Unmanned Aircraft Systems (ICUAS), 2015 International Conference on*, pages 1168–1177. IEEE, 2015.

[58] David Levy, Kaarthik Sundar, and Sivakumar Rathinam. Heuristics for routing heterogeneous unmanned vehicles with fuel constraints. *Mathematical Problems in Engineering*, 2014, 2014.

[59] Jing-An Li, Stephen CH Leung, Yue Wu, and Ke Liu. Allocation of empty containers between multi-ports. *European Journal of Operational Research*, 182(1):400–412, 2007.

[60] Jing-An Li, Ke Liu, Stephen CH Leung, and Kin Keung Lai. Empty container management in a port with long-run average criterion. *Mathematical and Computer Modelling*, 40(1-2):85–100, 2004.

[61] Ilkyeong Moon and Hwajin Hong. Repositioning of empty containers using both standard and foldable containers. *Maritime Economics & Logistics*, 18(1):61–77, 2016.

[62] Olanrewaju Oladimeji, Petros Isaakidis, Olusegun J Obasanya, Osman Eltayeb, Mohammed Khogali, Rafael Van den Bergh, Ajay MV Kumar, Sven Gudmund Hinderaker, Saddiq T Abdurrahman, Lovett Lawson, et al. Intensive-phase treatment outcomes among hospitalized multidrug-resistant tuberculosis patients: results from a nationwide cohort in nigeria. *PloS one*, 9(4):e94393, 2014.

[63] Alessandro Olivo, Paola Zuddas, Massimo Di Francesco, and Antonio Manca. An operational model for empty container management. *Maritime Economics & Logistics*, 7(3):199–222, 2005.

[64] World Health Organization. *Global tuberculosis report 2016*. World Health Organization, 2016.

[65] World Health Organization et al. Policy statement: automated real-time nucleic acid amplification technology for rapid and simultaneous detection of tuberculosis and rifampicin resistance: Xpert mt. 2011.

[66] Nitika Pant Pai, Caroline Vadnais, Claudia Denkinger, Nora Engel, and Madhukar Pai. Point-of-care testing for infectious diseases: diversity, complexity, and barriers in low-and middle-income countries. *PLoS Med*, 9(9):e1001306, 2012.

[67] Hari K Rajagopalan, Cem Saydam, and Jing Xiao. A multiperiod set covering location model for dynamic redeployment of ambulances. *Computers & Operations Research*, 35(3):814–826, 2008.

[68] Ronald L Rardin. *Optimization in operations research*, volume 166. Prentice Hall Upper Saddle River, NJ, 1998.

[69] Timothy C Rodwell, Faramarz Valafar, James Douglas, Lishi Qian, Richard S Garfein, Ashu Chawla, Jessica Torres, Victoria Zadorozhny, Min Soo Kim, Matt Hoshide, et al. Predicting extensively drug-resistant mycobacterium tuberculosis phenotypes with genetic mutations. *Journal of clinical microbiology*, 52(3):781–789, 2014.

[70] Matthew A Russell and Gary B Lamont. A genetic algorithm for unmanned aerial vehicle routing. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1523–1530. ACM, 2005.

[71] Norberto Sáinz Bernat, Frederik Schulte, Stefan Voß, and Jürgen Böse. Empty container management at ports considering pollution, repair options, and street-turns. *Mathematical problems in engineering*, 2016, 2016.

[72] Verena Schmid and Karl F Doerner. Ambulance location and relocation problems with time-dependent travel times. *European journal of operational research*, 207(3):1293–1303, 2010.

[73] Gideon Schwarz et al. Estimating the dimension of a model. *The annals of statistics*, 6(2):461–464, 1978.

[74] WS Shen and CM Khoong. A dss for empty container distribution planning. *Decision Support Systems*, 15(1):75–82, 1995.

[75] Koichi Shintani, Akio Imai, Etsuko Nishimura, and Stratos Papadimitriou. The container shipping network design problem with empty container repositioning. *Transportation Research Part E: Logistics and Transportation Review*, 43(1):39–59, 2007.

[76] Lawrence V Snyder and Mark S Daskin. Reliability models for facility location: the expected failure cost case. *Transportation Science*, 39(3):400–416, 2005.

[77] Byung Duk Song, Jonghoe Kim, and James R Morrison. Towards real time scheduling for persistent uav service: A rolling horizon milp approach, rhta and the stah heuristic. In *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pages 506–515. IEEE, 2014.

[78] Dong-Ping Song and Jing-Xin Dong. Empty container management in cyclic shipping routes. *Maritime Economics & Logistics*, 10(4):335–361, 2008.

[79] Dong-ping Song and Jing-xin Dong. Flow balancing-based empty container repositioning in typical shipping service routes. *Maritime Economics & Logistics*, 13(1):61–77, 2011.

[80] Dong-Ping Song and Jing-Xin Dong. Cargo routing and empty container repositioning in multiple shipping service routes. *Transportation Research Part B: Methodological*, 46(10):1556–1575, 2012.

[81] Dong-Ping Song and Qing Zhang. A fluid flow model for empty container repositioning policy with a single port and stochastic demand. *SIAM Journal on Control and Optimization*, 48(5):3623–3642, 2010.

[82] M SteadieSeifi, NP Dellaert, W Nuijten, and T Van Woensel. A metaheuristic for the multimodal network flow problem with product quality preservation and empty repositioning. *Transportation Research Part B: Methodological*, 106:321–344, 2017.

[83] Karen R Steingart, Ian Schiller, David J Horne, Madhukar Pai, Catharina C Boehme, and Nandini Dendukuri. Xpert® mtb/rif assay for pulmonary tuberculosis and rifampicin resistance in adults. *The Cochrane Library*, 2014.

[84] Kaarthik Sundar and Sivakumar Rathinam. Algorithms for routing an unmanned aerial vehicle in the presence of refueling depots. *IEEE Transactions on Automation Science and Engineering*, 11(1):287–294, 2014.

[85] Koji AO Suzuki, Paulo Kemper Filho, and James R Morrison. Automatic battery replacement system for uavs: Analysis and design. *Journal of Intelligent & Robotic Systems*, 65(1-4):563–586, 2012.

[86] San-Nah Sze, Siaw-Ying Doreen Sek, Kang-Leng Chiew, and Wei-King Tiong. Simulation of empty container logistic management at depot. In *AIP Conference Proceedings*, volume 1863, page 050016. AIP Publishing, 2017.

[87] Sotirios Theofanis and Maria Boile. Empty marine container logistics: facts, issues and management strategies. *GeoJournal*, 74(1):51, 2009.

[88] Constantine Toregas, Ralph Swain, Charles ReVelle, and Lawrence Bergman. The location of emergency service facilities. *Operations Research*, 19(6):1363–1373, 1971.

[89] A Graf von Westarp and O Schinas. A fuzzy approach for container positioning considering sustainable profit optimization. *Transportation Research Part E: Logistics and Transportation Review*, 92:56–66, 2016.

[90] Owen Worley, Diego Klabjan, and Timothy M Sweda. Simultaneous vehicle routing and charging station siting for commercial electric vehicles. In *Electric Vehicle Conference (IEVC), 2012 IEEE International*, pages 1–3. IEEE, 2012.

[91] Yangyang Xie, Xiaoying Liang, Lijun Ma, and Houmin Yan. Empty container management and coordination in intermodal transport. *European Journal of Operational Research*, 257(1):223–232, 2017.

[92] Jun Yang and Hao Sun. Battery swap station location-routing problem with capacitated electric vehicles. *Computers & Operations Research*, 55:217–232, 2015.

[93] Richard A. Young, Sandra K. Burge, Kaparaboyna A. Kumar, Jocelyn M. Wilson, and Daniela F. Ortiz. A time-motion study of primary care physicians work in the electronic health record era. *Family Medicine*, 50(2):91–99, 2 2018.

[94] Won Young Yun, Yu Mi Lee, and Yong Seok Choi. Optimal inventory control of empty containers in inland transportation system. *International Journal of Production Economics*, 133(1):451–457, 2011.

[95] Bo Zhang, CT Ng, and TCE Cheng. Multi-period empty container repositioning with stochastic demand and lost sales. *Journal of the Operational Research Society*, 65(2):302–319, 2014.

[96] Y Zhang and WR Jr Jacobs. Mechanisms of drug action, drug resistance and drug tolerance in mycobacterium tuberculosis: expected phenotypes from evolutionary pressures from a highly successful pathogen. *Handbook of tuberculosis. Weinheim: Wiley-VCH Verlag*, 2008.

A Chapter 2 Programming Codes

A.1 Tabu Search Class

```
package phdalgorithm;
import ilog.concert.IloException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
/**
 *
 * @author aghahari
 */
public class Heuristic {
    Solution CurrentSolution;
    DataReader Data;
    String CurrentCandidate;
    String CurrentGrade;
    String[] PotentialCandidate;
    String TabuInsert = "";
    String TabuRemove = "";
    String TabuUpgrade = "";
    int TabuSize;
    public Heuristic(DataReader InputData) {
        Data = InputData;
    }
    public Solution InitialAll(int Num, String R) throws CloneNotSupportedException, IloException, IOException {
        AI Baran = new AI(R);
        String WC = Baran.Run();
        return new Solution(WC.split("!")[0], WC.split("!")[1], Data, "");
    }
    public Solution Run(int Iteration, int SizeNeighbor, int TabuSize1, int NCandidate, String Name) throws CloneNotSupportedException,
           IloException, IOException {
        TabuSize = TabuSize1;
        long myTime = System.currentTimeMillis();
        CurrentSolution = InitialAll(NCandidate, Name);
        myTime = System.currentTimeMillis() - myTime;
        System.out.print(CurrentSolution.ObjectiveValue + "\t" + myTime);
        myTime = System.currentTimeMillis();
        PotentialCandidate = Data.getListType(DataReader.PointType.Candidate);
        Solution MinSolution = null;
        double MinSoFar = Double.MAX_VALUE;
        for (int i = 0; i < Iteration; i++) {
            List<Solution> Neighbors = new ArrayList<Solution>();
            for (int j = 0; j < SizeNeighbor; j++) {
                Neighbors.add(PMInsert(CurrentSolution));
                Neighbors.add(PNInsert(CurrentSolution));
                Neighbors.add(TPInsert(CurrentSolution));
                Neighbors.add(Swap(CurrentSolution));
                Neighbors.add(Upgrade(CurrentSolution));
                Neighbors.add(Downgrade(CurrentSolution));
                Neighbors.add(Remove(CurrentSolution));
                Neighbors.add(RemoveSmart(CurrentSolution));
```

```
            Neighbors.add(RandomInsert(CurrentSolution));
        }
        double Min = Double.MAX_VALUE;
        for (Solution S : Neighbors) {
            if (S.ObjectiveValue < MinSoFar) {
                MinSoFar = S.ObjectiveValue;
                CurrentSolution = S.clone();
                MinSolution = S.clone();
                break;
            } else if (Min > S.ObjectiveValue) {
                if (TabuValid(S.TabuMove)) {
                    Min = S.ObjectiveValue;
                    CurrentSolution = S.clone();
                }
            }
        }
        ChangeTabu(CurrentSolution.TabuMove);

    }
    myTime = System.currentTimeMillis() - myTime;
    System.out.println("\t" + MinSoFar + "\t" + myTime);
    return MinSolution;
}
public Solution Initial() throws CloneNotSupportedException, IloException {
    CurrentCandidate = "";
    CurrentGrade = "";
    return new Solution("", "", Data, "");
}
public Solution Initial(String Candidate, String Type) throws CloneNotSupportedException, IloException {
    return new Solution(Candidate, Type, Data, "");
}
public Solution Swap(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    String Current = S.Candidates + "-";
    String NeighBSS = "";
    String DelBSS = "";
    if (!S.Candidates.equals("")) {
        DelBSS = S.Candidates.split("-")[Data.R.nextInt(S.Candidates.split("-").length)];
        for (String C : PotentialCandidate) {
            if (!Current.contains(C + "-")) {
                if (DataReader.Distance(C, DelBSS, DataReader.UAVmax / 2, Data) == 1) {
                    NeighBSS += C + "-";
                }
            }
        }
    }
    if (!NeighBSS.equals("")) {
        return S.removeCandidate(DelBSS).InsertCandidate(NeighBSS.split("-")[Data.R.nextInt(NeighBSS.split("-").length)]);
    }
    S.changeMove("NONE");
    return S;
}
public Solution Remove(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    return S.removeCandidate(Data.R.nextInt(S.Candidates.split("-").length));
}
```

```java
public Solution RemoveSmart(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    double Min = Double.MAX_VALUE;
    String NameMin = "";
    for (String Name : S.Utilize.keySet()) {
        if (S.Utilize.get(Name) < Min) {
            Min = S.Utilize.get(Name);
            NameMin = Name;
        }
    }
    return S.removeCandidate(NameMin);
}
public Solution RandomInsert(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    return S.InsertCandidate();
}
public Solution Broading(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    String DelBSS = S.Candidates.split("-")[Data.R.nextInt(S.Candidates.split("-").length)];
    String Current = S.Candidates + "-";
    String NeighBSS = "";
    for (String C : PotentialCandidate) {
        if (!Current.contains(C + "-")) {
            if (DataReader.Distance(C, DelBSS, DataReader.UAVmax, Data) == 1) {
                NeighBSS += C + "-";
            }
        }
    }
    if (!NeighBSS.equals("")) {
        S.removeCandidate(DelBSS);
        if (NeighBSS.split("-").length >= 2) {
            String C1 = NeighBSS.split("-")[Data.R.nextInt(NeighBSS.split("-").length)];
            String C2 = NeighBSS.split("-")[Data.R.nextInt(NeighBSS.split("-").length)];
            while (C1.equals(C2)) {
                C2 = NeighBSS.split("-")[Data.R.nextInt(NeighBSS.split("-").length)];
            }
            return S.InsertCandidate(C1).InsertCandidate(C2);
        } else {
            return S.InsertCandidate(NeighBSS.split("-")[Data.R.nextInt(NeighBSS.split("-").length)]);
        }
    }
    S.changeMove("NONE");
    return S;
}
public Solution Upgrade(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    double Max = -1;
    String MaxName = "";
    if (!S.Utilize.isEmpty()) {
        for (String C : S.Utilize.keySet()) {
            if (Max < S.Utilize.get(C)) {
                Max = S.Utilize.get(C);
                MaxName = C;
            }
        }
        String T = "";
```

```java
        for (int i = 0; i < S.Candidates.split("-").length; i++) {
            if (S.Candidates.split("-")[i].equals(MaxName)) {
                if (Integer.parseInt(S.CandidateType.split("-")[i]) < Data.MaxUpgrade) {
                    T += Integer.toString(Integer.parseInt(S.CandidateType.split("-")[i]) + 1) + "-";
                } else {
                    T += Data.MaxUpgrade + "-";
                }
            } else {
                T += S.CandidateType.split("-")[i] + "-";
            }
        }
        T = T.substring(0, T.length() - 1);
        return new Solution(S.Candidates, T, Data, MaxName + "-Up");
    }
    return S;
}
public Solution PMInsert(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    HashMap<String, Double> DemandNeighborHood = new HashMap<String, Double>();
    String[] DemandArray = S.Demands.split("-");
    String CCandidate = S.Candidates + "-";
    for (String C : PotentialCandidate) {
        double NDemand = 0;
        if (!CCandidate.contains(C + "-")) {
            for (String D : DemandArray) {
                if ((DataReader.Distance(C, D, DataReader.UAVmax, Data) != 0)) {
                    NDemand += 1;
                }
            }
            DemandNeighborHood.put(C, NDemand);
        }
    }
    double Max = -1 * Double.MAX_VALUE;
    String CandidateInsert = "";
    for (String C : DemandNeighborHood.keySet()) {
        if (Max < DemandNeighborHood.get(C)) {
            Max = DemandNeighborHood.get(C);
            CandidateInsert = C;
        }
    }
    return S.InsertCandidate(CandidateInsert);
}
public Solution TPInsert(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    HashMap<String, Double> AllNeighbor = new HashMap<String, Double>();
    String AllPoint = Data.Suppliers + "-" + S.Candidates + "-";
    String[] DemandArray = S.Demands.split("-");
    String[] AllCurrentLocations = AllPoint.substring(0, AllPoint.length() - 1).split("-");
    for (String C : PotentialCandidate) {
        double NDemand = 0;
        double MDemand = 0;
        if (!AllPoint.contains(C + "-")) {
            for (String D : AllCurrentLocations) {
                if ((DataReader.Distance(C, D, DataReader.UAVmax, Data) != 0)) {
                    NDemand += 1;
                }
```

```
            }
            for (String D : DemandArray) {
                if ((DataReader.Distance(C, D, DataReader.UAVmax, Data) != 0)) {
                    MDemand += 1;
                }
            }
            AllNeighbor.put(C, (NDemand * MDemand) / Data.CostConstruction[0]);
        }
    }
    double Max = -1 * Double.MAX_VALUE;
    String CandidateInsert = "";
    for (String C : AllNeighbor.keySet()) {
        if (Max < AllNeighbor.get(C)) {
            Max = AllNeighbor.get(C);
            CandidateInsert = C;
        }
    }
    return S.InsertCandidate(CandidateInsert);
}
public Solution PNInsert(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    HashMap<String, Double> AllNeighbor = new HashMap<String, Double>();
    String AllPoint = Data.Demands + "-" + Data.Suppliers + "-" + S.Candidates + "-";
    String[] AllCurrentLocations = AllPoint.substring(0, AllPoint.length() - 1).split("-");
    for (String C : PotentialCandidate) {
        double NDemand = 0;
        if (!AllPoint.contains(C + "-")) {
            for (String D : AllCurrentLocations) {
                if ((DataReader.Distance(C, D, DataReader.UAVmax, Data) != 0)) {
                    NDemand += 1;
                }
            }
            AllNeighbor.put(C, NDemand);
        }
    }
    double Max = -1 * Double.MAX_VALUE;
    String CandidateInsert = "";
    for (String C : AllNeighbor.keySet()) {
        if (Max < AllNeighbor.get(C)) {
            Max = AllNeighbor.get(C);
            CandidateInsert = C;
        }
    }
    return S.InsertCandidate(CandidateInsert);
}
public Solution AreaInsertation(Solution CS) {
    return null;
}
public Solution Downgrade(Solution CS) throws CloneNotSupportedException, IloException {
    Solution S = CS.clone();
    double Min = 2;
    String MinName = "";
    if (!S.Utilize.isEmpty()) {
        for (String C : S.Utilize.keySet()) {
            if ((Min > S.Utilize.get(C)) && (S.Utilize.get(C) > 0) && (S.getType(C) > 1)) {
                Min = S.Utilize.get(C);
```

```
                MinName = C;
            }
        }
        String T = "";
        for (int i = 0; i < S.Candidates.split("-").length; i++) {
            if (S.Candidates.split("-")[i].equals(MinName)) {
                T += Integer.toString(Integer.parseInt(S.CandidateType.split("-")[i]) - 1) + "-";
            } else {
                T += S.CandidateType.split("-")[i] + "-";
            }
        }
        T = T.substring(0, T.length() - 1);
        return new Solution(S.Candidates, T, Data, MinName + "-Up");
    }
    return S;
}
private void ChangeTabu(String TabuMove) {
    if (TabuMove.split("-")[1].equals("Insert")) {
        if (!TabuRemove.equals("")) {
            TabuRemove += "-" + TabuMove.split("-")[0];
        } else {
            TabuRemove = TabuMove.split("-")[0];
        }
        if (TabuRemove.split("-").length > TabuSize) {
            String[] T = TabuRemove.split("-");
            TabuRemove = "";
            for (int i = T.length - 1; i > T.length - TabuSize - 1; i--) {
                TabuRemove += T[i] + "-";
            }
        }
    } else if (TabuMove.split("-")[1].equals("Remove")) {
        if (!TabuInsert.equals("")) {
            TabuInsert += "-" + TabuMove.split("-")[0];
        } else {
            TabuInsert = TabuMove.split("-")[0];
        }
        if (TabuInsert.split("-").length > TabuSize) {
            String[] T = TabuInsert.split("-");
            TabuInsert = "";
            for (int i = T.length - 1; i > T.length - TabuSize - 1; i--) {
                TabuInsert += T[i] + "-";
            }
        }
    } else {
        if (!TabuUpgrade.equals("")) {
            TabuUpgrade += "-" + TabuMove.split("-")[0];
        } else {
            TabuUpgrade = TabuMove.split("-")[0];
        }
        if (TabuUpgrade.split("-").length > TabuSize) {
            String[] T = TabuUpgrade.split("-");
            TabuUpgrade = "";
            for (int i = T.length - 1; i > T.length - TabuSize - 1; i--) {
                TabuUpgrade += T[i] + "-";
            }
        }
```

```
        }
    }
    private boolean TabuValid(String TabuMove) {
        String T = "";
        if (!TabuMove.equals("")) {
            if (TabuMove.equals("NONE")) {
                return false;
            }
            if (TabuMove.split("-")[1].equals("Insert")) {
                T = TabuInsert;
            } else if (TabuMove.split("-")[1].equals("Remove")) {
                T = TabuRemove;
            } else {
                T = TabuUpgrade;
            }
            for (int i = 0; i < T.split("-").length; i++) {
                if (T.split("-")[i].equals(TabuMove.split("-")[0])) {
                    return false;
                }
            }
        }
        return true;
    }
}
```

