Optimal Electric Vehicle Charging Station Location Allocation using Agent-Based Modeling and Simulation: A case study of city of Montreal

Akhil Raj Kizhakkan

A Thesis

In the Department

Of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Master of Applied Science (Electrical and Computer Engineering)

at Concordia University

Montreal, Quebec, Canada.

December 2020

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: Akhil Raj Kizhakkan

Entitled: Optimal Electric Vehicle Charging Station Location Allocation using Agent-Based Modeling and Simulation: A case study of city of Montreal

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

Complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair

_____ Examiner

_____ Examiner

_____ Co-Supervisor

Dr. Akshay Kumar Rathore

_____ Co-Supervisor

Dr. Anjali Awasthi

Approved by: _____

Dr. Yousef R. Shayan, Chair
Department of Electrical and Computer Engineering

December 17th, 2020
Date of Defence

_____

Dr. Mourad Debbabi, Interim Dean
Gina Cody School of Engineering and Computer Science

Abstract

Optimal Electric Vehicle Charging Station Location Allocation using Agent-Based Modeling and Simulation: A case study of city of Montreal

Akhil Raj Kizhakkan

Widespread acceptance of all electric vehicles faces two major challenges. First being the higher price tag compared to a similar utility IC engine vehicle, while giving equal or lesser range. Second being the under-developed infrastructure support for refueling. Current trends in Electric Vehicle (EV) industry shows an increase in battery capacity and higher charging speed capabilities owing to an increased adoption of EVs. This thesis focuses on the second challenge of range anxiety of EV users due to lack of enough charging infrastructure compared to their gasoline powered counterparts. Public fast charging infrastructure is proposed as the solution to solve range anxiety and wider acceptance of EVs by public. As setting up the public charging stations at the initial stages of Electric Vehicle (EV) market penetration can be budget demanding, it is therefore crucial that the locations chosen should cover maximum demand at least cost and best convenience.

This thesis discusses the review of research publications focused on optimal placing of Alternative Fuel/Electric Vehicle Charging Stations (EVCS), by considering various approaches and models they have used. Heuristic methods of solving optimization problems was given an additional focus in the review. This thesis addresses the discrete, multi-objective, capacitated location allocation problem of electric vehicle charging station, using agent-based modeling. The developed model of EV trips in an urban environment and their charging events was fed with real data from city of Montreal, allowing the model to replicate real charging demand situations at the charging stations at various locations across the city of Montreal. A pareto optimization method is developed using a simple evolutionary genetic algorithm to find all the best trade-offs between each objective value. The multiple objectives considered are utilization of charger resources, the average reroute distance of EVs to reach a charging station and number of infeasible trips and optimization is done through running the agent-based model iteratively through the genetic algorithm, evolving its solutions in each iteration. Proposed solutions for each optimal objective value and solution with the best trade-off between the objectives are discussed.

Acknowledgements

Hereby, I would like to express my gratitude to several people for making this thesis work possible. This work and successful completion would not have been possible without their help.

I would like to thank the Gina Cody School of Engineering and Computer Science for giving me this opportunity to pursue my education at Concordia University. I would like to thank all the staff and faculty members of the ECE and CIISE department and School of Graduate Studies for their kindness and support.

I would like to express my deepest gratitude towards my supervisors Dr. Akshay Kumar Rathore and Dr. Anjali Awasthi for their guidance, hard work, expert suggestions and their continual support towards the successful completion of this thesis. Their patience, diligence and encouragement will motivate me throughout my life.

I would also like to thank my parents Rajan K and Praseetha K for their support and encouragement, along with other members of my family for always supporting me in my career endeavors. Without their everlasting emotional and moral support, this hard work would not have been possible.

I am very grateful to Pierre-Arbor Foundation for their assistance to my research efforts with scholarship and their continual support by providing valuable research insights and collaborations from their scholars, alumni and experts.

A special thanks to Stephanie Whitehouse and Emily Fjeldsted from the Student Success Center at Concordia for their continual support and motivation throughout my academic life at Concordia University.

I am thankful to Anylogic support team for their technical assistance and suggestions in debugging the Anylogic modeling software.

Last but not the least, I wish to offer special thanks to my friends and colleagues Karan Pande, Abhinandan Dixit, Mohsen Amoei, Ujjval Khanna, Safwan Ahmed, Amar Fayyad, Swati Tandon, Koyel Khatun and Negar Ghodsi for their constructive suggestions and ideas throughout my period of work.

Contents

List of figures

List of Tables

List of Abbreviations

BEVs	Battery-operated electric vehicles
EV	Electric vehicles
ARTM	Autorité Régionale de Transport Métropolitain
GIS	Geographical Information system
EVCS	Electric Vehicle Charging Stations
PEVs	Plug-in Electric Vehicles
BEVs	Battery Electric Vehicles
PHEVs	Plug-in Hybrid Electric Vehicles
HEVs	Hybrid Electric Vehicles
FCEVs	Fuel-Cell Electric Vehicles
PEM	Proton Exchange Membrane
LA	Location Allocation
DC-FCS	DC-Fast Charging Stations
GIS	Geographical Information System
FCLM	Flow Capture Location Methods
OD	Origin Destination
MILP	Mixed-Integer Linear Programming
FIFLM	Flow Interception Facility Location Problem
VMT	Vehicles Miles Travelled
MILP	Mixed-integer linear programming
PSO	Particle Swarm Optimization
ABM	Agent Based Modelling
UML	Unified modelling language
CS	Charging Stations
CMM	Montreal Metropolitan Community
RTL	Réseau de Transport de Longueuil
MTQ	Ministère des Transports du Québec
STM	Société de Transport de Montréal
STL	Société de Transport de Laval