A.2 Max Flow Solver Class

```
package phdalgorithm;
/**
 *
 * @author aghahari
 */
import ilog.concert.*;
import ilog.cplex.*;
import java.util.ArrayList;
import java.util.HashMap;
public class CplexSolver {
    IloCplex Model;
    // List of Each Arc associated with a Node!
    ArrayList<ArrayList<String>> AdjList = new ArrayList();
    ArrayList<ArrayList<String>> RevAdjList = new ArrayList();
    // String = Name of Node - Integer = Row of Array Associate with
    HashMap<String, Integer> NodeList = new HashMap<String, Integer>();
    IloNumVar[] X;
    IloNumVar V;
    public CplexSolver(String InputPoints, HashMap<String, Integer> InputArcs) throws IloException {
        Model = new IloCplex();
        Model.setOut(null);
        String[] ArrPoint = InputPoints.split("-");
        for (int i = 0; i < ArrPoint.length; i++) {
            if (!NodeList.containsKey(ArrPoint[i])) {
                NodeList.put(ArrPoint[i], i);
            }
```

```
            ArrayList<String> NewAdj = new ArrayList<String>();
            ArrayList<String> RevNewAdj = new ArrayList<String>();
            for (String ID : InputArcs.keySet()) {
                String IDTemp = ID + "!!"; //Half ID the same
                if (ID.contains(ArrPoint[i] + "-")) {
                    NewAdj.add(ID);
                }
                if (IDTemp.contains("-" + ArrPoint[i] + "!!")) {
                    RevNewAdj.add(ID);
                }
            }
            AdjList.add(NewAdj);
            RevAdjList.add(RevNewAdj);
        }
        V = Model.numVar(0, Double.MAX_VALUE, "Maximum_Flow");
        X = new IloNumVar[InputArcs.size()];
        HashMap<String, Integer> ArcList = new HashMap<String, Integer>();
        int Counter = 0;
        for (String ID : InputArcs.keySet()) {
            if (!ArcList.containsKey(ID)) {
                ArcList.put(ID, Counter);
            }
            X[Counter] = Model.numVar(0.0, InputArcs.get(ID), "X_" + ID);
            Counter++;
        }
        // Objective Value MAXIMIZE FLOW
        IloLinearNumExpr Obj1 = Model.linearNumExpr();
        Obj1.addTerm(V, 1.0);
        Model.addMaximize(Obj1);
        // Flow Balance Constraint
        for (String S : NodeList.keySet()) {
            IloLinearNumExpr FlowExpr = Model.linearNumExpr();
            int NodeNum = NodeList.get(S);
            ArrayList<String> Adj = AdjList.get(NodeList.get(S));
            ArrayList<String> RevAdj = RevAdjList.get(NodeList.get(S));
            for (int i = 0; i < Adj.size(); i++) {
                FlowExpr.addTerm(1.0, X[ArcList.get(Adj.get(i))]);
            }
            for (int i = 0; i < RevAdj.size(); i++) {
                FlowExpr.addTerm(-1.0, X[ArcList.get(RevAdj.get(i))]);
            }
            if (S.equals("MainSupplier")) {
                FlowExpr.addTerm(-1.0, V);
            }
            if (S.equals("MainDemand")) {
                FlowExpr.addTerm(1.0, V);
            }
            Model.addEq(FlowExpr, 0.0, "FlowBalance-" + S);
        }
    }
    public void Solve() throws IloException {
        Model.solve();
    }
    public void end() {
        Model.end();
    }
```

```
    public double getObj() throws IloException {
        return Model.getObjValue();
    }
    public HashMap<String, Integer> getPreflow() throws IloException {
        double[] XValue = Model.getValues(X);
        HashMap<String, Integer> Ans = new HashMap<String, Integer>();
        for (int i = 0; i < XValue.length; i++) {
            if (!Ans.containsKey(X[i].getName().replace("X_", ""))) {
                Ans.put(X[i].getName().replace("X_", ""), (int) XValue[i]);
            }
        }
        return Ans;
    }
}
```

A.3 Dynamic Programming Class

```
package phdalgorithm;
import java.io.IOException;
/**
 *
 * @author aghahari
 */
public class AI {
    ReadData Data;
    private Station[] StationList;
    private DemandPoint[] DemandList;
    private int CoveredDemand = 0;
    private int SelectedSize = 0;
    private int CoveredSize = 0;
    private double CoveredPoint = 0.5;
    private Station[] Network;
    public AI(String Name) throws IOException {
        Data = new ReadData(this, Name);
        // Supplier ID is negative one
        Station Supplier = new Station(this, -1, 0, 0, "Supplier");
        Station.setMySides(StationList);
        Network = new Station[StationList.length];
        this.Selecting(Supplier);
        this.AddtoNetwork(Supplier);
    }
    public String Run() {
        DemandPoint Dem = null;
        Dem = getFarDemand();
        while (Dem != null) {
            ConnectDemand(Dem);
            Dem = getFarDemand();
        }
        String Sr = "";
        String SrG = "";
        for (Station S : Network) {
            if (S != null) {
                Selecting(S);
```

```
            if (S.getID() >= 0) {
                Sr += "Candidate" + Integer.toString(S.getID() + 1) + "-";
                SrG += S.getType() + "-";
            }
        }
    }
    return Sr.substring(0, Sr.length() - 1) + "!" + SrG.substring(0, SrG.length() - 1);
} // Run The AI
int getTimeLine() {
    return Data.getTime();
} // Get Number of Time Slot
void setStationArray(int NumCandidates) {
    // Num Candidates + Supplier
    StationList = new Station[NumCandidates + 1];
} // Set the Size of Station
void addStation(Station aThis) {
    if (aThis.getID() != -1) {
        StationList[aThis.getID()] = aThis;
    } else {
        StationList[StationList.length - 1] = aThis;
    }
} // Add a Station
void setDemandArray(int NumDemand) {
    DemandList = new DemandPoint[NumDemand];
} // Set the Size of Demand Point
void addDemand(DemandPoint aThis) {
    DemandList[aThis.getID()] = aThis;
} // Add Demand Point
int FinalDecision = -1;
public void ConnectDemand(DemandPoint Destination) {
    Station Source = getNearestStation(Destination);
    int MaxMove = getMaxMove(Source, Destination);
    String History = "";
    Station St = Source;
    Station[] Route = new Station[MaxMove + 1];
    for (int i = 0; i <= MaxMove; i++) {
        History = "-" + St.getName() + "-";
        AddtoNetwork(St);
        ConnectPair(St, Destination, History, MaxMove - i);
        switch (FinalDecision) {
            case 2:
                St = St.getUp();
                break;
            case 3:
                St = St.getDown();
                break;
            case 1:
                St = St.getLeft();
                break;
            case 0:
                St = St.getRight();
                break;
        }
    }
} // Connect a Demand to Network
public void stationstatus() {
```

```java
    for (Station S : StationList) {
        if (S.isSelected()) {
            System.out.println(S.getName());
        }
    }
}
public DemandPoint getFarDemand() { //
    Station[] selectedStation = getSelectedStation();
    DemandPoint[] uncoveredDemand = getUnCoveredDemand();
    double Distance = -1 * Double.MAX_VALUE;
    DemandPoint FarDemand = null;
    if (uncoveredDemand != null) {
        for (DemandPoint Dem : uncoveredDemand) {
            double DistanceMin = Double.MAX_VALUE;
            for (Station St : selectedStation) {
                if (Distance(St, Dem) < DistanceMin) {
                    DistanceMin = Distance(St, Dem);
                }
            }
            if (DistanceMin > Distance) {
                FarDemand = Dem;
                Distance = DistanceMin;
            }
        }
    }
    return FarDemand;
} // Get Farthest Demand to a Station
public Station[] getSelectedStation() {
    if (getSelectedSize() != 0) {
        Station[] SelectedStation = new Station[getSelectedSize()];
        int Counter = 0;
        for (Station A : StationList) {
            if (A.isSelected()) {
                SelectedStation[Counter++] = A;
            }
        }
        return SelectedStation;
    }
    return null;
} // Return All Selected Station
private Station getNearestStation(DemandPoint Demand) {
    Station[] SelectedList = getSelectedStation();
    double Distance = Double.MAX_VALUE;
    Station NearestOne = null;
    for (Station St : SelectedList) {
        if (Distance(St, Demand) < Distance) {
            NearestOne = St;
            Distance = Distance(St, Demand);
        }
    }
    return NearestOne;
} // Get Nearest Station to a Demand Point
private int getSelectedSize() {
    return SelectedSize;
} // Get Selected Stations Size
private int getCoveredSize() {
```

```
    return CoveredSize;
} // Return Demand Covered Size
private void Selecting(Station Station) {
    if (!Station.isSelected()) {
        SelectedSize += 1;
        DemandPoint[] unCoveredDemand = getUnCoveredDemand();
        if (unCoveredDemand != null) {
            for (DemandPoint Dem : unCoveredDemand) {
                if (Distance(Station, Dem) <= Data.getFlyingRange()) {
                    Covering(Dem, Station);
                    Station.incrementCoverage();
                }
            }
        }
        Station.selecting(3);
    }
} // Selecting a Station
private void Deselecting(Station Station) {
    if (!IsNetwork(Station)) {
        if (Station.isSelected()) {
            SelectedSize -= 1;
            Station.deselect();
            DemandPoint[] CoverdDemand = getCoveredDemand(Station);
            if (CoverdDemand != null) {
                for (DemandPoint Dem : CoverdDemand) {
                    Uncovering(Dem);
                    Station.DecrementCoverage();
                }
            }
        }
    }
}
private void Covering(DemandPoint Demand, Station station) {
    CoveredSize += 1;
    CoveredDemand += Demand.getDemand();
    Demand.Covering(station);
} // Adjust Parameters - Cover Demand Object
private String Cord(Station S) {
    return S.getXString() + "," + S.getYString();
} // String Coordinate X-Y Station Object
private String Cord(DemandPoint D) {
    return D.getXString() + "," + D.getYString();
} // String Coordinate X-Y Demand Object
private double Distance(String Cord, String Cord0) {
    return Data.getDistance(Cord, Cord0);
} // Distance Based on String Coordinate
private DemandPoint[] getUnCoveredDemand() {
    DemandPoint[] Uncovered = new DemandPoint[DemandList.length - getCoveredSize()];
    int Counter = 0;
    for (DemandPoint Demand : DemandList) {
        if (!Demand.isCover()) {
            Uncovered[Counter++] = Demand;
        }
    }
    if (Counter == 0) {
        return null;
```

```
    }
    return Uncovered;
} // Get All Uncovered Demand
private double ConnectPair(Station Source, DemandPoint Destination, String MoveHistory, int MaxMove) {
    Selecting(Source);
    // STOP SIGN
    if (MaxMove == 0) {
        if (Destination.isCover()) {
            Deselecting(Source);
            return 0;
        } else {
            Deselecting(Source);
            return -10000;
        }
    } else {
        double LeftMove;
        if (Source.isLeft() && (!MoveHistory.contains("-" + Source.getLeft().getName() + "-"))) {
            LeftMove = ConnectPair(Source.getLeft(), Destination, MoveHistory + "-" + Source.getLeft().getName() + "-", MaxMove - 1) +
                    ScoreSelecting(Source.getLeft());
        } else {
            LeftMove = -100;
        }
        double RightMove;
        if (Source.isRight() && (!MoveHistory.contains("-" + Source.getRight().getName() + "-"))) {
            RightMove = ConnectPair(Source.getRight(), Destination, MoveHistory + "-" + Source.getRight().getName() + "-", MaxMove - 1) +
                    ScoreSelecting(Source.getRight());
        } else {
            RightMove = -100;
        }
        double UpMove;
        if (Source.isUp() && (!MoveHistory.contains("-" + Source.getUp().getName() + "-"))) {
            UpMove = ConnectPair(Source.getUp(), Destination, MoveHistory + "-" + Source.getUp().getName() + "-", MaxMove - 1) +
                    ScoreSelecting(Source.getUp());
        } else {
            UpMove = -100;
        }
        double DownMove;
        if (Source.isDown() && (!MoveHistory.contains("-" + Source.getDown().getName() + "-"))) {
            DownMove = ConnectPair(Source.getDown(), Destination, MoveHistory + "-" + Source.getDown().getName() + "-", MaxMove - 1) +
                    ScoreSelecting(Source.getDown());
        } else {
            DownMove = -100;
        }
        double Max;
        //System.out.println(MoveHistory);
        if (LeftMove < RightMove) {
            Max = RightMove;
            FinalDecision = 0;//"Right";
        } else {
            Max = LeftMove;
            FinalDecision = 1;//"Left";
        }
        if (Max < UpMove) {
            Max = UpMove;
            FinalDecision = 2;//"Up";
        }
```

```java
        if (Max < DownMove) {
            Max = DownMove;
            FinalDecision = 3;//"Down";
        }
        Deselecting(Source);
        return Max;
    }
} // Connecting Two Pair of Station and Demand Based on best Route!!
private double Distance(Station Station, DemandPoint Dem) {
    return Distance(Cord(Station), Cord(Dem));
} // Get Distance based on Objects
private int getMaxMove(Station Source, DemandPoint Destination) {
    return (int) ((Math.abs(Source.getY() - Destination.getY()) + Math.abs(Source.getX() - Destination.getX())) / Data.getFlyingRange());
}
private double ScoreSelecting(Station sStation) {
    double Point = 0;
    for (DemandPoint Dem : DemandList) {
        if (reachable(sStation, Dem)) {
            if (!Dem.isCover()) {
                Point += Dem.getDemand();
            } else {
                Point += CoveredPoint * Dem.getDemand();
            }
        } else if (!Dem.isCover()) {
            Point += Dem.getDemand() / Distance(sStation, Dem);
        } else {
            Point += CoveredPoint * Dem.getDemand() / Distance(sStation, Dem);
        }
    }
    return Point;
}
private boolean reachable(Station sStation, DemandPoint Dem) {
    if (Distance(sStation, Dem) < Data.getFlyingRange()) {
        return true;
    } else {
        return false;
    }
}
DemandPoint[] getDemandList() {
    return DemandList;
}
private DemandPoint[] getCoveredDemand(Station Station) {
    DemandPoint[] Demands = new DemandPoint[Station.getCoverage()];
    int Counter = 0;
    for (DemandPoint Dem : DemandList) {
        if (Dem.isCover()) {
            if (Dem.getStation().getID() == Station.getID()) {
                Demands[Counter++] = Dem;
            }
        }
    }
    if (Counter == 0) {
        return null;
    }
    return Demands;
}
```

```
    private DemandPoint[] getCoveredDemand() {
        DemandPoint[] CoveredDemand = new DemandPoint[CoveredSize];
        int Counter = 0;
        for (DemandPoint Dem : DemandList) {
            if (Dem.isCover()) {
                CoveredDemand[Counter++] = Dem;
            }
        }
        return CoveredDemand;
    }
    private void Uncovering(DemandPoint Dem) {
        Station[] SelectedStation = getSelectedStation();
        boolean reset = false;
        for (Station S : SelectedStation) {
            if (reachable(S, Dem)) {
                Dem.ResetStation(S);
                S.incrementCoverage();
                reset = true;
                break;
            }
        }
        if (!reset) {
            CoveredDemand -= Dem.getDemand();
            CoveredSize -= 1;
            Dem.Uncover();
        }
    }
    private void AddtoNetwork(Station St) {
        if (!IsNetwork(St)) {
            for (int i = 0; i < Network.length; i++) {
                if (Network[i] == null) {
                    Network[i] = St;
                    break;
                }
            }
        }
    }
    private boolean IsNetwork(Station St) {
        for (int i = 0; i < Network.length; i++) {
            if (Network[i] != null) {
                if (Network[i].getID() == St.getID()) {
                    return true;
                }
            }
        }
        return false;
    }
}
```

A.4 Data Reader Class

```
package phdalgorithm;
import java.io.BufferedReader;
```

```java
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Random;
/**
 *
 * @author aghahari
 */
public class DataReader {
    HashMap<String, Point> Points = new HashMap<String, Point>();
    static double UAVmax;
    HashMap<String, Integer> Distance = new HashMap<String, Integer>();
    double[] CostConstruction;
    double CostLostSale;
    double[] CostOperational;
    HashMap<Integer, Integer> Battery;
    String Suppliers;
    String Demands;
    int Horizon;
    HashMap<String, Integer> DemandScenario;
    int MaxUpgrade = 3;
    static Random R = new Random();
    String DistanceType;
    HashMap<String, Double> ObjHistory = new HashMap<String, Double>();
    int NCandidate;
    int NSupplier;
    int NDemand;
    private PointType getType(String Name) {
        if (Name.contains("Supplier")) {
            return PointType.Supplier;
        } else if (Name.contains("Candidate")) {
            return PointType.Candidate;
        }
        return PointType.Demand;
    }
    static public enum PointType {

        Candidate, Supplier, Demand
    }
    public void InsertObj(String ID, double Value) {
        if (!ObjHistory.containsKey(ID)) {
            ObjHistory.put(ID, Value);
        }
    }
    public double getObjective(String ID) {
        if (ObjHistory.get(ID)==null){
            return -1.0;
        }else{
            return ObjHistory.get(ID);
        }
    }
```

```java
public DataReader(String Name, String Distance) throws IOException {
    ReadBender(Name, Distance);
}
public DataReader(double UAV, String InputBattery, String InputSupplier, String InputDemandPoint, HashMap<String, Integer>
        InputDemandScenario, int InputHorizon, String InCostConstruction, String InCostOperation, double CLostSale, String DistanceT) throws
        FileNotFoundException, IOException {
    DistanceType = DistanceT;
    CostLostSale = CLostSale;
    CostConstruction = new double[InCostConstruction.split("-").length];
    for (int i = 0; i < InCostConstruction.split("-").length; i++) {
        CostConstruction[i] = Double.parseDouble(InCostConstruction.split("-")[i]);
    }
    CostOperational = new double[InCostOperation.split("-").length];
    for (int i = 0; i < InCostOperation.split("-").length; i++) {
        CostOperational[i] = Double.parseDouble(InCostOperation.split("-")[i]);
    }
    BufferedReader Input = new BufferedReader(new FileReader("Input.txt"));
    String newLine = "";
    while (!newLine.equals("Data")) {
        newLine = Input.readLine();
    }
    String[] NBattery = InputBattery.split("-");
    Battery = new HashMap<Integer, Integer>();
    MaxUpgrade = InputBattery.split("-").length;
    for (int i = 0; i < NBattery.length; i++) {
        Battery.put(i + 1, Integer.parseInt(NBattery[i]));
    }
    while ((newLine = Input.readLine()) != null) {
        Points.put(newLine.split("!")[0], new Point(Double.parseDouble(newLine.split("!")[1]), Double.parseDouble(newLine.split("!")[2]),
                newLine.split("!")[0]));
    }
    Suppliers = InputSupplier;
    Demands = InputDemandPoint;
    DemandScenario = InputDemandScenario;
    UAVmax = UAV;
    Horizon = InputHorizon;
    for (Point P1 : Points.values()) {
        for (Point P2 : Points.values()) {
            if (!Distance.containsKey(P1.Name + "-" + P2.Name)) {
                Distance.put(P1.Name + "-" + P2.Name, DistanceCalculator(P1, P2, this));
            }
        }
    }
    System.out.println("Data Read!");
    WriteBender("NInput.txt");
}
public HashMap<String, Integer> getDemandScenario() {
    return DemandScenario;
}
public int getDistance(String T1, String T2) {
    if (Distance.get(T1 + "-" + T2)==null){
        return 0;
    }else{
        return Distance.get(T1 + "-" + T2);
    }
}
```

```java
public int getBattery(int Type) {
    return Battery.get(Type);
}
public String[] getListType(PointType Type) {
    List<String> A = new ArrayList<String>();
    for (Point P : Points.values()) {
        if (this.getType(P.Name) == Type) {
            A.add(P.Name);
        }
    }
    String[] Ans = new String[A.size()];
    for (int i = 0; i < Ans.length; i++) {
        Ans[i] = A.get(i);
    }
    return Ans;
}
public static int DistanceCalculator(Point P1, Point P2, DataReader Data) {
    if (Data.DistanceType.equals("Linear")) {
        return DistanceLinear(P1, P2);
    } else {
        return Distance(P1, P2);
    }
}
public static int Distance(Point P1, Point P2, double UAVMax) {
    double R = 3963.1676;
    double lat1 = P1.lat * (Math.PI / 180.0);
    double lng1 = P1.lng * (Math.PI / 180.0);
    double lat2 = P2.lat * (Math.PI / 180.0);
    double lng2 = P2.lng * (Math.PI / 180.0);
    double deltaPhi = lat2 - lat1;
    double deltaLambda = lng2 - lng1;
    double a = Math.sin(deltaPhi / 2.0) * Math.sin(deltaPhi / 2.0) + Math.cos(lat1) * Math.cos(lat2) * Math.sin(deltaLambda / 2.0) *
          Math.sin(deltaLambda / 2.0);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double d = R * c;
    if ((d > UAVMax) || (d == 0)) {
        return 0;
    } else {
        return 1;
    }
}
public static int Distance(String PS1, String PS2, double UAVMax, DataReader Data) {
    double R = 3963.1676;
    Point P1 = null;
    Point P2 = null;
    for (Point P : Data.Points.values()) {
        if (P.Name.equals(PS1)) {
            P1 = P;
        }
        if (P.Name.equals(PS2)) {
            P2 = P;
        }
    }
    double lat1 = P1.lat * (Math.PI / 180.0);
    double lng1 = P1.lng * (Math.PI / 180.0);
    double lat2 = P2.lat * (Math.PI / 180.0);
```

```
    double lng2 = P2.lng * (Math.PI / 180.0);
    double deltaPhi = lat2 - lat1;
    double deltaLambda = lng2 - lng1;
    double a = Math.sin(deltaPhi / 2.0) * Math.sin(deltaPhi / 2.0) + Math.cos(lat1) * Math.cos(lat2) * Math.sin(deltaLambda / 2.0) *
          Math.sin(deltaLambda / 2.0);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double d = R * c;
    if ((d > UAVMax) || (d == 0)) {
       return 0;
    } else {
       return 1;
    }
}
public static int Distance(Point P1, Point P2) {
    double R = 3963.1676;
    double lat1 = P1.lat * (Math.PI / 180.0);
    double lng1 = P1.lng * (Math.PI / 180.0);
    double lat2 = P2.lat * (Math.PI / 180.0);
    double lng2 = P2.lng * (Math.PI / 180.0);
    double deltaPhi = lat2 - lat1;
    double deltaLambda = lng2 - lng1;
    double a = Math.sin(deltaPhi / 2.0) * Math.sin(deltaPhi / 2.0) + Math.cos(lat1) * Math.cos(lat2) * Math.sin(deltaLambda / 2.0) *
          Math.sin(deltaLambda / 2.0);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    double d = R * c;
    if ((d > UAVmax) || (d == 0)) {
       return 0;
    } else {
       return 1;
    }
}
public String getSupplier() {
    return Suppliers;
}
public String getDemandPoint() {
    return Demands;
}
int getHorizon() {
    return Horizon;
}
public static int DistanceLinear(Point P1, Point P2) {
    double d = Math.sqrt(Math.pow(P1.lat - P2.lat, 2) + Math.pow(P1.lng - P2.lng, 2));
    if ((d > UAVmax) || (d == 0)) {
       return 0;
    } else {
       return 1;
    }
}
public void WriteBender(String Name) throws IOException {
    BufferedWriter OutputData = new BufferedWriter(new FileWriter(Name));
    int NumDemand = 0;
    String SDemand = "";
    int NumCandidate = 0;
    String SCandidate = "";
    int NumSupplier = 0;
    String SSupplier = "";
```

```
    for (String S : Points.keySet()) {
        Point P = Points.get(S);
        if (this.getType(P.Name) == PointType.Candidate) {
            NumCandidate += 1;
        } else if (this.getType(P.Name) == PointType.Supplier) {
            NumSupplier += 1;
        } else {
            NumDemand += 1;
        }
    }
    OutputData.write(UAVmax + "\n");
    OutputData.write(NumCandidate + "\n");
    OutputData.write(CostConstruction.length + "\n");
    OutputData.write(Horizon + "\n");
    for (int i = 0; i < CostConstruction.length; i++) {
        OutputData.write(Double.toString(CostConstruction[i]) + "\n");
    }
    for (int i = 0; i < CostOperational.length; i++) {
        OutputData.write(Double.toString(CostOperational[i]) + "\n");
    }
    for (int i = 1; i <= CostConstruction.length; i++) {
        OutputData.write(Battery.get(i) + "\n");
    }

    for (String S : Points.keySet()) {
        Point P = Points.get(S);
        if (this.getType(P.Name) == PointType.Candidate) {
            OutputData.write(P.lat + "," + P.lng + "\n");
        }
    }
    OutputData.write(NumDemand + "\n");
    for (String S : Points.keySet()) {
        Point P = Points.get(S);
        if (this.getType(P.Name) == PointType.Demand) {
            OutputData.write(P.lat + "," + P.lng + "," + DemandScenario.get(P.Name) + "\n");
        }
    }
    OutputData.write(NumSupplier + "\n");
    for (String S : Points.keySet()) {
        Point P = Points.get(S);
        if (this.getType(P.Name) == PointType.Supplier) {
            OutputData.write(P.lat + "," + P.lng + "\n");
        }
    }
    OutputData.write(Double.toString(CostLostSale));
    OutputData.close();
}
public void WriteCplex(String Name) throws IOException {
    BufferedWriter OutputData = new BufferedWriter(new FileWriter(Name));
    OutputData.write("param b :=" + "\n");
    String Allpoint = "";
    for (int C : Battery.keySet()) {
        OutputData.write(C + "\t" + Battery.get(C) + "\n");
    }
    OutputData.write(";" + "\n" + "param S:=" + "\n");
    for (int i = 0; i < this.CostConstruction.length; i++) {
```

```
    OutputData.write(Integer.toString(i + 1) + "\t" + CostConstruction[i] + "\n");
}
OutputData.write(";" + "\n" + "param Loss = " + this.CostLostSale + ";" + "\n");

OutputData.write("param Operational :=" + "\n");
for (int i = 0; i < this.CostOperational.length; i++) {
    OutputData.write(Integer.toString(i + 1) + "\t" + CostOperational[i] + "\n");
}
OutputData.write(";" + "\n" + "set Time := ");
for (int i = 1; i <= Horizon; i++) {
    OutputData.write(Integer.toString(i) + "\t");
}
OutputData.write(";" + "\n" + "set Type:=" + "\n");
for (int i = 1; i <= MaxUpgrade; i++) {
    OutputData.write(Integer.toString(i) + "\t");
}
OutputData.write(";" + "\n" + "set Scenario := 1;" + "\n" + "set Supplier :=" + "\n");
for (int i = 0; i < this.Suppliers.split("-").length; i++) {
    OutputData.write(this.Suppliers.split("-")[i] + "\n");
    Allpoint += this.Suppliers.split("-")[i] + "-";
}
OutputData.write(";" + "\n" + "set Demands :=" + "\n");
for (int i = 0; i < Demands.split("-").length; i++) {
    OutputData.write(Demands.split("-")[i] + "\n");
    Allpoint += Demands.split("-")[i] + "-";
}
OutputData.write(";" + "\n" + "set Candidate :=" + "\n");
for (String P1 : Points.keySet()) {
    if (P1.contains("Candidate")) {
        OutputData.write(P1 + "\n");
        Allpoint += P1 + "-";
    }
}
OutputData.write(";" + "\n" + "param Demand := " + "\n");
for (String C : DemandScenario.keySet()) {
    OutputData.write(C + "\t1\t" + DemandScenario.get(C) + "\n");
}
Allpoint = Allpoint.substring(0, Allpoint.length() - 1);
OutputData.write(";" + "\n" + "param a:\t");
for (int i = 0; i < Allpoint.split("-").length; i++) {
    OutputData.write(Allpoint.split("-")[i] + "\t");
}
OutputData.write(":=" + "\n");
int Siz1 = Allpoint.split("-").length;
for (int i = 0; i < Siz1; i++) {
    Point P1 = Points.get(Allpoint.split("-")[i]);
    String distance = P1.Name + "\t";
    for (int j = 0; j < Siz1; j++) {
        Point P2 = Points.get(Allpoint.split("-")[j]);
        distance += Integer.toString((int) DistanceCalculator(P1, P2, this)) + "\t";
    }
    OutputData.write(distance + "\n");
}
OutputData.write(";" + "\n");
OutputData.close();
System.out.println("CPLEX Done");
```

```java
}

public static class Point {

    double lat;
    double lng;
    String Name;

    public Point(double lat1, double lng1, String Name1) {
        lat = lat1;
        lng = lng1;
        Name = Name1;
    }
}

public void ReadBender(String Name, String DistanceT) throws IOException {
    BufferedReader Input = new BufferedReader(new FileReader(Name));
    DistanceType = DistanceT;
    UAVmax = Integer.parseInt(Input.readLine());
    NCandidate = Integer.parseInt(Input.readLine());
    CostConstruction = new double[Integer.parseInt(Input.readLine())];
    Horizon = Integer.parseInt(Input.readLine());
    for (int i = 0; i < CostConstruction.length; i++) {
        CostConstruction[i] = Double.parseDouble(Input.readLine());
    }
    CostOperational = new double[CostConstruction.length];
    for (int i = 0; i < CostOperational.length; i++) {
        CostOperational[i] = Double.parseDouble(Input.readLine());
    }
    Battery = new HashMap<Integer, Integer>();
    for (int i = 1; i <= CostConstruction.length; i++) {
        Battery.put(i, Integer.parseInt(Input.readLine()));
    }
    MaxUpgrade = Battery.size();
    for (int i = 1; i <= NCandidate; i++) {
        String A = Input.readLine();
        Point P = new Point(Double.parseDouble(A.split(",")[0]), Double.parseDouble(A.split(",")[1]), "Candidate" + i);
        Points.put("Candidate" + i, P);
    }
    NDemand = Integer.parseInt(Input.readLine());
    DemandScenario = new HashMap<String, Integer>();
    Demands = "";
    for (int i = 1; i <= NDemand; i++) {
        String A = Input.readLine();
        Demands += "Demand" + i + "-";
        DemandScenario.put("Demand" + i, Integer.parseInt(A.split(",")[2]));
        Point P = new Point(Double.parseDouble(A.split(",")[0]), Double.parseDouble(A.split(",")[1]), "Demand" + i);
        Points.put("Demand" + i, P);
    }
    Demands = Demands.substring(0, Demands.length() - 1);
    NSupplier = Integer.parseInt(Input.readLine());
    Suppliers = "";
    for (int i = 1; i <= NSupplier; i++) {
        String A = Input.readLine();
        Suppliers += "Supplier" + i + "-";
        Point P = new Point(Double.parseDouble(A.split(",")[0]), Double.parseDouble(A.split(",")[1]), "Supplier" + i);
```

```
            Points.put("Supplier" + i, P);
        }
        Suppliers = Suppliers.substring(0, Suppliers.length() - 1);
        CostLostSale = Double.parseDouble(Input.readLine());
        for (Point P1 : Points.values()) {
            for (Point P2 : Points.values()) {
                if (!Distance.containsKey(P1.Name + "-" + P2.Name)) {
                    Distance.put(P1.Name + "-" + P2.Name, DistanceCalculator(P1, P2, this));
                }
            }
        }
    }
    public String getCor(String Name) {
        return Points.get(Name).lat + "," + Points.get(Name).lng;
    }
}
```

A.5 Demand Point Class

```
package phdalgorithm;
/**
 *
 * @author aghahari
 */
public class DemandPoint {
    private int ID;
    private String Name;
    private double X;
    private double Y;
    private double Amount;
    private double CoveredAmount;
    private boolean Covered;
    private Station CoverStation;
    public DemandPoint(AI parent, int ID1, double Amount1, double X1, double Y1) {
        this(parent, ID1, "Demand_" + Integer.toString(ID1), Amount1, X1, Y1);
    }
    public DemandPoint(AI parent, int ID1, String Name1, double Amount1, double X1, double Y1) {
        ID = ID1;
        Name = Name1;
        Amount = Amount1;
        CoveredAmount = 0;
        X = X1;
        Y = Y1;
        Covered = false;
        CoverStation = null;
        parent.addDemand(this);
    }
    public double getDemand() {
        return Amount;
    }
    public String getName() {
        return Name;
    }
```

```
    public int getID() {
        return ID;
    }
    String getXString() {
        return Double.toString(X);
    }
    String getYString() {
        return Double.toString(Y);
    }
    double getX() {
        return X;
    }
    double getY() {
        return Y;
   }
    public void Covering(Station station) {
        this.Covered = true;
        this.CoverStation = station;
    }
    boolean isCover() {
        return this.Covered;
    }
    Station getStation() {
        return CoverStation;
    }
    void ResetStation(Station S) {
        this.CoverStation = S;
    }
    void Uncover() {
        this.Covered = false;
        this.CoverStation = null;
    }
}
```

A.6 BSS Station Class

```
package phdalgorithm;
/**
 *
 * @author aghahari
 */
public class Station {
    AI Parent;
    private final int ID;
    private final double X;
    private final double Y;
    private final String Name;
    private Station Up;
    private Station Down;
    private Station Left;
    private Station Right;
    private int Battery;
    private int UsedBattery;
```

```java
private double BuildCost;
private double OperationCost;
private int[] Operational;
private boolean Selected;
private int CoveringDemand;
private int Type;
static int distance = 5;
public Station(AI parent, int ID1, double X1, double Y1, String Name1) {
    Parent = parent;
    ID = ID1;
    X = X1;
    Y = Y1;
    Name = Name1;
    Type = 0;
    Battery = 0;
    UsedBattery = 0;
    BuildCost = 0;
    OperationCost = 0;
    CoveringDemand = 0;
    Selected = false;
    Parent.addStation(this);
}
public Station(AI parent, int ID1, double X1, double Y1) {
    this(parent, ID1, X1, Y1, "Station_" + Integer.toString(ID1));
}
static public void setMySides(Station[] StationList) {
    for (Station thisOne : StationList) {
        thisOne.Up = null;
        thisOne.Right = null;
        thisOne.Down = null;
        thisOne.Left = null;
        for (Station Other : StationList) {
            if ((Other.X - thisOne.X == 0) && (Other.Y - thisOne.Y == distance)) {
                thisOne.Up = Other;
            }
            if ((Other.X - thisOne.X == distance) && (Other.Y - thisOne.Y == 0)) {
                thisOne.Right = Other;
            }
            if ((Other.X - thisOne.X == 0) && (Other.Y - thisOne.Y == -1 * distance)) {
                thisOne.Down = Other;
            }
            if ((Other.X - thisOne.X == -1 * distance) && (Other.Y - thisOne.Y == 0)) {
                thisOne.Left = Other;
            }
        }
    }
}
public Station getLeft() {
    return Left;
}
public Station getRight() {
    return Right;
}
public Station getUp() {
    return Up;
}
```

```
public Station getDown() {
    return Down;
}
public int getID() {
    return ID;
}
public String getName() {
    return Name;
}
void selecting(int Type1) {
    Type = Type1 + 1;
    Battery = Parent.Data.getBattery(Type1);
    BuildCost = Parent.Data.getConstruction(Type1);
    OperationCost = Parent.Data.getOperation(Type1);
    UsedBattery = 0;
    Selected = true;
}
boolean isSelected() {
    return Selected;
}
String getXString() {
    return Double.toString(X);
}
String getYString() {
    return Double.toString(Y);
}
double getX() {
    return X;
}
double getY() {
    return Y;
}
boolean isLeft() {
    return Left != null;
}
boolean isUp() {
    return Up != null;
}
boolean isDown() {
    return Down != null;
}
boolean isRight() {
    return Right != null;
}
void incrementCoverage() {
    CoveringDemand++;
}
void DecrementCoverage() {
    CoveringDemand--;
}
int getCoverage() {
    return CoveringDemand;
}
void deselect() {
    Selected = false;
    Type = 0;
```

```
        Battery = 0;
        BuildCost = 0;
        OperationCost = 0;
        UsedBattery = 0;
    }
    String getType() {
        return Integer.toString(this.Type);
    }
}
```