SRM	Secretariat for the Metropolitan Region
AQTIM	Association Québécoise du Transport Intermunicipal and Municipal
AMT	Agency Metropolitan Transport
OO	Object Oriented
CSA	Charging Stations Agents
TPA	Trip planner agents
CA	Car agents
ToD	Time of Destination
GA	Genetic Algorithm
SMOGA	Simulation-based Multi-Objective Genetic Algorithm

Chapter 1

Introduction

1.1 Environmental Impact

As more and more countries formally commit to reduce their carbon footprint to tackle the climate change and global warming crisis, immediate measures need to be taken in all industrial, residential and transportation sectors to achieve the common goal. According to the US Department of transportation[1] the public transportation contributes to 29% of the total green house emissions in USA, among which personal modes of transport(Cars, SUVs and Pickups)

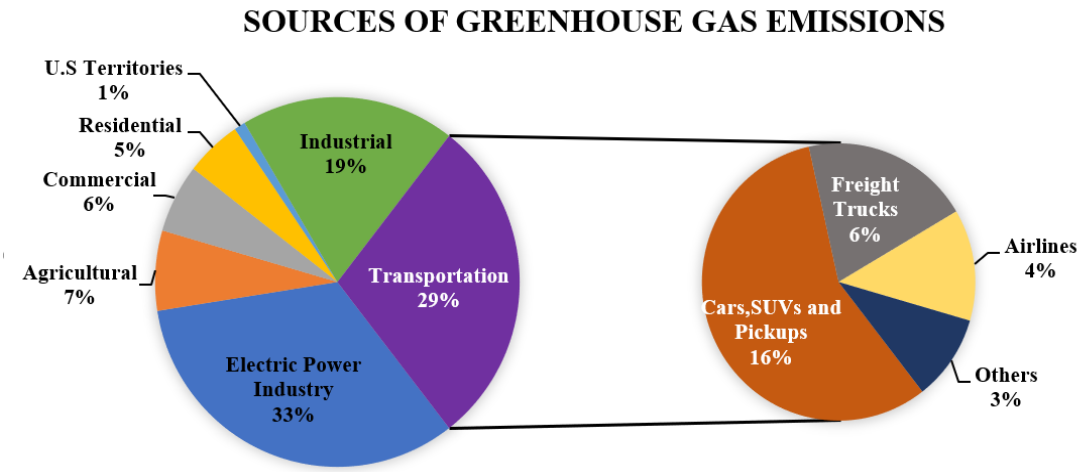

Figure 1.1 Transportation accounts for 29% of U.S Greenhouse Gas Emissions[1]

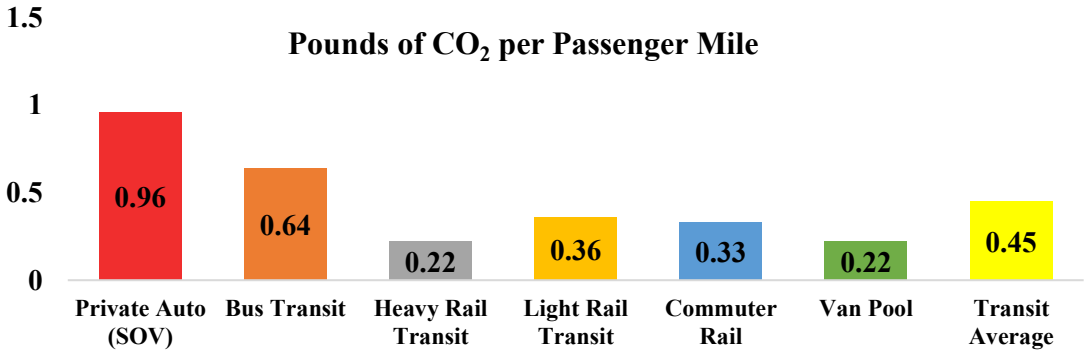

Figure 1.2 :Estimated CO_2 Emissions per Passenger Mile for Transit and Private Autos[1]

contribute to 57% as shown in Figure 1.1. As can be seen from Figure 1.2, personal use vehicles contribute to significantly higher per head carbon emission than other modes of transport.

1.2 Growth and Projected Penetration

Alternative-fuel based vehicles have been recommended as a sustainable way to travel and are gaining momentum due to recent popularity of electric vehicles. Statistically situations are still far away from the goal, as the global market share of electric vehicles is still under 1%[2] as shown in Figure 1.3, while the percentage projected growth of the same is promising as shown in Figure 1.4. While existing product line of the electric vehicles is close to achieving the desirability and

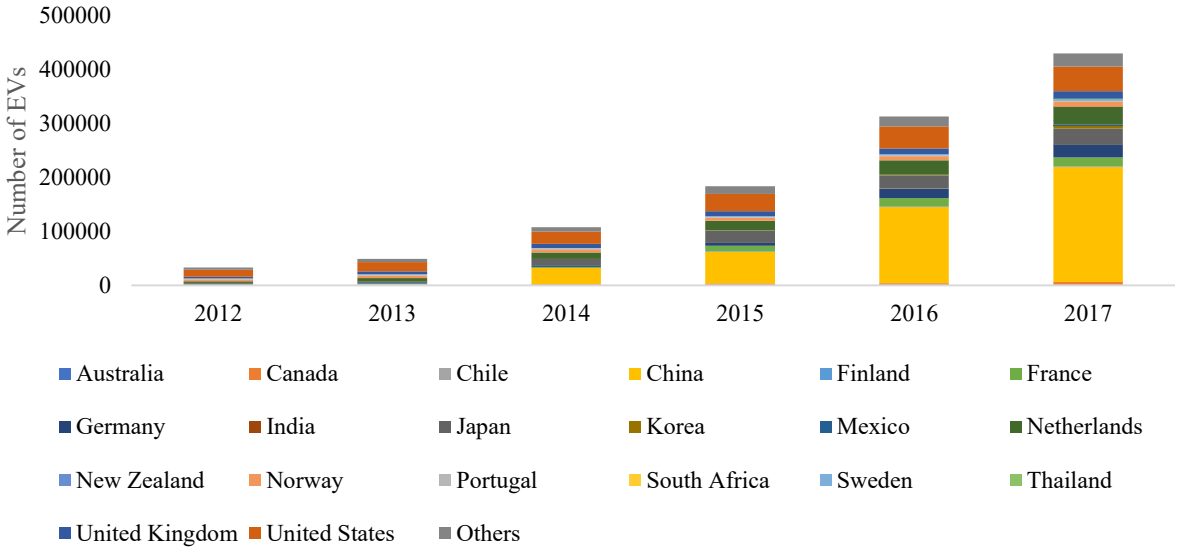

Figure 1.3 : Total number electric vehicles over the years[2]