A.7 Solution Class

```
package phdalgorithm;
import ilog.concert.IloException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
/**
 *
 * @author aghahari
 */
public class Solution implements Cloneable {
    HashMap<String, Integer> DemandScenario;
    String Points;
    HashMap<String, Integer> Arcs = new HashMap<String, Integer>();
    DataReader Data;
    String CandidateType;
    String Suppliers;
    String Demands;
    String Candidates;
    int Horizon;
    HashMap<String, Double> Utilize = new HashMap<String, Double>();
    double ObjectiveValue = 0;
    String TabuMove;
    public Solution(String InputCandidate, String InputCandidateType, DataReader InputData, String Tabu) throws CloneNotSupportedException,
           IloException {
        TabuMove = Tabu;
        int MaxDemand = 0;
        Data = InputData;
        for (int C : Data.getDemandScenario().values()) {
            MaxDemand += 2 * C;
        }
        Horizon = Data.getHorizon();
        Suppliers = Data.getSupplier();
        CandidateType = InputCandidateType;
        Demands = Data.getDemandPoint();
        DemandScenario = Data.getDemandScenario();
        Candidates = InputCandidate;
        Points = "";
        // Create different Type of BSS
        List<String> AllPoints = new ArrayList<String>();
        String[] ArrDemands = Divider(Demands);
        String[] ArrSupplier = Divider(Suppliers);
```

```java
String[] ArrCandidate = Divider(Candidates);
//All Demand Point
for (String D : ArrDemands) {
    AllPoints.add(D);
    Points += D + "-";
}
// All Supplier
for (String S : ArrSupplier) {
    AllPoints.add(S);
    Points += S + "-";
}
// Arcs b/w Supplier and Demand ONE WAY!
for (String S : ArrSupplier) {
    for (String D : ArrDemands) {
        int Dis = Data.getDistance(S, D);
        if (Dis == 1) {
            if (!Arcs.containsKey(S + "-" + D)) {
                Arcs.put(S + "-" + D, MaxDemand);
            }
        }
    }
}
// Update Candidate! Arcs b/w Candidate and Supplier and Demand!
for (int i = 0; i < ArrCandidate.length; i++) {
    String C = ArrCandidate[i];
    String HeadNode = C + "Start";
    String TailNode = C + "End";
    if (!Arcs.containsKey(HeadNode + "-" + TailNode)) {
        Arcs.put(HeadNode + "-" + TailNode, Data.getBattery(Integer.parseInt(CandidateType.split("-")[i])));
    }
}
for (String P : AllPoints) {
    for (int i = 0; i < ArrCandidate.length; i++) {
        String C = ArrCandidate[i];
        Utilize.put(C, 0.0);
        int Dis = Data.getDistance(P, C);
        String HeadNode = C + "Start";
        String TailNode = C + "End";
        if (Dis == 1) {
            if (!P.contains("Supplier")) {
                if (!Arcs.containsKey(TailNode + "-" + P)) {
                    Arcs.put(TailNode + "-" + P, MaxDemand);
                }
            }
            if ((P.contains("Supplier")) || (P.contains("Candidate"))) {
                if (!Arcs.containsKey(P + "-" + HeadNode)) {
                    Arcs.put(P + "-" + HeadNode, MaxDemand);
                }
            }
        }

    }
}
for (int i = 0; i < ArrCandidate.length; i++) {
    for (int j = 0; j < ArrCandidate.length; j++) {
        if (i != j) {
```

```
                String C = ArrCandidate[i];
                String D = ArrCandidate[j];
                int Dis = Data.getDistance(D, C);
                if (Dis == 1) {
                    if (!Arcs.containsKey(C + "End-" + D + "Start")) {
                        Arcs.put(C + "End-" + D + "Start", MaxDemand);
                    }
                }
            }
        }
    }
    for (String C : ArrCandidate) {
        Points += C + "Start-";
        Points += C + "End-";
    }
    Points += "MainSupplier-";
    for (String T : ArrSupplier) {
        if (!Arcs.containsKey("MainSupplier-" + T)) {
            Arcs.put("MainSupplier-" + T, MaxDemand);
        }
    }
    Points += "MainDemand";
    for (String T : ArrDemands) {
        if (!Arcs.containsKey(T + "-MainDemand")) {
            Arcs.put(T + "-MainDemand", DemandScenario.get(T));
        }
    }
    ObjectiveValue = this.getObjective();
}
public Solution(String InputCandidate, String InputCandidateType, DataReader InputData, String Tabu, double Obj, HashMap<String, Double>
       Utilize1) {
    TabuMove = Tabu;
    int MaxDemand = 0;
    Data = InputData;
    for (int C : Data.getDemandScenario().values()) {
        MaxDemand += C;
    }
    Horizon = Data.getHorizon();
    Suppliers = Data.getSupplier();
    CandidateType = InputCandidateType;
    Demands = Data.getDemandPoint();
    DemandScenario = Data.getDemandScenario();
    Candidates = InputCandidate;

    Points = "";
    // Create different Type of BSS
    List<String> AllPoints = new ArrayList<String>();
    String[] ArrDemands = Divider(Demands);
    String[] ArrSupplier = Divider(Suppliers);
    String[] ArrCandidate = Divider(Candidates);
    //All Demand Point
    for (String D : ArrDemands) {
        AllPoints.add(D);
        Points += D + "-";
    }
    // All Supplier
```

```
for (String S : ArrSupplier) {
    AllPoints.add(S);
    Points += S + "-";
}
// Arcs b/w Supplier and Demand ONE WAY!
for (String S : ArrSupplier) {
    for (String D : ArrDemands) {
        int Dis = Data.getDistance(S, D);
        if (Dis == 1) {
            if (!Arcs.containsKey(S + "-" + D)) {
                Arcs.put(S + "-" + D, MaxDemand);
            }
            //Arcs.putIfAbsent(D + "-" + S, MaxDemand); Arc az Demand be Supplier ?! Not Possible
        }
    }
}
// Update Candidate! Arcs b/w Candidate and Supplier and Demand!
for (int i = 0; i < ArrCandidate.length; i++) {
    String C = ArrCandidate[i];
    String HeadNode = C + "Start";
    String TailNode = C + "End";
    if (!Arcs.containsKey(HeadNode + "-" + TailNode)) {
        Arcs.put(HeadNode + "-" + TailNode, Data.getBattery(Integer.parseInt(CandidateType.split("-")[i])));
    }
}
for (String P : AllPoints) {
    for (int i = 0; i < ArrCandidate.length; i++) {
        String C = ArrCandidate[i];
        if (Utilize1 == null) {
            Utilize.put(C, 0.0);
        } else {
            if (Utilize1.get(C) == null) {
                Utilize.put(C, 0.0);
            } else {
                Utilize.put(C, Utilize1.get(C));
            };
        }
        int Dis = Data.getDistance(P, C);
        String HeadNode = C + "Start";
        String TailNode = C + "End";
        if (Dis == 1) {
            if (!P.contains("Supplier")) {
                if (!Arcs.containsKey(TailNode + "-" + P)) {
                    Arcs.put(TailNode + "-" + P, MaxDemand);
                }
            }
            if ((P.contains("Supplier")) || (P.contains("Candidate"))) {
                if (!Arcs.containsKey(P + "-" + HeadNode)) {
                    Arcs.put(P + "-" + HeadNode, MaxDemand);
                }
            }
        }

    }
}
for (int i = 0; i < ArrCandidate.length; i++) {
```

```
        for (int j = 0; j < ArrCandidate.length; j++) {
            if (i != j) {
                String C = ArrCandidate[i];
                String D = ArrCandidate[j];
                int Dis = Data.getDistance(D, C);
                if (Dis == 1) {
                    if (!Arcs.containsKey(C + "End-" + D + "Start")) {
                        Arcs.put(C + "End-" + D + "Start", MaxDemand);
                    }
                }
            }
        }
    }
    for (String C : ArrCandidate) {
        Points += C + "Start-";
        Points += C + "End-";
    }
    Points += "MainSupplier-";
    for (String T : ArrSupplier) {
        if (!Arcs.containsKey("MainSupplier-" + T)) {
            Arcs.put("MainSupplier-" + T, MaxDemand);
        }
    }
    Points += "MainDemand";
    for (String T : ArrDemands) {
        if (!Arcs.containsKey(T + "-MainDemand")) {
            Arcs.put(T + "-MainDemand", DemandScenario.get(T));
        }
    }
    ObjectiveValue = Obj;
}
public Solution(String InputCandidate, String InputCandidateType, DataReader InputData, String Tabu, double Obj) {
    this(InputCandidate, InputCandidateType, InputData, Tabu, Obj, null);
}
@Override
public Solution clone() {
    return new Solution(this.Candidates, this.CandidateType, Data, this.TabuMove, this.ObjectiveValue, this.Utilize);
}

public String[] Divider(String S) {
    return S.split("-");
}
public void changeCapacity(HashMap<String, Integer> InputDemandScenario) {
    // Update Capacity of Arcs
    for (String T : Divider(Demands)) {
        Arcs.put(T + "-MainDemand", InputDemandScenario.get(T));
    }
}
public double getObjective() throws CloneNotSupportedException, IloException {
    if (Data.getObjective(this.Candidates + "!" + this.CandidateType) != -1) {
        return Data.getObjective(this.Candidates + "!" + this.CandidateType);
    }
    // Calculate the Construction Cost
    double Obj = 0;
    if (!CandidateType.equals("")) {
        for (int i = 0; i < CandidateType.split("-").length; i++) {
```

```
                Obj += Data.CostConstruction[Integer.parseInt(CandidateType.split("-")[i]) - 1];
            }
        }
        this.changeCapacity(DemandScenario);
        HashMap<String, Integer> preflow = new HashMap<String, Integer>();
        Utilize = new HashMap<String, Double>();
        for (int i = 0; i < Horizon; i++) {
            CplexSolver CSolver = new CplexSolver(Points, Arcs);
            CSolver.Solve();
            HashMap<String, Integer> InputDemandScenario = new HashMap<String, Integer>();
            preflow = CSolver.getPreflow();
            for (String C : preflow.keySet()) {
                if (C.contains("-MainDemand")) {
                    // Update Arcs based on satisfied demand in this time slot
                    if (!InputDemandScenario.containsKey(C.split("-")[0])) {
                        InputDemandScenario.put(C.split("-")[0], Math.max(Arcs.get(C) - preflow.get(C), 0));
                    }
                }
                if ((C.contains("Start-")) && (C.contains("End"))) {
                    if (preflow.get(C) != 0) {
                        // Calculate the cost if a station is used
                        String Name = C.split("-")[0].replace("Start", "");
                        for (int j = 0; j < this.Candidates.split("-").length; j++) {
                            if (this.Candidates.split("-")[j].equals(Name)) {
                                Obj += Data.CostOperational[Integer.parseInt(this.CandidateType.split("-")[j]) - 1];
                            }
                        }
                    }
                }
            }
            //Calculate Stat
            this.updateStat(preflow);
            // Change the Capacity on the Arcs at the end of the time horizon
            this.changeCapacity(InputDemandScenario);
            CSolver.end();
        }
        for (String C : preflow.keySet()) {
            if (C.contains("-MainDemand")) {
                Obj += Data.CostLostSale * Arcs.get(C);
            }
        }
        Data.InsertObj(this.Candidates + "!" + this.CandidateType, Obj);
        ObjectiveValue = Obj;
        return Obj;
    }
    public Solution removeCandidate(int R) throws CloneNotSupportedException, IloException {
        String[] C;
        String[] T;
        String AnsC = "";
        String AnsT = "";
        String Name = "";
        if (!this.Candidates.equals("")) {
            Name = this.Candidates.split("-")[R];
            Utilize.remove(Name);
            C = this.Candidates.split("-");
            T = this.CandidateType.split("-");
```

```java
        for (int i = 0; i < C.length; i++) {
            if (!C[i].equals(Name)) {
                AnsC += C[i] + "-";
                AnsT += T[i] + "-";
            }
        }
        if (AnsC.equals("")) {
            this.Candidates = "";
            this.CandidateType = "";
        } else {
            this.Candidates = AnsC.substring(0, AnsC.length() - 1);
            this.CandidateType = AnsT.substring(0, AnsT.length() - 1);
        }
    }
    return new Solution(this.Candidates, this.CandidateType, Data, Name + "-Remove2");
}
public Solution InsertCandidate() throws CloneNotSupportedException, IloException {
    String[] PotentialCandidate = Data.getListType(DataReader.PointType.Candidate);
    String C = this.Candidates;
    List<String> ActualCandidate = new ArrayList<String>();
    for (String C1 : PotentialCandidate) {
        if (!C.contains(C1)) {
            ActualCandidate.add(C1);
        }
    }
    String Select = "";
    if (!ActualCandidate.isEmpty()) {
        Select = ActualCandidate.get(Data.R.nextInt(ActualCandidate.size()));
        if (!this.Candidates.equals("")) {
            this.Candidates += "-" + Select;
            this.CandidateType += "-1";
        } else {
            this.Candidates += Select;
            this.CandidateType += "1";
        }
    }
    return new Solution(this.Candidates, this.CandidateType, Data, Select + "-Insert");
}
private void updateStat(HashMap<String, Integer> preflow) {
    for (String C : preflow.keySet()) {
        if ((C.contains("Start-")) && (C.contains("End"))) {
            String Name = C.split("-")[0].replace("Start", "");
            double Utl = 0;
            if (Utilize.get(Name) != null) {
                Utl = Utilize.get(Name);
            }
            Utilize.put(Name, Utl + (preflow.get(C) * 1.0) / (Horizon * Arcs.get(C)));
        }
    }
}
public Solution removeCandidate(String NameMin) throws CloneNotSupportedException, IloException {
    String[] C;
    String[] T;
    String AnsC = "";
    String AnsT = "";
    Utilize.remove(NameMin);
```

```
        if (!this.Candidates.equals("")) {
            C = this.Candidates.split("-");
            T = this.CandidateType.split("-");
            for (int i = 0; i < C.length; i++) {
                if (!NameMin.equals(C[i])) {
                    AnsC += C[i] + "-";
                    AnsT += T[i] + "-";
                }
            }
            if (AnsC.equals("")) {
                this.Candidates = "";
                this.CandidateType = "";
            } else {
                this.Candidates = AnsC.substring(0, AnsC.length() - 1);
                this.CandidateType = AnsT.substring(0, AnsT.length() - 1);
            }
        }
        return new Solution(this.Candidates, this.CandidateType, Data, NameMin + "-Remove1");
    }
    public Solution InsertCandidate(String Name) throws CloneNotSupportedException, IloException {
        if (!this.Candidates.equals("")) {
            this.Candidates += "-" + Name;
            this.CandidateType += "-1";
        } else {
            this.Candidates = Name;
            this.CandidateType = "1";
        }
        return new Solution(this.Candidates, this.CandidateType, Data, Name + "-Insert");
    }
    int getType(String C) {
        for (int i = 0; i < this.Candidates.split("-").length; i++) {
            if (this.Candidates.split("-")[i].equals(C)) {
                return Integer.parseInt(this.CandidateType.split("-")[i]);
            }
        }
        return -1;
    }
    void changeMove(String string) {
        TabuMove = string;
    }
    public String writeAnswer() throws CloneNotSupportedException, IloException {
        String AnsFinal = "";
        AnsFinal = Double.toString(this.ObjectiveValue) + "\t";
        this.changeCapacity(DemandScenario);
        HashMap<String, Integer> preflow = new HashMap<String, Integer>();
        Utilize = new HashMap<String, Double>();
        for (int i = 0; i < Horizon; i++) {
            // Calculate the Max flow Using the Updated or Initial Arc
            CplexSolver Solver = new CplexSolver(Points, Arcs);
            Solver.Solve();
            HashMap<String, Integer> InputDemandScenario = new HashMap<String, Integer>();
            preflow = Solver.getPreflow();
            for (String C : preflow.keySet()) {
                if (C.contains("-MainDemand")) {
                    // Update Arcs based on satisfied demand in this time slot
                    if (!InputDemandScenario.containsKey(C.split("-")[0])) {
```

```java
                    InputDemandScenario.put(C.split("-")[0], Math.max(Arcs.get(C) - preflow.get(C), 0));
                }
            }
        }
        //Calculate Stat
        this.updateStat(preflow);
        // Change the Capacity on the Arcs at the end of the time horizon
        this.changeCapacity(InputDemandScenario);
    }
    int LsS = 0;
    for (String C : preflow.keySet()) {
        if (C.contains("-MainDemand")) {
            LsS += Arcs.get(C);
        }
    }
    AnsFinal += Integer.toString(LsS) + "\t";
    return AnsFinal;
}
public String Latex(int Period) throws CloneNotSupportedException, IloException {
    String Ans = "";
    Ans += ("\\begin{tikzpicture}[scale = 0.5]") + "\n";
    Ans += ("\\tikzstyle{every node} = [draw, shape=circle, minimum size = 12pt]") + "\n";
    Ans += ("\\path (0,0) node (-1) {S};") + "\n";
    HashMap<String, Integer> P = new HashMap<String, Integer>();
    HashMap<String, String> D = new HashMap<String, String>();
    for (int i = 0; i < this.Candidates.split("-").length; i++) {
        Ans += ("\\path (" + Data.getCor(this.Candidates.split("-")[i]) + ") node (C" + Integer.toString(i) + ") {" +
               this.Candidates.split("-")[i].replace("Candidate", "") + "," + this.CandidateType.split("-")[i] + "};") + "\n";
        P.put(this.Candidates.split("-")[i].replace("Candidate", ""), i);
    }
    String Sr = Data.getDemandPoint();
    for (String S : Sr.split("-")) {
        D.put(S, Data.getCor(S));
    }
    this.changeCapacity(DemandScenario);
    HashMap<String, Integer> preflow = new HashMap<String, Integer>();
    Utilize = new HashMap<String, Double>();
    Ans += ("\\tikzstyle{every node} = []") + "\n";
    for (int i = 0; i < Horizon; i++) {
        // Calculate the Max flow Using the Updated or Initial Arc
        CplexSolver Solver = new CplexSolver(Points, Arcs);
        Solver.Solve();
        HashMap<String, Integer> InputDemandScenario = new HashMap<String, Integer>();
        preflow = Solver.getPreflow();
        for (String C : preflow.keySet()) {
            if (C.contains("-MainDemand")) {
                // Update Arcs based on satisfied demand in this time slot
                if (!InputDemandScenario.containsKey(C.split("-")[0])) {
                    InputDemandScenario.put(C.split("-")[0], Math.max(Arcs.get(C) - preflow.get(C), 0));
                }
            }
            if (preflow.get(C) != 0) {
                if (i == Period) {
                    int dem = preflow.get(C);
                    if (!(C.contains("Start-") && C.contains("End"))) {
                        if (C.contains("End-") && C.contains("Start")) {
```

```
                        C = C.replace("Candidate", "");
                        C = C.replace("End", "");
                        C = C.replace("Start", "");
                        Ans += ("\\path (C" + Integer.toString(P.get(C.split("-")[0])) + ") edge[->, >=stealth'] node[above] {" +
                               Integer.toString(dem) + "} (C" + Integer.toString(P.get(C.split("-")[1])) + ");") + "\n";
                    }
                    if (C.contains("Supplier") && (!C.contains("MainSupplier"))) {
                        if (C.contains("Demand")) {
                            C = C.replace("Supplier1-", "");
                            Ans += ("\\path (-1) edge[->, >=stealth',color=red] node[above] {" + Integer.toString(dem) + "} (" + D.get(C) +
                                   ");") + "\n";
                        } else if (C.contains("Candidate")) {
                            C = C.replace("Supplier1-Candidate", "");
                            C = C.replace("Start", "");
                            Ans += ("\\path (-1) edge[->, >=stealth'] node[above] {" + Integer.toString(dem) + "} (C" +
                                   Integer.toString(P.get(C)) + ");") + "\n";
                        }
                    }
                    if (C.contains("Demand") && C.contains("Candidate")) {
                        C = C.replace("Candidate", "");
                        C = C.replace("End", "");
                        Ans += ("\\path (C" + Integer.toString(P.get(C.split("-")[0])) + ") edge[->, >=stealth',color=red] node[above] {" +
                               Integer.toString(dem) + "} (" + D.get(C.split("-")[1]) + ");") + "\n";
                    }
                }
            }
        }
    }
    //Calculate Stat
    this.updateStat(preflow);
    // Change the Capacity on the Arcs at the end of the time horizon
    this.changeCapacity(InputDemandScenario);
  }
  Ans += ("\\end{tikzpicture}");
  return Ans;
}
void writeAnswer(int Period) throws IloException {
  this.changeCapacity(DemandScenario);
  HashMap<String, Integer> preflow = new HashMap<String, Integer>();
  Utilize = new HashMap<String, Double>();
  for (int i = 0; i < Horizon; i++) {
    // Calculate the Max flow Using the Updated or Initial Arc
    CplexSolver Solver = new CplexSolver(Points, Arcs);
    Solver.Solve();
    HashMap<String, Integer> InputDemandScenario = new HashMap<String, Integer>();
    preflow = Solver.getPreflow();
    for (String C : preflow.keySet()) {
      if (C.contains("-MainDemand")) {
        // Update Arcs based on satisfied demand in this time slot
        if (!InputDemandScenario.containsKey(C.split("-")[0])) {
          InputDemandScenario.put(C.split("-")[0], Math.max(Arcs.get(C) - preflow.get(C), 0));
        }
      }
    }
    if (i == Period) {
      System.out.println("Operational at " + Integer.toString(i + 1));
```

```
            for (String C : preflow.keySet()) {
                if ((C.contains("Start-")) && (C.contains("End"))) {
                    if (preflow.get(C) != 0) {
                        // Calculate the cost if a station is used
                        String Name = C.split("-")[0].replace("Start", "");
                        System.out.println(Name);
                    }
                }
            }
        }
        //Calculate Stat
        this.updateStat(preflow);
        // Change the Capacity on the Arcs at the end of the time horizon
        this.changeCapacity(InputDemandScenario);
    }
}
}
```

A.8 CPLEX Solver Class

```
package bendersversion;
import ilog.concert.IloException;
import ilog.concert.IloIntVar;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.concert.IloRange;
import ilog.cplex.IloCplex;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.HashMap;
/**
 *
 * @author aghahari
 */
public class BendersClass {
    private static double UAVmax;
    protected IloCplex Master;   // the master model
    //Master Variable
    protected IloNumVar FlowCost; // Sirrogate Variable Sub/Master
    protected IloIntVar[][] Z;
    protected IloIntVar[][][] O;
    //Master Data
    protected int NumCandidates;
    protected int NumTypes;
    protected double[] ConstructionCost;
    protected double[] OperationalCost;
    protected int NumTime;
    protected HashMap<String, String[]> Connection;
    protected HashMap<String, String[]> RevConnection;
```

```
//End Master
// Sub Variable
protected IloNumVar[][] X;
protected IloNumVar[][] V;
protected IloNumVar[] L;
// Sub Data
protected int[] Demand;
protected int[] BatteryCapacity;
protected double LostCost;
protected int NumAllPoint;
protected HashMap<String, Integer> SIArcs;
protected HashMap<Integer, String> ISArcs;
// End Sub
protected IloRange TrustRegion;
protected final double FUZZ = 1.0e-7;
protected final int NumArcs;
protected int NumDemand;
protected int NumSupply;
protected HashMap<String, Integer> CanTime;
private final String FileName;
private double[][] IValue;
public BendersClass(String InputName) throws IloException, FileNotFoundException, IOException {
    // READ DATA //
    BufferedReader Input = new BufferedReader(new FileReader(InputName));
    FileName = InputName;
    UAVmax = Integer.parseInt(Input.readLine());
    NumCandidates = Integer.parseInt(Input.readLine());
    NumTypes = Integer.parseInt(Input.readLine());
    NumTime = Integer.parseInt(Input.readLine());
    ConstructionCost = new double[NumTypes];
    for (int i = 0; i < NumTypes; i++) {
        ConstructionCost[i] = Double.parseDouble(Input.readLine());
    }
    OperationalCost = new double[NumTypes];
    for (int i = 0; i < NumTypes; i++) {
        OperationalCost[i] = Double.parseDouble(Input.readLine());
    }
    BatteryCapacity = new int[NumTypes];
    for (int i = 0; i < NumTypes; i++) {
        BatteryCapacity[i] = Integer.parseInt(Input.readLine());
    }
    HashMap<String, String> Allpoints = new HashMap<String, String>();
    for (int i = 0; i < NumCandidates; i++) {
        Allpoints.put("C" + i, Input.readLine());
    }
    NumDemand = Integer.parseInt(Input.readLine());
    Demand = new int[NumDemand];
    for (int i = 0; i < NumDemand; i++) {
        String Line = Input.readLine();
        Allpoints.put("D" + i, Line);
        Demand[i] = Integer.parseInt(Line.split(",")[2]);
    }
    NumSupply = Integer.parseInt(Input.readLine());
    for (int i = 0; i < NumSupply; i++) {
        Allpoints.put("S" + i, Input.readLine());
    }
```

```
Connection = new HashMap<String, String[]>();
RevConnection = new HashMap<String, String[]>();
for (String S : Allpoints.keySet()) {
    String[] A = new String[Allpoints.size() + 1];
    java.util.Arrays.fill(A, "0");
    Connection.put(S, A);
    RevConnection.put(S, A);
}
for (String S1 : Allpoints.keySet()) {
    for (String S2 : Allpoints.keySet()) {
        if (!S1.contains("D")) {
            if (!S1.equals(S2)) {
                if (DistanceLinear(Allpoints.get(S2), Allpoints.get(S1)) == 1) {
                    if (!S2.contains("S")) {
                        String[] F = Connection.get(S1).clone();
                        String[] R = RevConnection.get(S2).clone();
                        F[0] = Integer.toString(Integer.parseInt(F[0]) + 1);
                        R[0] = Integer.toString(Integer.parseInt(R[0]) + 1);
                        F[Integer.parseInt(F[0])] = S2;
                        R[Integer.parseInt(R[0])] = S1;
                        Connection.put(S1, F);
                        RevConnection.put(S2, R);
                    }
                }
            }
        }
    }
}
LostCost = Double.parseDouble(Input.readLine());
// END DATA //
Master = new IloCplex();
// Setup Master
// Variables
String[][] CandidNames = new String[NumCandidates][NumTypes];
IValue = new double[NumCandidates][NumTypes];
String[][][] OperationNames = new String[NumCandidates][NumTypes][NumTime];
for (int i = 0; i < NumCandidates; i++) {
    for (int j = 0; j < NumTypes; j++) {
        CandidNames[i][j] = "Candidate_" + i + "_" + j;
        IValue[i][j] = 0;
        for (int k = 0; k < NumTime; k++) {
            OperationNames[i][j][k] = "Operational_" + Integer.toString(i + 1) + "_" + Integer.toString(j + 1) + "_" + Integer.toString(k +
                    1);
        }
    }
}
Z = new IloIntVar[NumCandidates][NumTypes];
O = new IloIntVar[NumCandidates][NumTypes][NumTime];
for (int i = 0; i < NumCandidates; i++) {
    Z[i] = Master.boolVarArray(NumTypes, CandidNames[i]);
    for (int j = 0; j < NumTypes; j++) {
        O[i][j] = Master.boolVarArray(NumTime, OperationNames[i][j]);
    }
}
FlowCost = Master.numVar(0.0, Double.MAX_VALUE, "EstimateZ");
// Modeling
```

```
IloLinearNumExpr exprMaster = Master.linearNumExpr();
// Objective Function
for (int i = 0; i < NumCandidates; i++) {
    for (int j = 0; j < NumTypes; j++) {
        exprMaster.addTerm(ConstructionCost[j], Z[i][j]);
        for (int k = 0; k < NumTime; k++) {
            exprMaster.addTerm(OperationalCost[j], O[i][j][k]);
        }
    }
}
exprMaster.addTerm(1.0, FlowCost);
Master.addMinimize(exprMaster);
// Constraint 1
for (int i = 0; i < NumCandidates; i++) {
    for (int j = 0; j < NumTypes; j++) {
        exprMaster.clear();
        for (int k = 0; k < NumTime; k++) {
            exprMaster.addTerm(1.0, O[i][j][k]);
        }
        exprMaster.addTerm(-1.0 * NumTime, Z[i][j]);
        Master.addLe(exprMaster, 0, "Operational_Candidate" + i + "_Type" + j);
    }
}
// Constraint 2
for (int i = 0; i < NumCandidates; i++) {
    exprMaster.clear();
    for (int j = 0; j < NumTypes; j++) {
        exprMaster.addTerm(1.0, Z[i][j]);
    }
    String[] Adj = Connection.get("C" + i);
    boolean add = true;
    for (int j = 1; j <= Integer.parseInt(Adj[0]); j++) {
        if (Adj[j].contains("C")) {
            for (int k = 0; k < NumTypes; k++) {
                exprMaster.addTerm(-1.0, Z[Integer.parseInt(Adj[j].substring(1))][k]);
            }
        } else if (Adj[j].contains("S")) {
            add = false;
            break;
        }
    }
    if (add) {
        Master.addLe(exprMaster, 0, "Establishment_" + i);
    }
}
//Constraint 3
for (int i = 0; i < NumCandidates; i++) {
    exprMaster.clear();
    for (int j = 0; j < NumTypes; j++) {
        exprMaster.addTerm(1.0, Z[i][j]);
    }
    Master.addLe(exprMaster, 1.0, "LocationSetup" + i);
}
//End Master
// Setup Subproblem
IloLinearNumExpr exprSub = Master.linearNumExpr();
```

```
int Counter = 0;
SIArcs = new HashMap<String, Integer>();
ISArcs = new HashMap<Integer, String>();
for (String S : Connection.keySet()) {
    String[] Adj = Connection.get(S);
    for (int i = 1; i <= Integer.parseInt(Adj[0]); i++) {
        SIArcs.put("!" + S + "-" + Adj[i] + "!", Counter);
        ISArcs.put(Counter, "!" + S + "-" + Adj[i] + "!");
        Counter++;
    }
}
NumArcs = SIArcs.size();
X = new IloNumVar[NumArcs][NumTime];
L = new IloNumVar[NumDemand];
V = new IloNumVar[NumDemand + NumSupply][NumTime];
for (int i = 0; i < NumArcs; i++) {
    for (int j = 0; j < NumTime; j++) {
        X[i][j] = Master.numVar(0, Double.MAX_VALUE, "Shipment_" + ISArcs.get(i) + "_" + j);
    }
}
for (int i = 0; i < NumDemand; i++) {
    L[i] = Master.numVar(0, Double.MAX_VALUE, "LostSale_" + i);
}
for (int i = 0; i < NumDemand; i++) {
    for (int j = 0; j < NumTime; j++) {
        V[i][j] = Master.numVar(0, Double.MAX_VALUE, "Arrived_" + i + "_" + j);
    }
}
for (int i = NumDemand; i < NumDemand + NumSupply; i++) {
    for (int j = 0; j < NumTime; j++) {
        V[i][j] = Master.numVar(0, Double.MAX_VALUE, "Send_" + i + "_" + j);
    }
}
// Objective
exprSub.clear();
for (int i = 0; i < NumDemand; i++) {
    exprSub.addTerm(LostCost, L[i]);
}
exprSub.addTerm(-1.0, FlowCost);
Master.addEq(exprSub, 0.0);
// Constraint 1
for (String S : Connection.keySet()) {
    for (int i = 0; i < NumTime; i++) {
        exprSub.clear();
        String[] Adj = Connection.get(S);
        String[] RevAdj = RevConnection.get(S);
        for (int j = 1; j <= Integer.parseInt(Adj[0]); j++) {
            exprSub.addTerm(1.0, X[SIArcs.get("!" + S + "-" + Adj[j] + "!")][i]);
        }
        for (int j = 1; j <= Integer.parseInt(RevAdj[0]); j++) {
            exprSub.addTerm(-1.0, X[SIArcs.get("!" + RevAdj[j] + "-" + S + "!")][i]);
        }
        if (S.contains("S")) {
            exprSub.addTerm(-1.0, V[NumDemand + Integer.parseInt(S.substring(1))][i]);
        } else if (S.contains("D")) {
            exprSub.addTerm(1.0, V[Integer.parseInt(S.substring(1))][i]);
```

```java
            }
            Master.addEq(exprSub, 0.0, "FlowBalance_" + S);
        }
    }
    //Constraint 2
    for (int i = 0; i < NumTime; i++) {
        exprSub.clear();
        for (int j = 0; j < NumDemand; j++) {
            exprSub.addTerm(1.0, V[j][i]);
        }
        for (int j = NumDemand; j < NumDemand + NumSupply; j++) {
            exprSub.addTerm(-1.0, V[j][i]);
        }
        Master.addEq(exprSub, 0.0, "Feasibility_" + i);
    }
    //Constraint 3
    for (int i = 0; i < NumDemand; i++) {
        exprSub.clear();
        exprSub.addTerm(1.0, L[i]);
        for (int j = 0; j < NumTime; j++) {
            exprSub.addTerm(1.0, V[i][j]);
        }
        Master.addGe(exprSub, Demand[i], "LostSale_" + i);
    }
    //Constraint 4
    for (int i = 0; i < NumDemand; i++) {
        exprSub.clear();
        for (int j = 0; j < NumTime; j++) {
            exprSub.addTerm(1.0, V[i][j]);
        }
        Master.addLe(exprSub, Demand[i], "OverSale_" + i);
    }
    //Constraint 5
    int Count = 0;
    CanTime = new HashMap<String, Integer>();
    for (int i = 0; i < NumCandidates; i++) {
        for (int j = 0; j < NumTime; j++) {
            String[] Adj = Connection.get("C" + i);
            exprSub.clear();
            for (int k = 1; k <= Integer.parseInt(Adj[0]); k++) {
                exprSub.addTerm(1.0, X[SIArcs.get("!C" + i + "-" + Adj[k] + "!")][j]);
            }
            for (int k = 0; k < NumTypes; k++) {
                exprSub.addTerm(-1 * BatteryCapacity[k], O[i][k][j]);
            }
            Master.addLe(exprSub, 0.0, "OperationalCost_" + i);
            Count++;
        }
    }
    //Invalid inequality
        for (int i = 0; i < NumCandidates; i++) {
            String[] RevAdj = RevConnection.get("C" + Integer.toString(i).trim());
            exprSub.clear();
            for (int j = 0; j < NumTypes; j++) {
                exprSub.addTerm(1.0, Z[i][j]);
            }
```

```
            boolean Supplier = false;
            for (int j = 1; j <= Integer.parseInt(RevAdj[0]); j++) {
                if (RevAdj[j].contains("S")) {
                    Supplier = true;
                    break;
                } else {
                    int Su = Integer.parseInt(RevAdj[j].replace("C", ""));
                    for (int k = 0; k < NumTypes; k++) {
                        exprSub.addTerm(Z[Su][k], -1.0);
                    }
                }
            }
            if (!Supplier) {
                Master.addLe(exprSub, 0.0);
            }
        }
    Master.setWarning(null);
    Master.setName(InputName);
    Master.setParam(IloCplex.DoubleParam.TiLim, 60 * 60 * 1);
}
int CCut = 0;
void end() {
    Master.end();
}
private Solution PureDegrade(Solution CurrentSolution) throws IloException {
    Solution CS = CurrentSolution.Copy();
    boolean Change = true;
    while (Change) {
        Change = false;
        for (Station S : CS.ListStations) {
            CS.Degrade(S);
            Solution NS = CS.CopyNoRepair();
            if (NS.getObj() < CS.getObj()) {
                CS = NS.Copy();
                Change = true;
            } else {
                CS.Upgrade(S);
            }
        }
    }
    return CS.Copy();
}
private Solution Initialize() throws IOException, IloException, Exception {
    AI Baran = new AI(FileName);
    Solution BestSoFar = null;
    int min = Baran.getMinDistance();
    int max = Baran.getMaxDistance();
    for (int i = min; i <= max; i++) {
        Baran = new AI(FileName);
        Baran.Run(i);
        Solution Current = new Solution(Baran.getData(), Baran.getSelectedStation());
        Current = PureRemove(Current);
        Current = PureDegrade(Current);
        System.out.println("Obj at Move " + Integer.toString(i) + "\t" + Current.getObj());
        if (BestSoFar != null) {
            if (Current.getObj() < BestSoFar.getObj()) {
```

```
                BestSoFar = Current;
            }
        } else {
            BestSoFar = Current.Copy();
        }
        Baran.release();
        Baran = null;
    }
    Baran = new AI(FileName);
    DemandPoint[] Dems = Baran.getDemandList();
    for (DemandPoint Dem : Dems) {
        Baran = new AI(FileName);
        Baran.Run(Dem);
        if (Baran.getSelectedStation().length != 1) {
            Solution Current = new Solution(Baran.getData(), Baran.getSelectedStation());
            Current = PureRemove(Current);
            Current = PureDegrade(Current);
            System.out.println("Obj at DemandPoint " + Dem.getName() + "\t" + Current.getObj());
            if (BestSoFar != null) {
                if (Current.getObj() < BestSoFar.getObj()) {
                    BestSoFar = Current;
                }
            } else {
                BestSoFar = Current.Copy();
            }
            Baran.release();
            Baran = null;
        }
    }
    return BestSoFar;
}
private Solution PureRemove(Solution CurrentSolution) throws IloException {
    Solution CS = CurrentSolution.Copy();
    boolean Change = true;
    while (Change) {
        Change = false;
        for (Station S : CS.ListStations) {
            CS.Deselect(S);
            Solution NS = CS.CopyNoRepair();
            if (NS.getObj() < CS.getObj()) {
                CS = NS.Copy();
                Change = true;
            } else {
                CS.Select(S, 3);
            }
        }
    }
    return CS.Copy();
}

public void solve() throws IloException, IOException, Exception {
    AI Baran = new AI(FileName);
    Baran.Run();
    Station[] SList = Baran.getSelectedStation();
    for (Station S : SList) {
        if (S.getID() != -1) {
```

```
            IValue[S.getID()][S.getType() - 1] = 1;
         }
      }
      for (int i = 0; i < NumCandidates; i++) {
         Master.addMIPStart(Z[i], IValue[i]);
      }
      Master.setOut(null);
      Master.setWarning(null);
      Master.solve();

   }
   public String Summary() throws IloException {
      return Master.getName() + "\t\t" + Master.getStatus() + "\t\t" + Master.getMIPRelativeGap() + "\t\t" + Master.getObjValue() + "\t\t" +
            Master.getBestObjValue() + "\t\t" + "NOTIME" + "\t\t" + Integer.toString((int) Master.getValue(FlowCost) / (int) LostCost);// +
            Master.getCplexTime();
   }
   public void FullReport() throws IOException, IloException {
      BufferedWriter FullReportFile = new BufferedWriter(new FileWriter("FullReport//" + Master.getName()));
      String Ans = "Stations = " + "\n";
      double o[][][] = new double[NumCandidates][NumTypes][NumTime];
      for (int i = 0; i < NumCandidates; i++) {
         for (int j = 0; j < NumTypes; j++) {
            o[i][j] = Master.getValues(O[i][j]);
         }
      }
      for (int i = 0; i < NumCandidates; i++) {
         for (int j = 0; j < NumTime; j++) {
            double Cap = 0;
            for (int k = 0; k < NumTypes; k++) {
               if (o[i][k][j] > 0.95) {
                  Ans += O[i][k][j].getName() + "\n";
               }
            }
         }
      }
      Master.writeSolution("ANSWER.sol");
      FullReportFile.write(Ans);
      FullReportFile.close();

   }
   public static int DistanceLinear(String P1, String P2) {
      double X1 = Double.parseDouble(P1.split(",")[0]);
      double Y1 = Double.parseDouble(P1.split(",")[1]);
      double X2 = Double.parseDouble(P2.split(",")[0]);
      double Y2 = Double.parseDouble(P2.split(",")[1]);
      double d = Math.sqrt(Math.pow(X1 - X2, 2) + Math.pow(Y1 - Y2, 2));
      if ((d > UAVmax) || (d == 0)) {
         return 0;
      } else {
         return 1;
      }
   }
}
```

B Chapter 3 Programming Codes

B.1 Network Class

```
package chapter2;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
/**
 *
 * @author aghahari
 */
public class Network {
    private final ArrayList<Ramp> RampList;
    private final ArrayList<Customer> CustomerList;
    private final ArrayList<Node> AllLocation;
    private final int ScenarioCounter;
    private final int TimeLineCounter;
    public Network(String FileName) throws FileNotFoundException, IOException {
        RampList = new ArrayList<>();
        CustomerList = new ArrayList<>();
        AllLocation = new ArrayList<>();
        BufferedReader Input = new BufferedReader(new FileReader(FileName + "//Parameters.txt"));
        int RampCount = Integer.parseInt(Input.readLine().split(",")[1]);
        int CustomerCount = Integer.parseInt(Input.readLine().split(",")[1]);
        TimeLineCounter = Integer.parseInt(Input.readLine().split(",")[1]);
        ScenarioCounter = Integer.parseInt(Input.readLine().split(",")[1]);
        Input.close();
        Input = new BufferedReader(new FileReader(FileName + "//RampInfo.txt"));
        Input.readLine();
        for (int i = 0; i < RampCount; i++) {
            String[] RampString = Input.readLine().split(",");
            Ramp R = new Ramp(i, RampString[0], Double.parseDouble(RampString[1]), Double.parseDouble(RampString[2]));
            RampList.add(R);
            AllLocation.add(R);
        }
        for (Ramp R1 : RampList) {
            for (Ramp R2 : RampList) {
                if (!R1.isEqual(R2)) {
                    R1.AddAdjacent(R2);
                }
            }
        }
        Input.close();
        if (Constants.CustomerMode) {
            Input = new BufferedReader(new FileReader(FileName + "//CustomerInfo.txt"));
            Input.readLine();
            for (int i = 0; i < CustomerCount; i++) {
                String[] RampString = Input.readLine().split(",");
                Customer C = new Customer(i, RampString[0], Double.parseDouble(RampString[1]), Double.parseDouble(RampString[2]), RampList);
                CustomerList.add(C);
```

```
                AllLocation.add(C);
            }
        }
        HashMap<String,String> Lines = new HashMap<>();
        Input = new BufferedReader(new FileReader(FileName + "//CustomerDemandInfo.txt"));
        Input.readLine();
    }
}
```

B.2 Node Class

```
package chapter2;
import java.io.IOException;
import java.util.ArrayList;
import org.rosuda.REngine.REXPMismatchException;
import org.rosuda.REngine.REngineException;
/**
 *
 * @author aghahari
 */
public class Node {
    private final int ID;
    private final String Name;
    private final ArrayList<TimeLine> TimeList;
    private final double Latt;
    private final double Long;
    private final ArrayList<Adjacency> ForwardAdjacent;
    private final ArrayList<Adjacency> ReverseAdjacent;
    public Node(int ID, String Name, double Latt, double Long) {
        this.ID = ID;
        this.Name = Name;
        this.Latt = Latt;
        this.Long = Long;
        TimeList = new ArrayList<>();
        ForwardAdjacent = new ArrayList<>();
        ReverseAdjacent = new ArrayList<>();
        int[] ForecastNumber = null;
        if (!Constants.HistoricalDemandData.getOrDefault(Name, "NONE").equals("NONE")) {
            try {
                ForecastNumber = Constants.Forecast(this.Name);
                System.out.println(this.Name + "\tDone");
                for (int i = 0; i < ForecastNumber.length; i++) {
                    TimeList.add(new TimeLine(i));
                    TimeList.get(i).AddTime(ForecastNumber[i]);
                }
            } catch (REXPMismatchException ex) {
                System.out.println(this.Name + "\tError In Forecasting (REXPRESSION)\t" + ex.toString());
            } catch (REngineException ex) {
                System.out.println(this.Name + "\tError In Forecasting (RENGINE)\t" + ex.toString());
            } catch (IOException ex) {
                System.out.println(this.Name + "\tError In Forecasting (I/O Exception)\t" + ex.toString());
            }
        } else {
```

```
            System.out.println(this.Name + "\tNONE!");
        }
    }
    public int getID() {
        return ID;
    }
    public String getName() {
        return Name;
    }
    public double getLatt() {
        return Latt;
    }
    public double getLong() {
        return Long;
    }
    /**
     *
     * @param ID
     * @param Scenario
     * @param Amount Add a Time (Positive is Supply and Negative is Demand)
     */
    public void AddTime(int ID, int Scenario, int Amount) {
        if (Scenario < TimeList.size()) {
            TimeList.get(Scenario).AddTime(ID, Amount);
        } else {
            throw new IllegalArgumentException("This Scenario is not Exist!");
        }
    }
    /**
     *
     * @param Scenario
     * @param Amount Add a Time at the end of list (Positive is Supply and
     * Negative is Demand)
     */
    public void AddTime(int Scenario, int Amount) {
        TimeList.get(Scenario).AddTime(Amount);
    }
    public void AddForwardAdjacent(Node Adj) {
        ForwardAdjacent.add(new Adjacency(this, Adj));
    }
    public void AddReverseAdjacent(Node Adj) {
        ReverseAdjacent.add(new Adjacency(Adj, this));
    }
    public void AddBothAdjacency(Node Adj) {
        AddForwardAdjacent(Adj);
        AddReverseAdjacent(Adj);
    }
    public void AddAdjacent(Node Adj) {
        if ((this instanceof Ramp) && (Adj instanceof Ramp)) {
            if (Constants.isRailData((Ramp) this, (Ramp) Adj)) {
                AddBothAdjacency(Adj);
            }
        } else if ((this instanceof Customer) && (Adj instanceof Customer)) {
            if (Constants.DistanceCalculator(this, Adj) < Constants.DrivingRadius) {
                AddBothAdjacency(Adj);
            }
```

```
        } else {
            AddBothAdjacency(Adj);
        }
    }
    /**
     *
     * @return Size of TimeLine
     */
    public int TimeSize() {
        return TimeList.size();
    }
    /**
     *
     * @return Size of adjacent Nodes.
     */
    public int AdjSize() {
        return ForwardAdjacent.size();
    }
}
```

B.3 Customer/Ramp Class

```
package chapter2;
import java.util.ArrayList;
/**
 *
 * @author aghahari
 */
public class Customer extends Node {
    ArrayList<Ramp> RampGroup;
    public Customer(int ID, String Name, double Latt, double Long, ArrayList<Ramp> RampList) {
        super(ID, Name, Latt, Long);
        Customer aThis = this;
        RampGroup = new ArrayList<>();
        for (Ramp R : RampList) {
            if (Constants.DistanceCalculator(R, aThis) < Constants.RampGroupRadius) {
                RampGroup.add(R);
                R.AddAdjacent(aThis);
            }
        }
    }
}
```

```
package chapter2;
/**
 *
 * @author aghahari
 */
public class Ramp extends Node {
    public Ramp(int ID, String Name, double Latt, double Long) {
        super(ID, Name, Latt, Long);
    }
```

```
    boolean isEqual(Ramp R2) {
        return this.getID() == R2.getID();
    }
}
```

B.4 Forcaster Class

```
package chapter2;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import org.rosuda.REngine.REXPMismatchException;
import org.rosuda.REngine.REngineException;
import org.rosuda.REngine.Rserve.RConnection;
import org.rosuda.REngine.Rserve.RserveException;
/**
 *
 * @author aghahari
 */
public class Forecaster {
    RConnection connection;
    private int max = 2;
    public Forecaster() throws RserveException {
        connection = new RConnection();
        connection.eval("library(forecast)");
        connection.eval("require(forecast)");
    }
    public void LoadData(String Address, int Skip) throws RserveException {
        connection.eval("data<-scan('" + Address + "',skip=" + Integer.toString(Skip) + ");");
    }
    public void LoadData(String Address) throws RserveException {
        LoadData(Address, 0);
    }
    public void setData(double[] Data) throws RserveException {
        String data = "c(";
        for (int i = 0; i < Data.length; i++) {
            data += Double.toString(Data[i]) + ",";
        }
        data = data.substring(0, data.length() - 1) + ")";
        connection.eval("data<-" + data);
    }
    public void setData(int[] Data) throws RserveException {
        String data = "c(";
        for (int i = 0; i < Data.length; i++) {
            data += Double.toString(Data[i]) + ",";
        }
        data = data.substring(0, data.length() - 1) + ")";
        connection.eval("data<-" + data);
    }
    public int[] Forecast(int Head, String Code) throws RserveException, REXPMismatchException, REngineException, IOException {
        double min = Double.MAX_VALUE;
        int pOptimal = -1;
```

```
    int qOptimal = -1;
    int spOptimal = -1;
    int sqOptimal = -1;
    int minimum = 0;
    for (int i = minimum; i < max; i++) {
        for (int j = minimum; j < max; j++) {
            for (int k = minimum; k < max; k++) {
                for (int l = minimum; l < max; l++) {
                    try {
                        connection.voidEval("fun <- arima(data, order = c(" + Integer.toString(i) + ", 1, " + Integer.toString(j) + "),seasonal =
                              list(order = c(" + Integer.toString(k) + ", 0, " + Integer.toString(l) + "), period = 52))");
                        if (min > connection.eval("fun$aic").asDouble()) {
                            pOptimal = i;
                            qOptimal = j;
                            spOptimal = k;
                            sqOptimal = l;
                        }
                    } catch (Exception ex) {

                    }
                }
            }
        }
    }
    System.out.println("order = c(" + Integer.toString(pOptimal) + ", 1, " + Integer.toString(qOptimal) + "),seasonal = list(order = c(" +
          Integer.toString(spOptimal) + ", 0, " + Integer.toString(sqOptimal) + ")");
    connection.eval("fun <- arima(data, order = c(" + Integer.toString(pOptimal) + ", 1, " + Integer.toString(qOptimal) + "),seasonal =
          list(order = c(" + Integer.toString(spOptimal) + ", 0, " + Integer.toString(sqOptimal) + "), period = 52))");
    connection.eval("fcast<-forecast(fun,h=" + Integer.toString(Head) + ");");
    connection.eval("fList <- as.data.frame(fcast)");
    int[] Ans = new int[Head];
    for (int i = 1; i <= Head; i++) {
        connection.eval("mean<-fList[[1]][" + Integer.toString(i) + "];");
        Ans[i - 1] = (int) connection.eval("mean").asDouble();
    }
    BufferedWriter OutputSetting = new BufferedWriter(new FileWriter("ForecastSetting\\" + Code + ".txt"));
    OutputSetting.write(pOptimal + "\t" + qOptimal + "\t" + spOptimal +"\t" + sqOptimal);
    OutputSetting.close();
    return Ans;
}
public int[] Forecast(int p, int q, int sp, int sq, int Head) {
    try {
        connection.voidEval("fun <- arima(data, order = c(" + Integer.toString(p) + ", 1, " + Integer.toString(q) + "),seasonal = list(order =
              c(" + Integer.toString(sp) + ", 0, " + Integer.toString(sq) + "), period = 52))");
        connection.eval("fcast<-forecast(fun,h=" + Integer.toString(Head) + ");");
        connection.eval("fList <- as.data.frame(fcast)");
        int[] Ans = new int[Head];
        for (int i = 1; i <= Head; i++) {
            connection.eval("mean<-fList[[1]][" + Integer.toString(i) + "];");
            Ans[i - 1] = (int) connection.eval("mean").asDouble();
        }
        return Ans;
    } catch (REXPMismatchException | RserveException ex) {
        return null;
    }
}
```

```
    public int[] EasyForecast(int Head) throws RserveException, REXPMismatchException {
        int pOptimal = 1;
        int qOptimal = 1;
        int spOptimal = 1;
        int sqOptimal = 1;
        connection.eval("fun <- arima(data, order = c(" + Integer.toString(pOptimal) + ", 1, " + Integer.toString(qOptimal) + "),seasonal =
             list(order = c(" + Integer.toString(spOptimal) + ", 0, " + Integer.toString(sqOptimal) + "), period = " + Integer.toString(Head) +
             "));");
        connection.eval("fcast<-forecast(fun,h=" + Integer.toString(Head) + ");");
        connection.eval("fList <- as.data.frame(fcast)");
        int[] Ans = new int[Head];
        for (int i = 1; i <= Head; i++) {
            connection.eval("mean<-fList[[1]][" + Integer.toString(i) + "];");
            Ans[i - 1] = (int) connection.eval("mean").asDouble();
        }
        return Ans;
    }
    void close() throws RserveException {
        connection.close();
    }
}
```