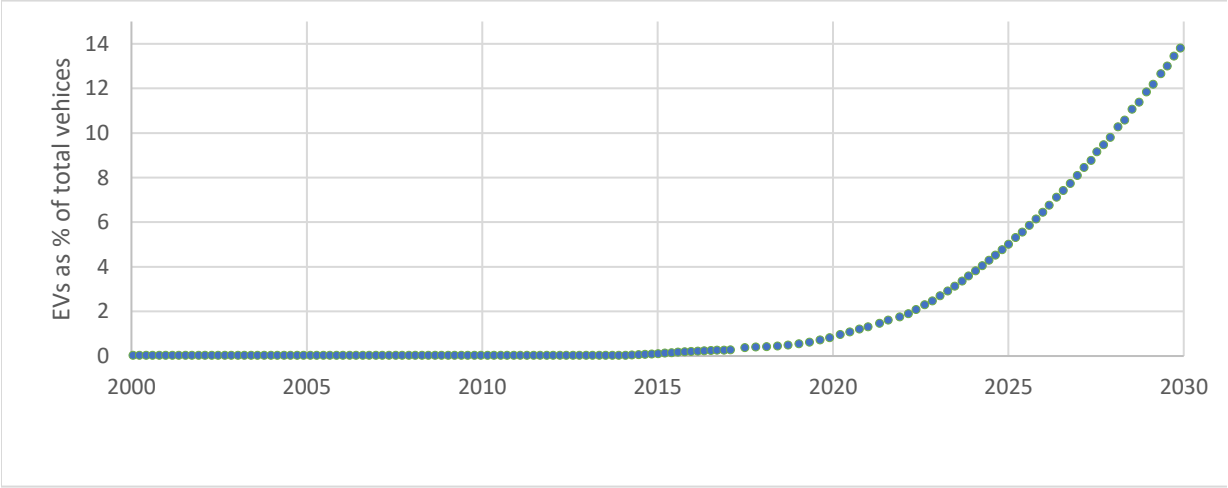

Figure 1.4 : Projected growth of Electric Vehicle Penetration (% of total vehicles) [2]

practicality in terms of driving range and performance to replace their gasoline counterparts, lack of sufficient and accessible charging stations is the foremost reason hindering further market penetration of EVs. Lack of accessible fast charging stations on routes leads to range anxiety, preventing more users to switch to electric vehicles.

1.3 EV Charger Classification

Based on the power and the range of voltages that are supported by EV chargers, they are classified into three levels[3]. A) lower than 3.7 kW are Level 1 chargers, B) between 3.7 and 22 kW are Level 2 chargers, and C) higher than 22 kW are Level 3 chargers as illustrated in Figure 1.5.

a. Level 1 Chargers : lower than 3.7kW
b. Level 2 Chargers : between 3.7kW to 22kW
c. Level 3 Chargers : higher than 22kW

1.4 Need for Fast Charging Stations

Considering the residential parking patterns in Montreal, where only a small proportion of population have access to personal garages and most users park their car in available parking slots in their respective streets, open space parking lots and nearest available locations, charging the car overnight using the in-built Level-1 charger is not an ideal option for every EV owner. Taking into account the projected growth of EVs, the only viable solution is to follow the gas station model of

PARAMETER	SLOW CHARGERS		FAST CHARGERS	
LEVEL	Level 1	Level 2	Level 3	Level 3
AC OR DC	AC	AC/DC	AC	DC
POWER RANGE	<3.7 kW	3.7 – 22 kW	22 – 43.5 kW	<400 kW
MODE	Mode 1 and 2	Mode 3	Mode 3	Mode 4
TYPE	Domestic sockets	IEC Type 1 IEC Type 2	IEC Type 2 IEC Type 3	CCS Combo 1 & 2 CHAdeMO, GB/T DC and Tesla connector
PLACE OF USE	Home	Home/Public	Public	Public
VEHICLES	2W, 3W, Cars	2W, 3W, Cars	Cars and Buses	Cars and Buses
CUSTOMERS	OEM/Retail	OEM/Retail, Charging Operators	Charging Operators	Charging Operators

Figure 1.5 Types of chargers and use cases [2]