B.5 Math Model Code

```
package jbhuntmodel;
import ilog.concert.IloException;
import ilog.concert.IloIntVar;
import ilog.concert.IloLinearNumExpr;
import ilog.concert.IloNumVar;
import ilog.cplex.IloCplex;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.HashMap;
import java.util.List;
/**
 *
 * @author aghahari
 */
public class MathModel {
    IloNumVar[][] Y; // ID (Use MapID) -- Time
    IloNumVar[][][] X; // Ramp -- Ramp -- Time
    IloNumVar[][] I; // Customer(Ramp) -- Time
    IloNumVar[][] Z; // Customer(Ramp) -- Time
    HashMap<String, Integer> MapID;
    public MathModel() throws FileNotFoundException, IloException, IOException {
        IloCplex Model = new IloCplex();
        MapID = new HashMap<>();
        DataReader Data = new DataReader();
        // Initializing
        int Ramp = Data.getRampNumber();
        int Time = Data.getTime();
        int Customer = Data.getCustomerNumber();
        int All = Ramp + Customer;
```

```java
int Counter = 0;
for (int i = 0; i < Ramp; i++) {
    String[] ListCustomer = Data.getCustomerList(i);
    for (int j = 0; j < ListCustomer.length; j++) {
        String ID = Data.CreateID(Data.getRampName(i), ListCustomer[j]);
        MapID.put(ID, Counter++);
        ID = Data.CreateID(ListCustomer[j], Data.getRampName(i));
        MapID.put(ID, Counter++);
    }
    for (int j = 0; j < ListCustomer.length; j++) {
        for (int k = 0; k < ListCustomer.length; k++) {
            if (j != k) {
                String ID = Data.CreateID(ListCustomer[k], ListCustomer[j]);
                MapID.put(ID, Counter++);
            }
        }
    }
}
Counter = MapID.size();
System.out.println("Map Created");
X = new IloNumVar[Ramp][Ramp][Time];
for (int i = 0; i < Ramp; i++) {
    for (int j = 0; j < Ramp; j++) {
        for (int k = 0; k < Time; k++) {
            X[i][j][k] = Model.numVar(0, Data.getRailLimit(), "X_" + Data.getRampName(i) + "_" + Data.getRampName(j) + "_" +
                    Integer.toString(k));
        }
    }
}
System.out.println("X Done");
I = new IloNumVar[All][Time];
for (int i = 0; i < All; i++) {
    for (int j = 0; j < Time; j++) {
        I[i][j] = Model.numVar(0, Data.getCapacity(i), "I_" + Data.getName(i) + "_" + Integer.toString(j));
    }
}
System.out.println("I Done");
Z = new IloNumVar[Ramp][Time];
for (int i = 0; i < Ramp; i++) {
    for (int j = 0; j < Time; j++) {
        Z[i][j] = Model.numVar(0, Data.isPurchase(i) * Data.getNewContainterLimit(), "Z_" + Data.getRampName(i) + "_" +
                Integer.toString(j));
    }
}
System.out.println("Z Done");
Y = new IloNumVar[Counter][Time];
for (int i = 0; i < Counter; i++) {
    for (int j = 0; j < Time; j++) {
        Y[i][j] = Model.numVar(0, Integer.MAX_VALUE, "Y_" + this.getNameMapID(i) + "_" + Integer.toString(j));
        if (i % 1000 == 0) {
            System.out.println(i + "\t" + Counter);
        }
    }
}
System.out.println("Y Done");
// End Initializing.
```

```
// Objective Function
IloLinearNumExpr Obj = Model.linearNumExpr();
for (int i = 0; i < Ramp; i++) {
    for (int j = 0; j < Ramp; j++) {
        for (int k = 0; k < Time; k++) {
            Obj.addTerm(Data.getRailCost(i, j), X[i][j][k]);
        }
    }
}
for (int i = 0; i <= Counter; i++) {
    for (int j = 0; j < Time; j++) {
        Obj.addTerm(Data.getTruckCost(this.getNameMapID(i)), Y[i][j]);
    }
}
for (int i = 0; i < All; i++) {
    for (int j = 0; j < Time; j++) {
        Obj.addTerm(Data.getIdleCost(i), I[i][j]);
    }
}
for (int i = 0; i < Ramp; i++) {
    for (int t = 0; t < Time; t++) {
        Obj.addTerm(Data.getNewCost(), Z[i][t]);
    }
}
Model.minimize(Obj);
Obj = null;
System.out.println("Obj Done");
//End Obj
//Constraints
// Constraint RAMP FLOW BALANCE For Time not Zero
IloLinearNumExpr Constraint = Model.linearNumExpr();
for (int i = 0; i < Ramp; i++) {
    for (int j = 1; j < Time; j++) {
        String[] ListCustomer = Data.getCustomerList(i);
        for (int k = 0; k < Ramp; k++) {
            if (i != k) {
                Constraint.addTerm(1.0, X[k][i][j - 1]);
                Constraint.addTerm(-1.0, X[i][k][j]);
            }
        }
        for (int k = 0; k < ListCustomer.length; k++) {
            Constraint.addTerm(1.0, Y[MapID.get(Data.CreateID(ListCustomer[k], Data.getRampName(i)))][j - 1]);
            Constraint.addTerm(-1.0, Y[MapID.get(Data.CreateID(Data.getRampName(i), ListCustomer[k]))][j]);
        }
        Constraint.addTerm(1.0, I[Data.getRampID(i)][j - 1]);
        Constraint.addTerm(-1.0, I[Data.getRampID(i)][j]);
        Constraint.addTerm(1.0, Z[i][j]);
        Model.addEq(Obj, Data.getRampDemand(i, j) - Data.getRampSupply(i, j));
        Constraint.clear();
    }
}
System.out.println("Constraint 1 Done");
// Constraint Customer FLOW BALANCE
for (int i = 0; i < Customer; i++) {
    for (int t = 1; t < Time; t++) {
        String[] ListRamp = Data.getRampList(i);
```

```
            for (int k = 0; k < ListRamp.length; k++) {
                Constraint.addTerm(1.0, Y[MapID.get(Data.CreateID(ListRamp[k], Data.getCustomerName(i)))][t]);
                Constraint.addTerm(-1.0, Y[MapID.get(Data.CreateID(Data.getCustomerName(i), ListRamp[k]))][t]);
                String[] ListCustomer = Data.getCustomerList(Data.getRampNumber(ListRamp[k]));
                for (int c = 0; c < ListCustomer.length; c++) {
                    Constraint.addTerm(1.0, Y[MapID.get(Data.CreateID(ListCustomer[c], Data.getCustomerName(i)))][t]);
                    Constraint.addTerm(-1.0, Y[MapID.get(Data.CreateID(Data.getCustomerName(i), ListCustomer[c]))][t]);
                }
            }
            Constraint.addTerm(1.0, I[Data.IDConvertor(i)][t - 1]);
            Constraint.addTerm(-1.0, I[Data.IDConvertor(i)][t]);
            Model.addEq(Obj, Data.getCustomerDemand(i, t) - Data.getCustomerSupply(i, t));
            Constraint.clear();
        }
    }
    System.out.println("Constraint 2 Done");
    // New Containers
    for (int i = 0; i < Ramp; i++) {
        for (int t = 0; t < Time; t++) {
            Constraint.addTerm(1.0, Z[i][t]);
        }
    }
    Model.addLe(Constraint, Data.getNewContainterLimit());
}
private String getNameMapID(int i) { // Return the ID from a number in MapID
    for (String key : MapID.keySet()) {
        if (MapID.get(key) == i) {
            return key;
        }
    }
    return "";
}
}
```

B.6 Simulation

B.6.1 Container Class

```
package containersnapshot;
import java.util.ArrayList;
/**
 *
 * @author aghahari
 */
public class Container {
    static int getType(ContainerStatus containerStatus) {
        int Counter1 = 0;
        return BoxList.stream().filter((B) -> (B.Status == containerStatus)).map((_item) -> 1).reduce(Counter1, Integer::sum);
    }
    int ID;
    Ramp CurrentRampLocation;
    ContainerStatus Status;
```

```
    private static int Counter = 0;
    static ArrayList<Container> BoxList = new ArrayList<>();
    boolean isFree() {
        if (Status == ContainerStatus.IDLE) {
            return true;
        }
        return false;
    }
    void setTransit() {
        Status = ContainerStatus.TRANSIT;
    }
    void setFree() {
        Status = ContainerStatus.IDLE;
    }
    void setCustomerHolding() {
        Status = ContainerStatus.BUSY;
    }
    public enum ContainerStatus {
        TRANSIT, IDLE, BUSY
    }
    public Container() {
        ID = Counter++;
        Status = ContainerStatus.IDLE;
        BoxList.add(this);
    }
    static public int getNumberBox() {
        return BoxList.size();
    }
}
```

B.6.2 Move Class

```
package containersnapshot;
import java.util.Calendar;
import java.util.Comparator;
/**
 *
 * @author aghahari
 */
public final class Move implements Comparable<Move> {
    static int getLastCode() {
        return counter;
    }
    Container Box;
    Ramp OriginRamp;
    Ramp DestinationRamp;
    String StartDate;
    String EndDate;
    String ReleaseDate;
    String ActionDate;
    int _YearStartDate;
    int _MonthStartDate;
    int _DayStartDate;
```

```
int _YearEndDate;
int _MonthEndDate;
int _DayEndDate;
int _YearReleaseDate;
int _MonthReleaseDate;
int _DayReleaseDate;
int _YearActionDate;
int _MonthActionDate;
int _DayActionDate;
MoveTypes MoveType;
ActionType Action;
Boolean BadData;
String R = "";
private static int counter = 0;
ActionType getAction() {
   return Action;
}
void setToEnd() {
   Action = ActionType.END_MOVE;
   setActionDate(_YearEndDate, _MonthEndDate, _DayEndDate);
}
boolean isBadData() {
   return BadData;
}
String getStartTime() {
   return _YearStartDate + "-" + _MonthStartDate + "-" + _DayStartDate;
}
String getEndTime() {
   return _YearEndDate + "-" + _MonthEndDate + "-" + _DayEndDate;
}
void setBox() {
   Box = OriginRamp.getBox();
   OriginRamp.addBox(Box);
   Box.setTransit();
}
private String AddSeven() {
   Calendar cal = Calendar.getInstance();
   cal.set(_YearEndDate, _MonthEndDate - 1, _DayEndDate);
   cal.add(Calendar.DAY_OF_MONTH, 7);
   return cal.get(Calendar.YEAR) + "-" + Integer.toString(cal.get(Calendar.MONTH) + 1) + "-" + cal.get(Calendar.DAY_OF_MONTH);
}
String getReleaseTime() {
   return _YearReleaseDate + "-" + _MonthReleaseDate + "-" + _DayReleaseDate;
}
void releaseBox() {
   Box.setFree();
   this.Action = ActionType.REMOVE;
}
String getActionTime() {
   return _YearActionDate + "-" + _MonthActionDate + "-" + _DayActionDate;
}
public enum MoveTypes {
   LOAD, EMPTY, Holding;
}
public enum ActionType {
   INITIAL_MOVE,
```

```
    END_MOVE,
    ReleaseBox, REMOVE
}
public Move(Ramp Origin1, Ramp Origin2, String SDate, String EDate, String InGate, String OutGate, String Status, String ActionD, String
        OrigLine) {
    R = OrigLine;
    counter++;
    OriginRamp = Origin1;
    DestinationRamp = Origin2;
    BadData = false;
    _YearEndDate = Integer.parseInt(OutGate.split("-")[0]);
    _MonthEndDate = Integer.parseInt(OutGate.split("-")[1]);
    _DayEndDate = Integer.parseInt(OutGate.split("-")[2]);
    EndDate = OutGate.split("-")[0] + "-" + OutGate.split("-")[1] + "-" + OutGate.split("-")[2];
    int Year = Integer.parseInt(InGate.split("-")[0]);
    if (Year == 1) {
        if (!SDate.equals("")) {
            StartDate = SDate;
            _YearStartDate = Integer.parseInt(SDate.split("-")[0]);
            _MonthStartDate = Integer.parseInt(SDate.split("-")[1]);
            _DayStartDate = Integer.parseInt(SDate.split("-")[2]);
            if (!EDate.equals("")) {
                _YearEndDate = Integer.parseInt(EDate.split("-")[0]);
                _MonthEndDate = Integer.parseInt(EDate.split("-")[1]);
                _DayEndDate = Integer.parseInt(EDate.split("-")[2]);
            }
            Action = ActionType.INITIAL_MOVE;
            setActionDate(_YearStartDate, _MonthStartDate, _DayStartDate);
        } else {
            StartDate = "NOTIME";
            BadData = true;
            System.out.println("NOTIME\t" + R.split(",")[0] + "\t" + R + "\t" + StartDate + "\t" + EndDate);
            setBox();
            Action = ActionType.END_MOVE;
        }
    } else {
        _YearStartDate = Integer.parseInt(InGate.split("-")[0]);
        _MonthStartDate = Integer.parseInt(InGate.split("-")[1]);
        _DayStartDate = Integer.parseInt(InGate.split("-")[2]);
        StartDate = InGate.split("-")[0] + "-" + InGate.split("-")[1] + "-" + InGate.split("-")[2];
        Action = ActionType.INITIAL_MOVE;
        setActionDate(_YearStartDate, _MonthStartDate, _DayStartDate);
    }
    if (_YearStartDate > _YearEndDate) {
        _YearEndDate = _YearStartDate;
        _MonthEndDate = _MonthStartDate;
        _DayEndDate = _DayStartDate;
        System.out.println("BADDATA\t" + R.split(",")[0] + "\t" + R + "\t" + StartDate + "\t" + EndDate);
    } else if (_YearStartDate == _YearEndDate && _MonthStartDate > _MonthEndDate) {
        _YearEndDate = _YearStartDate;
        _MonthEndDate = _MonthStartDate;
        _DayEndDate = _DayStartDate;
        System.out.println("BADDATA\t" + R.split(",")[0] + "\t" + R + "\t" + StartDate + "\t" + EndDate);
    } else if (_YearStartDate == _YearEndDate && _MonthStartDate == _MonthEndDate && _DayStartDate > _DayEndDate) {
        _YearEndDate = _YearStartDate;
        _MonthEndDate = _MonthStartDate;
```

```
            _DayEndDate = _DayStartDate;
            System.out.println("BADDATA\t" + R.split(",")[0] + "\t" + R + "\t" + StartDate + "\t" + EndDate);
        }
        if (Action == ActionType.END_MOVE) {
            setActionDate(_YearEndDate, _MonthEndDate, _DayEndDate);
        }
        EndDate = _YearEndDate + "-" + _MonthEndDate + "-" + _DayEndDate;
        if (Math.abs(_YearStartDate - _YearEndDate) >= 1) {
            System.out.println("CheckDate\t" + R.split(",")[0] + "\t" + R + "\t" + StartDate + "\t" + EndDate);
        }
        if (Status.trim().toUpperCase().equals("L")) {
            MoveType = MoveTypes.LOAD;
            String R = AddSeven();
            _YearReleaseDate = Integer.parseInt(R.split("-")[0]);
            _MonthReleaseDate = Integer.parseInt(R.split("-")[1]);
            _DayReleaseDate = Integer.parseInt(R.split("-")[2]);
        } else {
            MoveType = MoveTypes.EMPTY;
        }
    }
    void setActionDate(int Year, int Month, int Day) {
        _YearActionDate = Year;
        _MonthActionDate = Month;
        _DayActionDate = Day;
    }
    void endMove() {
        OriginRamp.removeBox(Box);
        DestinationRamp.addBox(Box);
        if (MoveType == MoveTypes.LOAD) {
            Box.setCustomerHolding();
            Action = ActionType.ReleaseBox;
            setActionDate(_YearReleaseDate, _MonthReleaseDate, _DayReleaseDate);
        } else {
            releaseBox();
        }
    }
    @Override
    public int compareTo(Move o) {
        int thisYear = _YearActionDate;
        int oYear = o._YearActionDate;
        int thisMonth = _MonthActionDate;
        int oMonth = o._MonthActionDate;
        int oDay = o._DayActionDate;
        int thisDay = _DayActionDate;
        if (thisYear > oYear) {
            return 1;
        } else if (thisYear < oYear) {
            return -1;
        } else if (thisMonth > oMonth) {
            return 1;
        } else if (thisMonth < oMonth) {
            return -1;
        } else if (thisDay > oDay) {
            return 1;
        } else if (thisDay < oDay) {
            return -1;
```

```java
        } else {
            return 0;
        }
    }
    static class MoveComparator implements Comparator<Move> {
        @Override
        public int compare(Move o1, Move o2) {
            return o1.compareTo(o2);
        }
    }
}
```

B.6.3 Ramp Class

```java
package containersnapshot;
import java.util.ArrayList;
/**
 *
 * @author aghahari
 */
public class Ramp {
    private final String Name;
    private int Inventory;
    private final ArrayList<Container> ContainerList;
    public static ArrayList<Ramp> RampList = new ArrayList<>();
    public Ramp(String _Name) {
        ContainerList = new ArrayList<>();
        Name = _Name;
        Inventory = 0;
        RampList.add(this);
    }
    String getCode() {
        return Name;
    }
    Container getBox() {
        for (Container Box : ContainerList) {
            if (Box.isFree()) {
                return Box;
            }
        }
        return new Container();
    }
    void addBox(Container Box) {
        Box.CurrentRampLocation = this;
        ContainerList.add(Box);
        if (ContainerList.size()>Inventory){
            Inventory = ContainerList.size();
        }
    }
    void removeBox(Container Box) {
        Box.CurrentRampLocation = null;
        ContainerList.remove(Box);
    }
```

```java
    int getInventory() {
        return Inventory;
    }
    int getBoxNumber(Container.ContainerStatus StatBox) {
        int Counter=0;
        return ContainerList.stream().filter((B)-> (B.Status==StatBox)).map((_item)->1).reduce(Counter, Integer::sum);
    }
}
```

B.6.4 Control Class

```java
package containersnapshot;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.HashMap;
/**
 *
 * @author aghahari
 */
public class Control {
    ArrayList<Ramp> RampList = new ArrayList<>();
    String[] Files;
    BufferedReader Input;
    int FileCounter;
    ArrayList<Move> ActionList = new ArrayList<>();
    private int StartCounter;
    private int CompletedCounter;
    private int EndCounter;
    private int ReleaseCounter;
    private int NewCounter;
    private final boolean isFast = false;

    private Move FindMinTime() {
        Move MinMove = null;
        String MinDate = "2100-1-1";
        for (Move M : ActionList) {
            String Time = M.ActionDate;
            if (isSooner(MinDate, Time)) {
                MinDate = Time;
                MinMove = M;
            }
        }
        return MinMove;
    }
    private boolean isSooner(String MinDate, String Time) {
        if (Integer.parseInt(MinDate.split("-")[0]) > Integer.parseInt(Time.split("-")[0])) {
            return true;
```

```java
    } else if (Integer.parseInt(MinDate.split("-")[1]) > Integer.parseInt(Time.split("-")[1])) {
        return true;
    } else if (Integer.parseInt(MinDate.split("-")[2]) > Integer.parseInt(Time.split("-")[2])) {
        return true;
    }
    return false;
}
private void StartMoving(Move MinMove) {
    MinMove.setBox();
    MinMove.setToEnd();
}
private void ReadNewLine() throws IOException {
    String R = Input.readLine();
    if (R != null) {
        ActionList.add(new Move(getRamp(CleanSplit(R, Columns.OriginRamp)), getRamp(CleanSplit(R, Columns.DestinationRamp)), CleanSplit(R,
                Columns.PickupDate), CleanSplit(R, Columns.DeliveryDate), CleanSplit(R, Columns.InGate), CleanSplit(R, Columns.OutGate),
                CleanSplit(R, Columns.LoadFlag), CleanSplit(R, Columns.ActionDate), R));
    } else if (FileCounter < Files.length) {
        InitializeNewFile();
        R = Input.readLine();
        ActionList.add(new Move(getRamp(CleanSplit(R, Columns.OriginRamp)), getRamp(CleanSplit(R, Columns.DestinationRamp)), CleanSplit(R,
                Columns.PickupDate), CleanSplit(R, Columns.DeliveryDate), CleanSplit(R, Columns.InGate), CleanSplit(R, Columns.OutGate),
                CleanSplit(R, Columns.LoadFlag), CleanSplit(R, Columns.ActionDate), R));
    }
}
private void EndMoving(Move MinMove) {
    MinMove.endMove();
}
private void ReleaseBox(Move MinMove) {
    MinMove.releaseBox();
}
HashMap<String, Integer> Printed = new HashMap<>();

private void ReportCreator(String Display) throws IOException {
    BufferedWriter Output = null;
    if (Printed.getOrDefault(Display, -1) != -1) {
        Output = new BufferedWriter(new FileWriter(System.getProperty("user.dir") + "\\BigOne\\" + Display + "-SUB_" + Printed.get(Display) +
                ".txt"));
        Printed.put(Display, Printed.get(Display) + 1);
    } else {
        Output = new BufferedWriter(new FileWriter(System.getProperty("user.dir") + "\\BigOne\\" + Display + ".txt"));
        Printed.put(Display, 1);
    }
    Output.write("\nNew Moves \t" + NewCounter + "\n");
    Output.write("Start Moves \t" + StartCounter + "\n");
    Output.write("Arrived Moves \t" + EndCounter + "\n");
    Output.write("Completed Moves \t" + CompletedCounter + "\n");
    Output.write("Available Container\t" + Container.getNumberBox() + "\n");
    Output.write("\tOn Rail\t" + Container.getType(Container.ContainerStatus.TRANSIT) + "\n");
    Output.write("\tFree\t" + Container.getType(Container.ContainerStatus.IDLE) + "\n");
    Output.write("\tBusy\t" + Container.getType(Container.ContainerStatus.BUSY) + "\n");
    Output.write("Name\tInventory\tIDLE\tBUSY\n");
    for (Ramp R : Ramp.RampList) {
        Output.write(R.getCode() + "\t" + R.getInventory() + "\t" + R.getBoxNumber(Container.ContainerStatus.IDLE) + "\t" +
                R.getBoxNumber(Container.ContainerStatus.BUSY) + "\n");
    }
```

```java
    Output.write("\nOrigin\tDestination\tType\tCount\n");
    HashMap<String, Integer> RailMove = new HashMap<>();
    HashMap<String, Integer> RailMoveDetail = new HashMap<>();
    HashMap<String, Integer> BoxDetail = new HashMap<>();
    for (Move M : ActionList) {
        if (M.getAction() == Move.ActionType.END_MOVE) {
            String Key = M.OriginRamp.getCode() + "\t" + M.DestinationRamp.getCode() + "\t" + M.MoveType;
            String Key1 = M.OriginRamp.getCode() + "\t" + M.DestinationRamp.getCode() + "\t" + M.MoveType + "\t" + M.getEndTime();
            RailMove.put(Key, RailMove.getOrDefault(Key, 0) + 1);
            RailMoveDetail.put(Key1, RailMove.getOrDefault(Key1, 0) + 1);
        } else if (M.getAction() == Move.ActionType.ReleaseBox) {
            String Key = M.DestinationRamp.getCode() + "\t" + M.getReleaseTime();
            BoxDetail.put(Key, BoxDetail.getOrDefault(Key, 0) + 1);
        }
    }
    for (String Key : RailMove.keySet()) {
        Output.write(Key + "\t" + RailMove.get(Key) + "\n");
    }
    Output.write("\nOrigin\tDestination\tType\tTime\tCount\n");
    for (String Key : RailMoveDetail.keySet()) {
        Output.write(Key + "\t" + RailMoveDetail.get(Key) + "\n");
    }
    Output.write("\nRamp\tReleaseDate\tBoxNumber\n");
    for (String Key : BoxDetail.keySet()) {
        Output.write(Key + "\t" + BoxDetail.get(Key) + "\n");
    }
    Output.write("\nLast Read Line\t" + Move.getLastCode());
    Output.close();
}
private void FastForward(String string) throws IOException {
    String R = Input.readLine();
    int Count = 0;
    BufferedWriter Output = new BufferedWriter(new FileWriter("Sorted.txt"));
    while (1 == 1) {
        Count++;
        Output.write(R + "\n");
        if (diff(CleanSplit(R, Columns.ActionDate), string) == -1) {
            ActionList.add(new Move(getRamp(CleanSplit(R, Columns.OriginRamp)), getRamp(CleanSplit(R, Columns.DestinationRamp)), CleanSplit(R,
                    Columns.PickupDate), CleanSplit(R, Columns.DeliveryDate), CleanSplit(R, Columns.InGate), CleanSplit(R, Columns.OutGate),
                    CleanSplit(R, Columns.LoadFlag), CleanSplit(R, Columns.ActionDate), R));
            break;
        }
        R = Input.readLine();
    }
    for (int i = 0; i < 10; i++) {
        Output.write(Input.readLine() + "\n");
    }
    Output.close();
}
private enum Columns {
    OrderNumber,
    ShipperCode,
    ShipperLat,
    ShipperLong,
    ReceiverCode,
    ReceiverLat,
```

```
        ReceiverLong,
        PickupDate,
        PickupTime,
        DeliveryDate,
        DeliveryTime,
        OriginRamp,
        DestinationRamp,
        TotalRevenue,
        InGate,
        OutGate,
        LoadFlag,
        ActionDate
    }
    public Control(String[] FileNames) throws FileNotFoundException, IOException {
        Files = new String[FileNames.length];
        System.arraycopy(FileNames, 0, Files, 0, Files.length);
        FileCounter = 0;
        InitializeNewFile();
        String R = Input.readLine();
        if (isFast) {
            FastForward("2010-01-01");
        } else {
            ActionList.add(new Move(getRamp(CleanSplit(R, Columns.OriginRamp)), getRamp(CleanSplit(R, Columns.DestinationRamp)), CleanSplit(R,
                    Columns.PickupDate), CleanSplit(R, Columns.DeliveryDate), CleanSplit(R, Columns.InGate), CleanSplit(R, Columns.OutGate),
                    CleanSplit(R, Columns.LoadFlag), CleanSplit(R, Columns.ActionDate), R));
        }
    }
    private void InitializeNewFile() throws FileNotFoundException, IOException {
        System.out.println("Starting File\t" + Files[FileCounter]);
        Input = new BufferedReader(new FileReader(Files[FileCounter]));
        Input.readLine();
        FileCounter++;
    }
    int Year = 0;
    int Month = 0;
    int Day = 0;
    public void Run() throws FileNotFoundException, IOException {
        String Display = "";
        StartCounter = 0;
        EndCounter = 0;
        ReleaseCounter = 0;
        NewCounter = 0;
        CompletedCounter = 0;
        while (!ActionList.isEmpty()) {
            ActionList.sort(new Move.MoveComparator());
            Move MinMove = ActionList.get(0);
            if (Display.equals("")) {
                Display = MinMove.getActionTime();
                Year = MinMove._YearActionDate;
                Month = MinMove._MonthActionDate;
                Day = MinMove._DayActionDate;
            } else if ((Year != MinMove._YearActionDate) || (Month != MinMove._MonthActionDate) || (Day != MinMove._DayActionDate)) {
                ReportCreator(Display);
                StartCounter = 0;
                EndCounter = 0;
                ReleaseCounter = 0;
```

```
            NewCounter = 0;
            CompletedCounter = 0;
            Display = MinMove.getActionTime();
            Year = MinMove._YearActionDate;
            Month = MinMove._MonthActionDate;
            Day = MinMove._DayActionDate;
            Calendar T = Calendar.getInstance();
            T = null;
        }
        switch (MinMove.Action) {
            case INITIAL_MOVE:
                StartCounter++;
                StartMoving(MinMove);
                NewCounter++;
                ReadNewLine();
                break;
            case END_MOVE:
                EndCounter++;
                EndMoving(MinMove);
                if (MinMove.isBadData()) {
                    ReadNewLine();
                    NewCounter++;
                }
                break;
            case ReleaseBox:
                ReleaseCounter++;
                ReleaseBox(MinMove);
                break;
        }
        if (MinMove.Action == Move.ActionType.REMOVE) {
            ActionList.remove(MinMove);
            CompletedCounter++;
        }
    }
    ReportCreator(Display);
}
private Ramp getRamp(String Code) {
    for (Ramp R : RampList) {
        if (R.getCode().equals(Code)) {
            return R;
        }
    }
    Ramp R = new Ramp(Code);
    RampList.add(R);
    return R;
}
private static String CleanSplit(String R, Columns Index) {
    return R.split(",")[Index.ordinal()].replace("\"", "").trim().toUpperCase();
}
private static int diff(String oDate, String thisDate) {
    int thisYear = Integer.parseInt(thisDate.split("-")[0]);
    int oYear = Integer.parseInt(oDate.split("-")[0]);
    int thisMonth = Integer.parseInt(thisDate.split("-")[1]);
    int oMonth = Integer.parseInt(oDate.split("-")[1]);
    int oDay = Integer.parseInt(oDate.split("-")[2]);
    int thisDay = Integer.parseInt(thisDate.split("-")[2]);
```

```
        if (thisYear > oYear) {
            return 1;
        } else if (thisYear < oYear) {
            return -1;
        } else if (thisMonth > oMonth) {
            return 1;
        } else if (thisMonth < oMonth) {
            return -1;
        } else if (thisDay > oDay) {
            return 1;
        } else if (thisDay < oDay) {
            return -1;
        } else {
            return 0;
        }
    }
}
```

C Chapter 4 Programming Codes

C.1 Care Center Class

```
package chapter3simulation;
import java.util.ArrayList;
import java.util.LinkedList;
/**
 *
 * @author Amir
 */
public abstract class CareCenter {

    public double X;
    public double Y;
    protected Simulator myParent;
    protected final LinkedList<Patient> myQueue;
    protected final LinkedList<Patient> myDeviceQueue;
    protected ArrayList<TBDevice> myDevices;
    protected Region myRegion;
    public Simulator getParent() {
        return myParent;
    }
    public CareCenter(Simulator myParent, Region myRegion) {
        this.X = RandomCreator.getX(myRegion);
        this.Y = RandomCreator.getY(myRegion);
        this.myRegion = myRegion;
        this.myParent = myParent;
        myQueue = new LinkedList<>();
        myDeviceQueue = new LinkedList<>();
        myDevices = new ArrayList<>();
    }
    public double getX() {
```

```
        return X;
    }
    public double getY() {
        return Y;
    }
    public Region getRegion() {
        return this.myRegion;
    }
    protected TBDevice getFreeDevice() {
        for (TBDevice myDevice : this.myDevices) {
            if (myDevice.isFree()) {
                return myDevice;
            }
        }
        return null;
    }
    protected boolean isDeviceFree() {
        for (TBDevice myDevice : this.myDevices) {
            if (myDevice.isFree()) {
                return true;
            }
        }
        return false;
    }
    abstract void receivePatient(Object myChild);
    abstract void patientLeave(Object myChild);
    abstract String getName();
    abstract void getNextPatient();
}
```

C.2 Clinic Class

```
package chapter3simulation;
import java.util.LinkedList;
/**
 *
 * @author Amir
 */
public class Clinic extends CareCenter {
    private static int myIDCounter = 0;
    private int ID;
    private final boolean Traditional;
    private int myDoctors;
    private int InPatient;
    private int OutPatient;
    private double LastQUpdate;
    private double LastDocUpdate;
    private double TotalQTime;
    private double TotalBusyPOC;
    private LinkedList<Patient> SampleToSend;
    private final double sendSampleInterval = 2 * 60;
    private LinkedList<Patient> SentSample;
    private int BusyAsset;
```

```
private TBCenter myCenter;
public Clinic(Simulator myParent, boolean traditional, int numDoc, int numDevice, Region myRegion) {
    super(myParent, myRegion);
    this.ID = myIDCounter++;
    this.Traditional = traditional;
    this.myDoctors = numDoc;
    if (!this.Traditional) {
        for (int i = 0; i < numDevice; i++) {
            myDevices.add(new TBDevice(this));
        }
    }
    this.SampleToSend = new LinkedList<>();
    this.SentSample = new LinkedList<>();
    myCenter = this.myParent.getTBCenter(this);
}
@Override
void receivePatient(Object myChild) {
    Patient myPatient = (Patient) myChild;
    InPatient++;
    myPatient.setMyCareCenter(this);
    myQueue.add(myPatient);
    UpdateQStat(true);
    myPatient.InQuTimeStamp();
    this.processNextPatient();
}
private void UpdateQStat(boolean add) {
    TotalQTime += (myQueue.size() + ((add) ? -1 : +1)) * (this.myParent.getTime() - LastQUpdate);
    LastQUpdate = this.myParent.getTime();
}
private void UpdatePOC(boolean add) {
    TotalBusyPOC += (this.BusyAsset + ((add) ? -1 : +1)) * (this.myParent.getTime() - LastDocUpdate);
    LastDocUpdate = this.myParent.getTime();
}
@Override
void patientLeave(Object myChild) {
    Patient myPatient = (Patient) myChild;
    OutPatient++;
    if (this.Traditional) {
        this.SampleToSend.add(myPatient);
        if (this.SampleToSend.size() == 1) {
            this.myParent.addToFEL(new Event((((int) (myParent.getTime() / (60 * 24))) + 1) * 60 * 24,
                    EventNames.EventName.SendSampleToTBCenter, this));
        }
    } else {
        this.processByDevice(myPatient);
    }
    this.reduceAsset();
}
private void reduceAsset() {
    BusyAsset--;
    this.UpdatePOC(false);
    this.processNextPatient();
}
private void processNextPatient() {
    if (this.myQueue.size() > 0) {
        Patient myPatient = this.myQueue.getFirst();
```

```java
            if (BusyAsset < this.myDoctors) {
                BusyAsset++;
                myQueue.remove(myPatient);
                this.UpdateQStat(false);
                this.UpdatePOC(true);
                myPatient.OutQuTimeStamp();
                this.myParent.addToFEL(new Event(myParent.getTime() + RandomCreator.getTraditionalDRVisit(), EventNames.EventName.EndOfDrVisit,
                        myPatient));
            }
        }
    }
    @Override
    public String toString() {
        TotalQTime += myQueue.size() * (this.myParent.getTime() - LastQUpdate);
        TotalBusyPOC += BusyAsset * (this.myParent.getTime() - LastDocUpdate);
        return "Clinic_" + ID + "\t" + this.InPatient + "\t" + this.OutPatient + "\t" + this.TotalQTime / this.myParent.getTime() + "\t" +
                this.TotalBusyPOC / this.myParent.getTime() + "\t" + this.SampleToSend.size() + "\t" + this.SentSample.size() + "\t" +
                this.myQueue.size();
    }
    public void sendSamples() {
        this.myParent.addToFEL(new Event(myParent.getTime() + RandomCreator.getTravelTime(this, this.getCenter()),
                EventNames.EventName.TBCenterReceived, this));
        SentSample = (LinkedList<Patient>) SampleToSend.clone();
        SampleToSend = new LinkedList<>();
    }
    public TBCenter getCenter() {
        return myCenter;
    }
    public LinkedList getSample() {
        return SentSample;
    }
    public String getName() {
        return "Clinic_" + ID;
    }
    private void processByDevice(Patient myPatient) {
        if (isDeviceFree()) {
            TBDevice myDevice = this.getFreeDevice();
            myDevice.getPatient(myPatient);
        } else {
            this.myDeviceQueue.add(myPatient);
        }
    }
    @Override
    void getNextPatient() {
        if (this.myDeviceQueue.size() > 0) {
            Patient myPatient = this.myDeviceQueue.getFirst();
            processByDevice(myPatient);
            this.myDeviceQueue.remove(myPatient);
        }
    }
}
```

C.3 Microscopy Center Class

```
package chapter3simulation;
import java.util.LinkedList;
/**
 *
 * @author Amir
 */
public class TBCenter extends CareCenter {
    private static int myIDCounter = 0;
    private int ID;
    private final int Capacity;
    private int BusyAsset;
    private Hospital myCenter;
    private LinkedList<Patient> SampleToSend;
    private final double sendSampleInterval = 4 * 60;
    private LinkedList<Patient> SentSample;
    private final boolean Traditional;
    private int OutPatient;
    private double LastQUpdate;
    private double LastDocUpdate;
    private double TotalQTime;
    private double TotalBusyPOC;
    private int InPatient;
    public TBCenter(Simulator myParent, boolean Tradit, int _Capacity, Region myRegion) {
        super(myParent, myRegion);
        this.ID = myIDCounter++;
        this.Capacity = _Capacity;
        this.SampleToSend = new LinkedList<>();
        this.SentSample = new LinkedList<>();
        this.Traditional = Tradit;
        if (!this.Traditional) {
            for (int i = 0; i < _Capacity; i++) {
                this.myDevices.add(new TBDevice(this));
            }
        }
        myCenter = this.myParent.getHospital(this);
    }
    @Override
    void receivePatient(Object myChild) {

        Clinic myClinic = (Clinic) myChild;
        LinkedList mySamples = myClinic.getSample();
        if (!mySamples.isEmpty()) {
            while (!mySamples.isEmpty()) {
                InPatient++;
                Patient myPatient = (Patient) mySamples.getFirst();
                this.myQueue.add(myPatient);
                this.UpdateQStat(true);
                mySamples.remove(myPatient);
                myPatient.TBCenterArriveTimeStamp();
            }
        }
        mySamples = new LinkedList<>();
        this.processNextPatient();
```

```
}
@Override
void patientLeave(Object myChild) {
    Patient myPatient = (Patient) myChild;
    myPatient.setTBCenterResult(RandomCreator.getTBCenterResult());
    OutPatient++;
    if (this.Traditional) {
        this.reduceAsset();
        if (!myPatient.getTBCenterResult()) {
            this.SampleToSend.add(myPatient);
            if(this.SampleToSend.size()==1){
                this.myParent.addToFEL(new Event((((int) (myParent.getTime() / (60 * 24))) + 3) * 60 * 24,
                        EventNames.EventName.SendSampleToHospital, this));
            }
        } else {
            myPatient.notifyOfResult();
        }
    } else {
        reduceAsset();
    }
}
public void sendSamples() {
    this.myParent.addToFEL(new Event(myParent.getTime() + RandomCreator.getTravelTime(this, this.getCenter()),
            EventNames.EventName.HospitalReceive, this));
    SentSample = (LinkedList<Patient>) SampleToSend.clone();
    SampleToSend = new LinkedList<>();
}
public Hospital getCenter() {
    return myCenter;
}
private void reduceAsset() {
    BusyAsset--;
    this.UpdatePOC(false);
    this.processNextPatient();
}
private void processNextPatient() {
    while (this.BusyAsset <= this.Capacity && !myQueue.isEmpty()) {
        this.BusyAsset++;
        this.UpdatePOC(true);
        Patient myPatient = (Patient) myQueue.getFirst();
        if (Traditional) {
            this.myParent.addToFEL(new Event(this.myParent.getTime() + RandomCreator.getTBCenterProcessingTime(),
                    EventNames.EventName.EndOfProcessingTBCenter, myPatient));
        } else {
            this.processByDevice(myPatient);
        }
        myQueue.remove(myPatient);
        this.UpdateQStat(false);
        myPatient.TBCenterProcessTimeStamp();
    }
}
private void UpdateQStat(boolean add) {
    TotalQTime += (myQueue.size() + ((add) ? -1 : +1)) * (this.myParent.getTime() - LastQUpdate);
    LastQUpdate = this.myParent.getTime();
}
private void UpdatePOC(boolean add) {
```

```
        TotalBusyPOC += (this.BusyAsset + ((add) ? -1 : +1)) * (this.myParent.getTime() - LastDocUpdate);
        LastDocUpdate = this.myParent.getTime();
    }
    @Override
    public String toString() {
        TotalQTime += myQueue.size() * (this.myParent.getTime() - LastQUpdate);
        TotalBusyPOC += BusyAsset * (this.myParent.getTime() - LastDocUpdate);
        return "TBCenter_" + ID + "\t" + this.InPatient + "\t" + this.OutPatient + "\t" + this.TotalQTime / this.myParent.getTime() + "\t" +
                this.TotalBusyPOC / this.myParent.getTime() + "\t" + this.SampleToSend.size() + "\t" + this.SentSample.size() + "\t" +
                this.myQueue.size();
    }
    LinkedList getSample() {
        return this.SentSample;
    }
    public String getName() {
        return "TBCenter_" + ID;
    }
    private void processByDevice(Patient myPatient) {
        if (isDeviceFree()) {
            TBDevice myDevice = this.getFreeDevice();
            myDevice.getPatient(myPatient);
        }
    }
    @Override
    void getNextPatient() {
        this.patientLeave(this);
    }
}
```