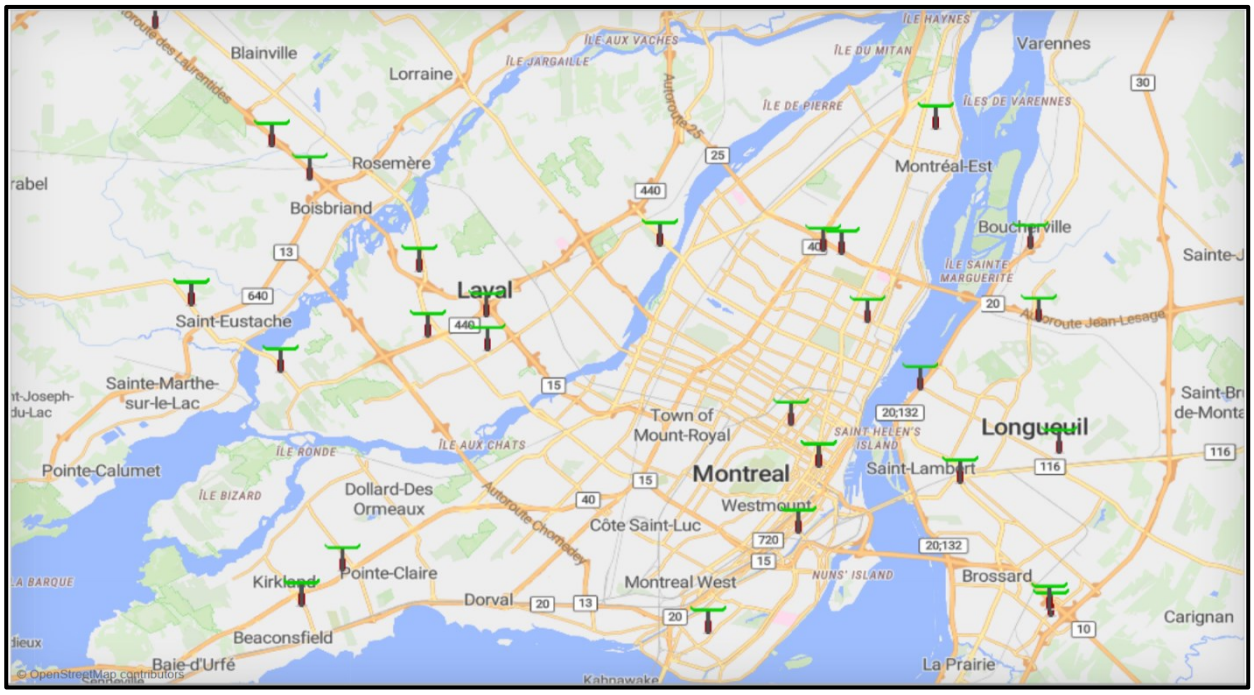

Figure 1.6: Distribution of DC- Fast Charging Stations in Montreal

meeting refueling demands similar to conventional cars, by setting up strategically located Fast Charging Stations in the different parts of the city to meet the charging demand of near future.

Figure 1.6 shows current distribution of DC- Fast Charging stations in Montreal (Circuit Electric Data 2019). With 52 Fast-DC charging stations spanned across 26 locations. The charging network can only cover 35 to 40% of the charging demand. Studies reveal that 83% of the consumers who wouldn't choose an EV over gasoline vehicle cite range anxiety as the primary reason, along with battery life concerns. While EV range has been steadily increasing over the years, led by market innovations from Tesla, they still lag behind the range offered by gasoline cars and refueling time. Availability of easily accessible fast charging stations with short charging times ensures a smooth transition to EVs from gasoline cars for all sections of population regardless of their access to the residential charging facilities.

1.5 Problem Statement and Research Objectives

This thesis addresses the multi-objective deterministic location allocation problem of EV fast charging stations. This involves selecting the minimum number of locations from a candidate set of slow chargers to install fast charging stations so as to cover the user demand for charging, while

minimizing the reroute time require to charge. This work stands apart from the existing research by two factors:

1. Using Agent Based Location allocation model for facility allocation and optimization.
2. Utilization of real population travel data from Autorité Régionale de Transport Métropolitain (ARTM) Montreal.

The agent based model of mobility of passenger cars through city of Montreal is developed by treating Cars, charging stations and trip planners as agents interacting in a Geographical Information System(GIS) environment. The trip planner agent initiates itself at the specified model time of the day and instructs the car agent to move from origin to destination. As the agents are inserted to the simulation model at their real departure times, allowing the model to simulate highly accurate traffic conditions and charging demand at each candidate solution of charging stations over a day.The multiple objectives optimized using this model are charging reroute time, number of fast chargers required and number of infeasible trips, while selecting its location in process. Simulation software Anylogic 8.6 is used as the modelling tool to verify and optimize the objectives.

The objectives of this thesis are listed below:

1. To develop an integrated agent-based model of urban mobility focused on electric vehicle passenger car trips and their charging events.
2. To develop a genetic algorithm based pareto optimization to find optimal location and number of chargers while minimizing cost of chargers, rerouting distance to charge and number of infeasible trips.
3. To verify the developed model with Anylogic 8.6 to investigate and record the effect of change in EV penetration on the proposed solution.

Design of each objective is discussed in Chapter 3 and the implementation and results are discussed in Chapter 4.

1.6 Thesis Outline

The major research contributions of this thesis are as follows:

In Chapter 2, literature review of presently used road traffic and traffic flow based optimization, electric grid voltage based stability and voltage regulation based optimization has

been discussed. Exact and heuristic solution approaches used to implement these methods are analyzed.

In Chapter 3, the design of the simulation model is presented extensively and discussed via UML diagrams.

In Chapter 4, agent-based definitions and the model simulation with Anylogic 8.4 software is demonstrated.

In Chapter 5, simulation model results are discussed in detail. Conclusion and recommendations for future works are presented.

1.7 Conclusion

This Chapter discusses the environmental impacts and ongoing developments in EVs and its promises for the world's shift towards sustainable energy and particularly towards reduced emission. EVs provide low-cost ownership, low maintenance, higher efficiency, higher fuel economy and high reliability leading to a rapid increase in the EV annual sales. Different types of charging facilities and its properties are also discussed in detail to show the importance of planning and installing more fast charging station in the city of Montreal to cope with the future demands of the electric vehicle population in the city and to boost desirability of the electric vehicles over gasoline vehicles among public.

In the next Chapter, a review of research articles focused on the optimal placing of Alternative Fuel/Electric Vehicle Charging Stations (EVCS), through various approaches and models are analyzed and discussed in detail. The Chapter also discusses the merits of using Agent Based Modeling to tackle continuous stochastic problems such as the dynamic facility allocation that forms this thesis.