C.4 Major Hospital Class

```
package chapter3simulation;
import java.util.LinkedList;
/**
 *
 * @author Amir
 */
public class Hospital extends CareCenter {
    private static int myIDCounter = 0;
    private int ID;
    private final int Capacity;
    private int BusyAsset;
    private int OutPatient;
    private double LastQUpdate;
    private double LastDocUpdate;
    private double TotalQTime;
    private double TotalBusyPOC;
    private int InPatient;
    public Hospital(Simulator myParent, int _Capacity, Region myRegion) {
        super(myParent, myRegion);
        this.ID = myIDCounter++;
        this.Capacity = _Capacity;
    }
```

```
@Override
void receivePatient(Object myChild) {
    TBCenter myTBCenter = (TBCenter) myChild;
    LinkedList mySamples = myTBCenter.getSample();
    if (!mySamples.isEmpty()) {
        while (!mySamples.isEmpty()) {
            InPatient++;
            Patient myPatient = (Patient) mySamples.getFirst();
            this.myQueue.add(myPatient);
            this.UpdateQStat(true);
            mySamples.remove(myPatient);
            myPatient.HospitalArrivalTimeStamp();
        }
    }
    mySamples = new LinkedList<>();
    this.processNextPatient();
}
@Override
void patientLeave(Object myChild) {
    Patient myPatient = (Patient) myChild;
    myPatient.setHospitalResult(RandomCreator.getHospitalResult());
    OutPatient++;
    this.reduceAsset();
    myPatient.notifyOfResult();
}
private void reduceAsset() {
    BusyAsset--;
    this.UpdatePOC(false);
    this.processNextPatient();
}
private void processNextPatient() {
    while (this.BusyAsset <= this.Capacity && !myQueue.isEmpty()) {
        this.BusyAsset++;
        this.UpdatePOC(true);
        Patient myPatient = (Patient) myQueue.getFirst();
        this.myParent.addToFEL(new Event(this.myParent.getTime() + RandomCreator.getHospitalProcessingTime(),
               EventNames.EventName.EndOfProcessingHospital, myPatient));
        myQueue.remove(myPatient);
        this.UpdateQStat(false);
        myPatient.HospitalProcessingTimeStamp();
    }
}
private void UpdateQStat(boolean add) {
    TotalQTime += (myQueue.size() + ((add) ? -1 : +1)) * (this.myParent.getTime() - LastQUpdate);
    LastQUpdate = this.myParent.getTime();
}
private void UpdatePOC(boolean add) {
    TotalBusyPOC += (this.BusyAsset + ((add) ? -1 : +1)) * (this.myParent.getTime() - LastDocUpdate);
    LastDocUpdate = this.myParent.getTime();
}
@Override
public String toString() {
    TotalQTime += myQueue.size() * (this.myParent.getTime() - LastQUpdate);
    TotalBusyPOC += BusyAsset * (this.myParent.getTime() - LastDocUpdate);
    return "Hospital_" + ID + "\t" + this.InPatient + "\t" + this.OutPatient + "\t" + this.TotalQTime / this.myParent.getTime() + "\t" +
          this.TotalBusyPOC / this.myParent.getTime() + "\t" + this.myQueue.size();
```

```
    }
    public String getName() {
        return "Hospital_" + ID;
    }
}
```

C.5 Patient Class

```
package chapter3simulation;
/**
 *
 * @author Amir
 */
public class Patient {
    private static int myIDCounter;
    private int ID;
    private final double X;
    private final double Y;
    private final Region myRegion;
    private final double startTime;
    private final Simulator myParent;
    private Clinic myCareCenter;
    private double InQ;
    private double OutQ;
    private double endTime;
    private boolean HospitalResult;
    private double TBCenterArriveTimeStamp;
    private double TBCenterProcessTimeStamp;
    private double HospitalProcessingTimeStamp;
    private double HospitalArrivalTimeStamp;
    private boolean TBCenterResult;
    public Clinic getCenter() {
        return (Clinic) myCareCenter;
    }
    public void setMyCareCenter(Clinic myCareCenter) {
        this.myCareCenter = myCareCenter;
    }
    public double getX() {
        return X;
    }
    public double getY() {
        return Y;
    }
    public Patient(Simulator myParent, Region region, double currentTime) {
        this.X = RandomCreator.getX(region);
        this.Y = RandomCreator.getY(region);
        this.ID = Patient.myIDCounter++;
        //System.out.println("Patient_" + ID + "\t" + this.X + "\t" + this.Y);
        this.myRegion = region;
        this.startTime = currentTime + RandomCreator.getNextPatientTime(region);
        this.myParent = myParent;
        if (myParent.TBOnDemand) {
            this.myParent.addToFEL(new Event(this.startTime, EventNames.EventName.PatientNotification, this));
```

```java
    } else {
        this.myParent.addToFEL(new Event(this.startTime, EventNames.EventName.PatientArrival, this));
    }
    this.myParent.addToAllList(this);
}
public void patientArrive() {
    this.myParent.getCareCenter(this).receivePatient(this);
    new Patient(this.myParent, this.myRegion, this.startTime);
}
void OutQuTimeStamp() {
    this.OutQ = this.myParent.getTime();
}
void InQuTimeStamp() {
    this.InQ = this.myParent.getTime();
}
void leaveCareCenter() {
    this.myCareCenter.patientLeave(this);
}
private String myString(double myTime) {
    return ((myTime == 0) ? "N/A" : Double.toString(myTime));
}
boolean isDone() {
    return (this.endTime != 0);
}
double getOutQ() {
    return this.OutQ;
}
double getInQ() {
    return this.InQ;
}
double getOut() {
    return this.endTime;
}
double arrived() {
    return this.startTime;
}
void TBCenterArriveTimeStamp() {
    this.TBCenterArriveTimeStamp = this.myParent.getTime();
}
void TBCenterProcessTimeStamp() {
    this.TBCenterProcessTimeStamp = this.myParent.getTime();
}
void setTBCenterResult(boolean result) {
    this.TBCenterResult = result;
}
void setHospitalResult(boolean result) {
    this.HospitalResult = result;
}
public boolean getTBCenterResult() {
    return this.TBCenterResult;
}
void notifyOfResult() {
    this.endTime = this.myParent.getTime();
}
void HospitalProcessingTimeStamp() {
    this.HospitalProcessingTimeStamp = this.myParent.getTime();
```

```
    }
    void HospitalArrivalTimeStamp() {
        this.HospitalArrivalTimeStamp = this.myParent.getTime();
    }
    @Override
    public String toString() {
        return "Patient_" + Integer.toString(ID) + "\t" + this.myCareCenter.getName() + "\t" + this.myCareCenter.getCenter().getName() + "\t" +
               this.myCareCenter.getCenter().getCenter().getName()
                + "\t" + this.InQ + "\t" + myString(OutQ) + "\t" + myString(TBCenterArriveTimeStamp) + "\t" + myString(TBCenterProcessTimeStamp) +
                       "\t" + myString(HospitalArrivalTimeStamp)
                + "\t" + myString(HospitalProcessingTimeStamp) + "\t" + myString(this.endTime) + "\t" + this.TBCenterResult + "\t" +
                       this.HospitalResult;
    }
    Region getRegion() {
        return this.myRegion;
    }
    void requestAVehicle() {
        this.myParent.receiveAPatient(this);
        new Patient(this.myParent, this.myRegion, this.myParent.CurrentTime);
    }
}
```

C.6 Vehicle Class

```
package chapter3simulation;
/**
 *
 * @author Amir
 */
public class Vehicle extends CareCenter {
    private static int IDCounter;
    private final int ID;
    public boolean isFree;
    Patient myCurrentPatient;
    public Vehicle(Simulator myParent, Region myRegion, int NumDevice) {
        super(myParent, myRegion);
        isFree = true;
        myCurrentPatient = null;
        this.ID = IDCounter++;
        for (int i = 0; i < NumDevice; i++) {
            myDevices.add(new TBDevice(this));
        }
        this.X = myRegion.getLowerX();
        this.Y = myRegion.getLowerY();
    }
    @Override
    String getName() {
        return "Vehicle_" + ID;
    }
    private void processByDevice(Patient myPatient) {
        if (isDeviceFree()) {
            TBDevice myDevice = this.getFreeDevice();
            myDevice.getPatient(myPatient);
```

```
    } else {
        this.myDeviceQueue.add(myPatient);
    }
}
double startTravel = 0;
double TotalTravel = 0;
double startWait = 0;
double TotalWait = 0;
double startFree = 0;
double TotalFree = 0;
void notifyOfNewPatient(Patient myPatient) {
    if (this.isFree) {
        startTravel = this.myParent.CurrentTime;
        this.myCurrentPatient = myPatient;
        TotalFree += this.myParent.CurrentTime - startFree;
        this.isFree = false;
        this.myParent.addToFEL(new Event(this.myParent.CurrentTime + RandomCreator.getTravelTime(myPatient, this),
               EventNames.EventName.VehicleArrive, this));
    } else {
        this.myQueue.add(myPatient);
    }
}
void ArriveAtPatientLocation() {
    TotalTravel += this.myParent.CurrentTime - this.startTravel;
    if (isDeviceFree()) {
        TBDevice myDevice = this.getFreeDevice();
        myDevice.getPatient(myCurrentPatient);
        this.isFree = true;
        startFree = this.myParent.CurrentTime;
        if (this.myQueue.size() > 0) {
            if (this.isDeviceFree()) {
                this.startTravel = this.myParent.CurrentTime;
                Patient myPatient = this.myQueue.get(0);
                this.myCurrentPatient = myPatient;
                TotalFree += this.myParent.CurrentTime - startFree;
                this.isFree = false;
                this.myParent.addToFEL(new Event(this.myParent.CurrentTime + RandomCreator.getTravelTime(myPatient, this),
                       EventNames.EventName.VehicleArrive, this));
                this.myQueue.remove(myPatient);
            } else {
                double leftTime = 90 - this.earliestDeviceTime() + this.myParent.CurrentTime;
                double minDistance = Double.MAX_VALUE;
                Patient myP = null;
                for (Patient myPatient : this.myQueue) {
                    double myDistance = RandomCreator.getTravelTime(myPatient, this);
                    if (myDistance > leftTime) {
                        if (myDistance < minDistance) {
                            minDistance = myDistance;
                            myP = myPatient;
                        }
                    }
                }
                if (myP == null) {
                    for (Patient myPatient : this.myQueue) {
                        double myDistance = RandomCreator.getTravelTime(myPatient, this);
                        if (myDistance < minDistance) {
```

```
                    minDistance = myDistance;
                    myP = myPatient;
                }
            }
        }
        this.startTravel = this.myParent.CurrentTime;
        this.myCurrentPatient = myP;
        TotalFree += this.myParent.CurrentTime - startFree;
        this.isFree = false;
        this.myParent.addToFEL(new Event(this.myParent.CurrentTime + RandomCreator.getTravelTime(myP, this),
              EventNames.EventName.VehicleArrive, this));
        this.myQueue.remove(myP);
      }
    }
  } else {
    this.startWait = this.myParent.CurrentTime;
    this.myDeviceQueue.add(myCurrentPatient);
  }
}
void notifyFreeDevice() {
  if (this.myDeviceQueue.size() > 0) {
    TBDevice myDevice = this.getFreeDevice();
    myDevice.getPatient(myCurrentPatient);
    this.myDeviceQueue.remove(myCurrentPatient);
    this.isFree = true;
    startFree = this.myParent.CurrentTime;
    this.TotalWait += this.myParent.CurrentTime - this.startWait;
    if (this.myQueue.size() > 0) {
      Patient myPatient = this.myQueue.get(0);
      this.myCurrentPatient = myPatient;
      this.startTravel = this.myParent.CurrentTime;
      TotalFree += this.myParent.CurrentTime - startFree;
      this.isFree = false;
      this.myParent.addToFEL(new Event(this.myParent.CurrentTime + RandomCreator.getTravelTime(myPatient, this),
            EventNames.EventName.VehicleArrive, this));
      this.myQueue.remove(myPatient);
    }
  }
}
private double earliestDeviceTime() {
  double minTime = Double.MAX_VALUE;
  for (TBDevice myDevice : this.myDevices) {
    if (minTime > myDevice.startTime) {
      minTime = myDevice.startTime;
    }
  }
  return minTime;
}
}
```

C.7 Simulator Class

```
package chapter3simulation;
```

```java
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
/**
 *
 * @author Amir
 */
public class Simulator {
    public FEL myFEL;
    public double SimulationRun;
    public ArrayList<Clinic> myClinics;
    public ArrayList<TBCenter> myTBCenters;
    public ArrayList<Hospital> myHospitals;
    public double CurrentTime;
    public ArrayList<Patient> AllPatients;
    public ArrayList<Region> myRegions;
    public int WarmUp = 1 * 20 * 7 * 24 * 60;
    public boolean TBOnDemand = true;
    public ArrayList<Vehicle> myVehicles = new ArrayList<>();
    private boolean DistrictBase = false;
    private double KPI;
    public Simulator() throws IOException {
        Event.myFile = new BufferedWriter(new FileWriter("Output.txt"));
        AllPatients = new ArrayList<>();
        myClinics = new ArrayList<>();
        myTBCenters = new ArrayList<>();
        myHospitals = new ArrayList<>();
        myRegions = new ArrayList<>();
        SimulationRun = 52 * 7 * 24 * 60;
        this.myFEL = new FEL();
        myRegions.add(new Region("Greater Accra", EventNames.RegionDrivingType.Easy, EventNames.RegionPatient.High, 45 * 5, 4 * 5, 31 * 5, 0, 123,
              60, 2, 3909764)); // Greater Accra - 123 Clinic - 60 TB Center 20*80
        if (!TBOnDemand) {
            myRegions.add(new Region("Volta", EventNames.RegionDrivingType.Medium, EventNames.RegionPatient.High, 56 * 5, 202, 237, 0, 64, 33, 0,
                  2099876)); // Volta - 64 Clinic - 33 TB Center 138*56
            myRegions.add(new Region("Upper East", EventNames.RegionDrivingType.Hard, EventNames.RegionPatient.High, 56 * 5, 85 * 5, 27 * 5, 57 *
                  5, 55, 22, 0, 1031479)); // Upper East - 55 Clinic - 22 TB Center 31*96
            myRegions.add(new Region("Northern", EventNames.RegionDrivingType.Hard, EventNames.RegionPatient.Low, 56 * 5, 57 * 5, 0, 27 * 5, 37,
                  28, 0, 2468557)); // Northern - 37 Clinic - 28 TB Center 150*293
            myRegions.add(new Region("Ashanti", EventNames.RegionDrivingType.Easy, EventNames.RegionPatient.Medium, 31 * 5, 19 * 5, 14 * 5, 6 * 5,
                  79, 50, 0, 4725046)); // Ashanti - 79 Clinic - 50 TB Center 100*100
            myRegions.add(new Region("Western", EventNames.RegionDrivingType.Easy, EventNames.RegionPatient.High, 14 * 5, 19 * 5, 0, 0, 52, 25, 0,
                  2325597)); // Western - 52 Clinic - 25 TB Center 100*70
            myRegions.add(new Region("Central", EventNames.RegionDrivingType.Medium, EventNames.RegionPatient.High, 31 * 5, 6 * 5, 14 * 5, 0, 55,
                  33, 0, 2107209)); // Central - 55 Clinic - 33 TB Center 50*70
            myRegions.add(new Region("Eastern", EventNames.RegionDrivingType.Medium, EventNames.RegionPatient.Medium, 45 * 5, 19 * 5, 31 * 5, 4 *
                  5, 62, 34, 1, 259013)); // Eastern - 62 Clinic - 34 TB Center 80*80
            myRegions.add(new Region("Upper West", EventNames.RegionDrivingType.Medium, EventNames.RegionPatient.Low, 27 * 5, 85 * 5, 0, 57 * 5,
                  21, 10, 0, 677763)); // Upper West - 21 Clinic - 10 TB Center 80*85
            myRegions.add(new Region("Brong Ahafo", EventNames.RegionDrivingType.Medium, EventNames.RegionPatient.Low, 45 * 5, 27 * 5, 0, 19 * 5,
                  10, 53, 0, 2356534)); // Brong Ahafo - 10 Clinic - 53 TB Center 110*202
        }
        for (Region myRegion : myRegions) {
            for (int i = 0; i < myRegion.getNumHospital(); i++) {
                myHospitals.add(new Hospital(this, Integer.MAX_VALUE, myRegion));
```

```
            }
        }
        if (TBOnDemand) {
            for (Region R : myRegions) {
                for (int i = 0; i < RandomCreator.numberOfVehicle; i++) {
                    myVehicles.add(new Vehicle(this, R, RandomCreator.numberOfDevices));
                }
            }
        }
        for (Region myRegion : myRegions) {
            for (int i = 0; i < myRegion.getNumTBCenter(); i++) {
                myTBCenters.add(new TBCenter(this, true, Integer.MAX_VALUE, myRegion));
            }
        }
        for (Region myRegion : myRegions) {
            for (int i = 0; i < myRegion.getNumClinic(); i++) {
                myClinics.add(new Clinic(this, true, 2, 2, myRegion));
            }
            new Patient(this, myRegion, 0);
        }
    }
    public void addToFEL(Event myEvent) {
        this.myFEL.addEvent(myEvent);
    }
    public double getTime() {
        return CurrentTime;
    }
    public double Run() throws IOException {
        CurrentTime = 0;
        double A = 0;
        while (CurrentTime < SimulationRun && this.myFEL.isRemainingEvents()) {
            Event myEvent = this.myFEL.getCurrentEvent();
            if (myEvent.getTime() < CurrentTime) {
                throw new IllegalArgumentException("CannotHappen!");
            }
            CurrentTime = myEvent.getTime();
            myEvent.Run();
        }
        KPICalculator();
        return KPI;
    }
    public CareCenter getCareCenter(Patient myPatient) {
        double distance = Double.MAX_VALUE;
        CareCenter mySelectedCenter = null;
        for (CareCenter myCenter : myClinics) {
            double myDis = Utility.getDistance(myCenter, myPatient);
            if (distance > myDis) {
                distance = myDis;
                mySelectedCenter = myCenter;
            }
        }
        return mySelectedCenter;
    }
    void addToAllList(Patient myPatient) {
        this.AllPatients.add(myPatient);
    }
```

```java
public String KPICalculator() {
    double totalProcessingTime = 0;
    int totalPatientProcessed = 0;
    double totalTimePatientSpentAtPOCQueue = 0;
    for (Patient myPatient : this.AllPatients) {
        if (myPatient.isDone()) {
            if (myPatient.getOut() > WarmUp) {
                totalPatientProcessed++;
                totalProcessingTime += myPatient.getOut() - myPatient.arrived();
                totalTimePatientSpentAtPOCQueue += myPatient.getOutQ() - myPatient.getInQ();
            }
        }
    }
    KPI = totalPatientProcessed;
    return totalPatientProcessed + "\t" + totalProcessingTime / totalPatientProcessed;
}
private void mySummary() throws IOException {
    KPICalculator();
    for (CareCenter myClinic : myClinics) {
        System.out.println(myClinic);
    }
    System.out.println("\nTB Centers");
    for (CareCenter myClinic : myTBCenters) {
        System.out.println(myClinic);
    }
    System.out.println("\nHospital");
    for (CareCenter myClinic : myHospitals) {
        System.out.println(myClinic);
    }
    System.out.println("\n\n");
    Event.myFile.close();
}
TBCenter getTBCenter(Clinic myClinic) {
    double distance = Double.MAX_VALUE;
    TBCenter mySelectedCenter = null;
    for (TBCenter myCenter : myTBCenters) {
        double myDis = Utility.getDistance(myCenter, myClinic);
        if (distance > myDis) {
            distance = myDis;
            mySelectedCenter = myCenter;
        }
    }
    return mySelectedCenter;
}
Hospital getHospital(TBCenter myTBCenter) {
    double distance = Double.MAX_VALUE;
    Hospital mySelectedCenter = null;
    for (Hospital myCenter : myHospitals) {
        double myDis = Utility.getDistance(myCenter, myTBCenter);
        if (distance > myDis) {
            distance = myDis;
            mySelectedCenter = myCenter;
        }
    }
    return mySelectedCenter;
}
```

```java
void receiveAPatient(Patient myPatient) {
    if (DistrictBase) {
        double distance = Double.MAX_VALUE;
        Vehicle myV = null;
        for (Vehicle myVehicle : myVehicles) {
            if (myVehicle.myRegion.equals(myPatient.getRegion())) {
                double dis = Utility.getDistance(myVehicle, myPatient) + 0.5 * myVehicle.myDeviceQueue.size() + .5 * myVehicle.myQueue.size() +
                        .5 * (myVehicle.isFree ? 0 : 1);
                if (dis < distance) {
                    myV = myVehicle;
                    distance = dis;
                }
            }
        }
        myV.notifyOfNewPatient(myPatient);
    } else {
        double distance = Double.MAX_VALUE;
        Vehicle myV = null;
        for (Vehicle myVehicle : myVehicles) {
            double dis = Utility.getDistance(myVehicle, myPatient) + 0.5 * myVehicle.myDeviceQueue.size() + .5 * myVehicle.myQueue.size() + .5
                    * (myVehicle.isFree ? 0 : 1);
            if (dis < distance) {
                myV = myVehicle;
                distance = dis;
            }
        }
        myV.notifyOfNewPatient(myPatient);
    }
}
void createDataFile(double Speed) {
    String Ans = "param c: ";
    for (int i = 0; i <= AllPatients.size(); i++) {
        Ans += Integer.toString(i) + "\t";
    }
    Ans += ":=\n0\t.\t";
    for (int i = 0; i < AllPatients.size(); i++) {
        Ans += Utility.getDistance(AllPatients.get(i)) / Speed + "\t";
    }
    Ans += "\n";
    for (int i = 0; i < AllPatients.size(); i++) {
        Ans += Integer.toString(i + 1) + "\t" + Utility.getDistance(AllPatients.get(i)) / Speed + "\t";
        for (int j = 0; j < AllPatients.size(); j++) {
            if (i != j) {
                Ans += Utility.getDistance(AllPatients.get(i), AllPatients.get(j)) / Speed + "\t";
            } else {
                Ans += ".\t";
            }
        }
        Ans += "\n";
    }
}
double getSpeed() {
    return this.myRegions.get(0).myType.getValue();
}
void getKPI() {
    double TotalT = 0;
```

```
        double TotalW = 0;
        double TotalF = 0;
        for (Vehicle myV : this.myVehicles) {
            TotalT += myV.TotalTravel;
            TotalW += myV.TotalWait;
            TotalF += myV.TotalFree;
        }
        System.out.println(TotalT + "\t" + TotalW + "\t" + TotalF);
    }
}
```