Chapter 2

Literature Review

2.1 Introduction

In recent years global automobile industry has taken major strides in transition to Electric Vehicles (EVs), to move towards sustainable energy goals and tackle climate change. Rapidly falling battery costs and continuously improving charging technologies are bringing EVs on par with conventional vehicles regarding practical usage and range. Researchers from electrical and transportation industry have been studying optimal ways of distributing the Electric Vehicle Charging Stations (EVCS) in the past decade. EVs can be broadly classified as

a. All-Electric Vehicles , where the battery charging is the only source of refilling, also known as Plug-in Electric Vehicles(PEVs) or Battery Electric Vehicles (BEVs).
b. Plug-in Hybrid Electric Vehicles (PHEVs), where the vehicle can be refueled through multiple sources like gasoline and battery charging.
c. Hybrid Electric Vehicles (HEVs), where the vehicle is propelled using both Electric Motor and IC engine, but the battery is charged through regenerative braking or IC engine only.
d. Fuel-Cell Electric Vehicles (FCEVs) where the energy source is a Hydrogen Fuel Cell and the vehicle is powered by electricity produced through a Proton Exchange Membrane (PEM) decomposing Hydrogen(H_2) fuel through oxidation.

In this Chapter, all mentions of Electric Vehicles (EVs) is refers to All-Electric Vehicles.

The transportation research is focused on the cost minimization with accessibility and demand coverage considering vehicle movement pattern, user behavior and other constraints imposed by the user or the road network. Centralized strategic planning and optimization in EVCS location selection is proven to drastically reduce the initial cost required to serve the EV charging demand and reduce range anxiety [4]. In literature, it is noticed that multiple modeling approaches and problem-solving algorithms were used to optimize the required parameters.

Optimization is referred to as the process of finding the best feasible values (maxima or minima) for some objective functions, while satisfying some given domains and constraints. Location

Allocation(LA) problem involves locating an optimal set of facilities to satisfy customer demand at minimal transportation cost from facilities to customer or vice versa[5]. LA solution methods have been in practice for a number of applications such as location citing for warehouses, gas stations, fast food outlets, electric transformers, emergency health care facilities, urban planning, etc.

2.2 Classification of location allocation problem

There are four components that can forms any location allocation problem. According to Revelle and Eiselt [6], these are as follow:

a) Customers: EV owners present in the Origin Destination Survey 2013 data base from ARTM Montreal are the customers in this problem.
b) Facilities to be located: Here, thesis work intends to use the existing set of Level-I charging stations across the city as potential (candidate) solution space to be upgraded to DC-Fast Charging Stations (DC-FCS).
c) Space in which the customers and facilities are located: This thesis use Anylogic GIS enabled Agent Based Model is used simulate the LA problem.
d) A metric that indicates the distances or times between customers and facilities: This thesis use standard metric system, meters and seconds is used.

Based on these components, any LA problem can be classified into the following main categories:

2.2.1 Customer demand-based classification

1. **Deterministic**: If the customer population value, their locations and demands are predetermined and unvarying the model is called deterministic.
2. **Stochastic**: If the customer demands, population etc. are modelled with probability distributions, the models are called stochastic.

In this thesis, a data set of customer travel demands with definite departure times in a simulated Geographical Information System(GIS) environment, is used. Therefore, it can be termed as deterministic.

2.2.2 Facilities based classification

1. **Single or Multi-facility**: If the facility to be allocated is a single entity, i.e. one facility at the most optimal location, the LA problem is a single facility problem. But, if it is a population of solution sets, i.e. a collection of facilities at different locations, then it is a multi-facility problem. In the contrary case, the number of facilities to be placed may not be known in advance. In such case, idea is to find the least number of facilities so that all demand points are covered within a prespecified distance standard, also called as location set covering problems[7].
2. **Capacitated or uncapacitated**: If the ability of the facility to serve customer demand is limited, it is called a Capacitated facility, else Uncapacitated.

In this thesis, it is aimed to cite multiple DC-FCS in the city of Montreal, each of which capacitated to serve a limited number of customers at any given time. Hence, this model is a multi-facility-capacitated model.

2.2.3 Physical space or location based classification

Based on the representation of the space in which the facility citing is done, the LA models can be classified into planar problems (*d*-dimensional real space) and network location problems. Each of these can be subdivided into continuous or discrete LA problems[6]. Distances in planar problems are measured as a family of distances with a single parameter (Minkowski distances) and distances in network problems are measured on the network itself, typically as the shortest route between the two points through the network of arcs connecting them.

The above classifications can be further subdivided into continuous and discrete location problems. In continuous locations problems, the facility to be allocated can be placed anywhere in a continuous solution space, examples of these are Cellular tower location problems and helicopter trauma pick up location problems. In discrete problems[8] the facilities can be located on only predetermined eligible points on the plane or network.

Therefore, this thesis deals with a network based discrete LA problem.

2.2.4 Location Objectives based classification

Typically LA methods are used to allocate or install facilities close to the customer locations to obtain better values of the objective function. This involves maximizing the demands served (capture problem), minimizing the cost of transportation (median problem) or minimizing the maximum distance between the customer and the facility. In this thesis, it is aimed to model the problem as maximum flow capture and minimum cost of transportation problem (also called as min-sum).

Considering the above classifications, the optimal location allocation of EVCS problem can be termed as Multi-facility, capacitated, deterministic, network based, discrete, maximum capture and min-sum problem.

2.3 Methods for location allocation problem

There are several algorithms used to solve or optimize the objective function or mathematical models. Exact solution methods like branch and bound algorithm, hierarchical clustering methods and weighted Voronoi diagram approaches give exact locations of installation to cover the charging demand, leaving less room for flexibility for decision makers. Solutions that cannot be practically implemented due to the constraints out of the scope of the optimization model can have higher impact on the final outcome of the exact solution methods. While numerous earlier studies used exact methods, more recent trend shows a growing trend towards using heuristic algorithms or a combination of multiple heuristic algorithmic results to render the required result. While it takes longer time for the exact solution methods to solve larger and more complex network problems, heuristic methods give close to exact solution in less time and computational cost.

This Chapter discusses EVCS location planning strategies adopted by various researches through an extensive literature review and are broadly categorized into the following:

1. Exact solution methods
2. Heuristic methods

Both strategies aim to minimize the cost of installation while maximizing service quality, though a variety of different parameter sets and optimization approaches were considered to achieve the same. Exact methods give precise points of solution but demands more processing time when the

problem size is large. Heuristic algorithms on the other hand, provide local optima solutions, which are found to be very close to the exact solutions. Heuristic solutions can also be used where the model cannot have any exact solutions.

2.4 Exact Solution Methods

In computer science and operations research, exact algorithms always solve an optimization problem to optimality, if the model is in Polynomial Time, i.e it can have an exact optimal solution. The main advantage of this method is that it provides the best possible solution to the given planning and optimization problem. The exact solution models often provide flexibility of incorporating a myriad of constraints relevant to the practical problems but demand mathematical expertise and computation time to do so. It should also be noted that not all practical problems can be modelled as a linear optimization model due to intangible factors like fairness of a solution, plausibility of implementation etc.

This section discusses various Exact optimization approaches taken into consideration and algorithms used to optimize the EVCS location planning. The earliest location planning optimization models were based on the exact method approach, giving precise points of solution. (Note that this thesis use the words “refueled,” “recharged,” and “served” interchangeably, as well as “tank” or “battery”.)

2.4.1 Flow Capture Location Methods (FCLM)

The earliest adaptations of the optimization techniques to plan EVCS locations dealt with long distance travel in interstate highways using different forms of Flow Capture Location Methods (FCLM)[9] for refueling stations. The limited range of EVs is the major constraint in these models and the shortest path Origin-Destination(OD) matrix between the cities/towns was the environment in which optimization was conducted [10]. The possibilities of alternate routes and use of midlink service stations to cover shorter distance trips are shortcomings of these models. Use of heuristic algorithms like Greedy Adding and Greedy Adding with Substitution to solve Mixed-Integer Linear Programming (MILP) based FCLM was demonstrated to be faster as the complexity of problem increases, while producing suboptimal location values for shorter range values. More extensions of OD matrix models were based on the Vehicle-Routing logics [11], battery swapping models for short distance recreational park rides [12] etc. Uniform weight distribution of the

different paths of OD matrix resulting in over utilized and under-utilized stations is cited as the major drawback of these model solutions.

Flow based set covering models [7][13] were also proposed to focus on minimizing the cost of the installation by treating the location citing as a set covering problem. Flow capture method combined with the nearest facility method [14] by taking population into consideration proposes allocating charging stations at the freeway exits of densely populated areas through a data driven approach. This approach is seen to be widely used to cater the demands of initial EV population in urban environments. Researchers have also considered a Flow Interception Facility Location problem (FIFLM) [15] optimized using a binary integer linear programming model to analyze the flow captured with varying number of fresh installations and possible retrofitting EVCS on existing gas station infrastructure. The results showed that installing fresh EVCS has a significant coverage than retrofitted ones (85% flows captured with 20 EVCS, while retrofitted ones could reach only upto 55%). But considering practical limitations and cost to set up a fresh refueling station, it was suggested that retrofitted stations could still be a better alternative in the initial stages of electic vehicle deployment.

2.4.2 Fuel-Travel-Back methods

Fuel-Travel-Back [16] models promised a unique approach based on the notion “where you travel more is more likely you need refueling” arguing that O-D model does not consider short distance errand trips or trips with multiple objectives. This model relies on the spatial distribution of the Vehicle Miles Travelled (VMT) data to minimize the total fuel-travel-back travel time – defined as the distance for the fuel burned along the road to travel back to the nearest station. Therefore, VMT provides a heatmap of the fuel consumption density, which can then be used to determine the refueling locations. Probabilities of any random vehicle needing to refuel and its associated travel time is considered by the frequency of travel and the routes used, giving this model a more practical approach than the OD pair-based approach. The model assumes an average fuel-travel-back time to optimize the minimize the number of charging stations required to satisfy the same. While this solution appears very practical oriented for a city planning problem, collecting VMT data for an entire city road network is costly and complex. This approach also suffers from the lack of consideration for EV range.

2.4.3 Voronoi Diagram

Apart from the mathematical models, geometrical models are also employed in determining the optimal locations. Voronoi diagrams are the partitioning of a plane with *n* points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to its generating point than to any other. A Voronoi diagram is also known as a Dirichlet tessellation. The cells are called Dirichlet regions, Thiessen polytopes, or Voronoi polygons. In location optimization problems, modified Voronoi diagram approaches to consider weights and range of each cell has been used.

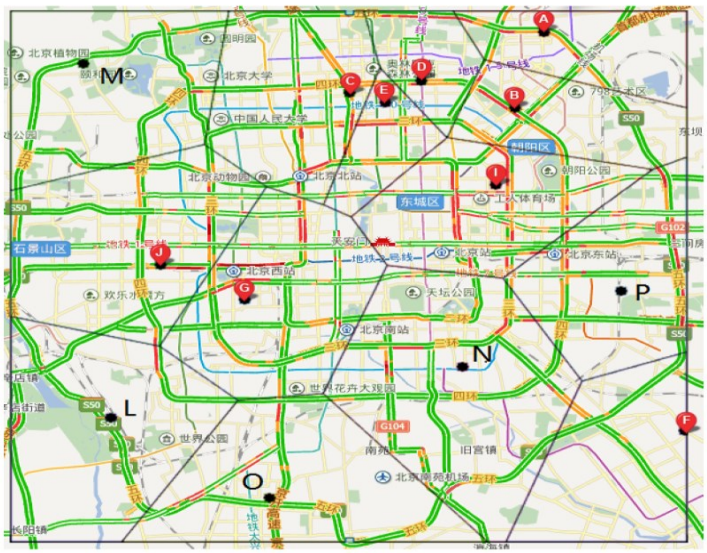

Figure 2.1 Voronoi Diagram method to find prospective locations for new EVCS [13]

Liu [13] proposed a model to assess the impact of incorporating three types of charging infrastructure, namely fast charging public stations, home charging posts and battery swapping stations and analyze its effects on the cost, charging time and impact on the distribution grid. This paper also considers retrofitting of existing gas stations with EVCS, considering their proximity from the electric distribution grid as a constraint for the selection. The location was determined based on the charging demand data through Thiessen Polygon (*aka Voronoi diagram*), due to the large geographical area to be covered (Figure 2.1). This paper also defined logical upper limits for workplace, public and residential parking area chargers to limit the maximum allowed distribution of these economical alternatives, compared to the costlier fast charging and battery swapping stations. Therefore, the faster alternatives are concentrated on the freeways for long-distance flows. While this approach gives practical results for a definite set of population with limited data on

actual mobility available, scaling the consumer base is a challenge and results tend to have high sensitivity compared to other traffic flow based approaches.

2.4.4 The p-Median Model

The p-median model is a location/allocation model [17], which locates p facilities among n demand points and allocates the demand points to the facilities. Several location planning models have used this approach and its extended versions [18] to calculate optimal locations considering various factors like user charging behaviors and EV range. The objective is to minimize the total demand-weighted distance between the demand points and the facilities. Basic formulation for a p-median problem can be expressed as below:

n = total number of demand points,

$x_{ij} = \begin{cases} 1 & if \text{ point i is assigned to facility located at point j,} \\ 0 & otherwise, \end{cases}$

$y_j = \begin{cases} 1 & if \ a \text{ facility located at point j,} \\ 0 & otherwise, \end{cases}$

w_i = demand at point *I,*

d_{ij} = travel distance between points i and j, wi =demand at point i,

p = number of facilities to be located.

where,

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} w_i d_{ij} x_{ij}$$

$$\text{s.t} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i,$$

$$x_{ij} \leq y_j \quad \forall i, j,$$

$$\sum_{j=1}^{n} y_j = p.$$

$$x_{ij = 0 \text{ or } 1} \quad \forall i, j,$$

$$y_{j = 0 \text{ or } 1} \quad \forall j,$$

This is an uncapacitated facility location model where each demand point is served by one facility and trips to the demand points are not combined.

2.5 Heuristic Methods

Research studies till 2010s mostly focused on the simpler and the exact optimization models, which are efficient at solving small scale data and providing exact solutions. But as the size of the problem and the number of parameters to be considered increases, these approaches tend to be less efficient in terms of computation time and problem-solving methods. Heuristic algorithms are developed to solve a problem quicker when classical methods are either too slow or fail to give an exact solution. This is achieved by letting the algorithm approximate the accurate solution, hence compromising the optimality, accuracy or precision for speed. A heuristic function ranks its solutions at each branching step based on the available information to decide which branch to follow. They are used to solve real life complex systems due to reasonable computation time demands and ability to handle large scale models without significant compromise in the accuracy of the solution. Also heuristic algorithms does not require a pre-generated data set with the all the possible combinations of the refueling stations that can serve a particular flow, which consumes more computation time than solving the MILP itself, thereby allowing heuristic based optimization techniques to provide solutions to complex real-life models within a reasonable time frame.

2.5.1 Greedy Adding and Greedy Adding with Substitution

In one of the firsts Kuby et.al [19] developed and applied three heuristic algorithms -greedy adding, greedy adding with substitution and genetic algorithm- to solve FLRM problems. It was shown to be much effective and efficient in solving complex models compared to exact approaches like mixed-integer linear programming (MILP). First proposed by Daskin M.S [20] the greedy algorithm first finds the facility locations that optimizes the objective, add this to the set, then select the next facility-which when added to the already selected-set will optimize the objective. This process of selecting and dding continues till the required number of facilities are selected. Starting with the set of fixed facilities specified by the user, the additional facilities selected should complement the fixed facilities to maximize the objective. It is known as greedy because it selects a facility at each iteration without looking ahead for the optimal set.[19] Greedy Adding with Substitution is an extension to Greedy adding algorithm, which tries to substitute the facilities from

the selected set with the unselected set after each selection has been made. If a substitution improves the optimality of the selected set, then the substitution is kept. The substitution loop runs after each facility is selected in the greedy adding loop. It is proven to be a great tool to solve maximum-covering, set-covering or FCLM problems and was found that over sufficient iterations, the accuracy never fell below 99.4% of the optimal solution.

2.5.2 Genetic Algorithms

Genetic algorithms which are modeled after the natural evolution process work in an entirely different way. The algorithm starts with the multiple sets of randomly generated solution sets, known as chromosomes. The algorithm mimics the natural interaction between subjects in an evolution set and comes up with a newer and generally better solution after each iteration. At each iteration, a specific number of solutions that best fit to the fitness function are selected (known as parents) from the set and used them to create the next set of solutions (known as children). The next set of candidate solutions can be considered diverse if the average length between the solutions is large. Having a diverse set of solutions is important to explore the required solution landscape. Proposed in early 70s by (Holland, 1975[21] and Mitchell, 1998[22]) it was first used to solve a P-median model problem [8] by encoding chromosomes as a binary string of the candidate location length (n), representing whether it is selected to locate a facility or not. To ensure the number of facilities chosen, they also introduced a penalty constraint. Later on [23] devised an extension to the previous model by encoding the chromosome to contain only the number of facilities to be selected (p) and using greedy-deletion heuristic to generate the child solution or chromosome. This approach has proven to be used for solving p-median, fixed charge, centroid, max-covering problems and FLRM problems.

Genetic algorithm with mutations were found to give minor improvements in results with much higher computation time and in some cases mutation allowed solutions to escape local optima solutions. Comparison of the three approaches showed that genetic algorithm approach was able to obtain marginally better results than the greedy adding and greedy adding with substitution algorithms, but at the expense of computation time [19].

2.5.3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization(PSO) is a relatively new addition in the population based optimization algorithms, where the particles or the objects fly through the multidimensional search space looking for a solution. The particle adjusts its position after each iteration, according to its personal best solutions and global solution of the whole swarm. For a D-dimensional search space, the position of the i^{th} particle is represented as $Xi = (xi_1, xi_2, xi_3, \ldots\ldots\ldots, xi_D)$ and the position of previous personal solution best for each particle is represented as $P_{Best\text{-}i} = (p_{i1}, p_{i1}, p_{i1}, \ldots\ldots, p_{iD})$. The best position among all is represented by $P_{GBest} = (p_{G1}, p_{G1}, p_{G1}, \ldots\ldots, p_{GD})$. The velocity of each particle is represented as $V_i = (v_{i1}, v_{i2}, v_{i3}, v_{i4}, \ldots\ldots, v_{iD})$ and is determined as a combination of its current velocity vector. The direction of personal and global best from the current position as shown in (2.1). The position of the particle at each iteration can be determined from the previous position and current velocity as expressed in (2.2).

$$v_{id} = a\, v_{id} + b\, r_1\, (p_{id} - x_{id}) + c\, r_2\, (p_{Gd} - x_{id}) \qquad (2.1)$$

$$x_{iD} = x_{iD} + v_{id} \qquad (2.2)$$

In (2.1), the first part represents the inertia component, ‘a’ being inertia constant fixed by the user. This value determines the extend to which the particle explores the search landscape or exploit the information from the personal and global bests, hence, is often dynamically varied through the search process. The second and third part represents contributions from the directions of the personal best and global best, with ‘b’ and ‘c’ as the acceleration constants fixed by the user, while ‘r_1’ and ‘r_2’ uniformly generated random numbers in the range of [0,1].

Taboo Mechanism based Binary Particle Swarm Optimization (TM-BPSO) [24] is another advanced heuristic method, which uses a list to store previous solutions and prevent visiting them again during further search. In this case, certain aspiration criteria are also used to prevent the risk of rejecting a solution that hasn’t been generated yet.

2.5.4 Agent Based Modeling (ABM)

Agent-Based Modelling (ABM) technique is an in-depth simulation method that has found various applications in the industry in the recent years. ABMs are able to mimic any real world complex system composed of autonomous components (*agents*), interaction between the components

(*behaviors*) and the changes in behavior of agents to the stimuli in their *environment*. ABM's conceptual depth is derived from its ability to model emergent behavior that may be counter intuitive and its ability to discern a complex behavioral system that is greater than the sum of its parts. ABMs inherent randomness or stochasticity lets the model proceed every practical way possible, which may not be explicitly visible to the modeler.

The simplicity of the model makes use of the current higher computational capabilities to model a real world system where each element (Agent) autonomously behaves according to the certain set of rules that defines them and interacts with other agents in the system doing it. This continuous and repetitive interaction derives desired results through computing all the possible scenarios.

2.5.4.1 Agents

Agents in an ABM can represent a variety of objects such as human beings, equipment, vehicles, companies or their projects, ideas, countries, etc. To ensure practicality of the systems modeled with the ABMs, an agent's fundamental characteristics described by Macal et al. [25] are as follows:

- An agent is an independent, modular, and identifiable individual. The modularity need implies that an agent has limitations. It means that one can easily specify whether something is a part of an agent or not or maybe is a shared and common attribute. Agents have attributes that allow the agents to be differentiated from and identified by other agents.
- An agent is autonomous and self-directed. An agent can work independently in its environment and in its interactions with other agents. An agent has behaviours, which connect information perceived by the agent to its decisions and actions. An agent's information is processed and notified through interactions with other agents and with the environment. An agent's behaviour can be clearly described by simple rules to the extent of abstract models, such as neural networks or genetic programs that relate agent inputs to outputs through adaptive mechanisms.
- An agent has a state that changes over time. It means that a system has a state consisting of the collection of its state variables. Besides an agent has a state that shows the crucial variables associated with its current condition. An agent's state contains of a set or subset of its attributes. The state of an agent-based model is the collective states of all the agents

along with the state of the environment. In an agent-based simulation, the state at any time is all the information required to move the system from that point forward.

- An agent is social having dynamic interactions with other agents that influence its behavior. Agents have protocols for interaction with other agents, such as for communication, movement and contention for space, the capability to respond to the environment, and others. Agents have the ability to recognize and distinguish the traits of other agents.

Additionally, agents may have extended characteristics as below :

- An agent may be adaptive, for example, by having rules or more abstract mechanisms that modify its behaviors. An agent may have the ability to learn and adapt its behaviors based on its accumulated experiences. Learning requires some form of memory.
- An agent may be goal-directed, having objectives to achieve with respect to its behaviors. This allows an agent to compare the outcome of its behaviors relative to its goals and adjust its responses and behaviors in future interactions.
- Agents may be heterogeneous. Agent simulations consider the full range of agent diversity across a population. Agent characteristics and behaviors can be different in their extent and complexity.

The agents for this Chapter are identified and described later in section 3.2.1 in Chapter 3.

2.5.4.2 Simulation software platform

This thesis model the Optimal Location Allocation of EV Charging Stations as an Agent-Based-Modeling problem using Anylogic 8.6 University Researcher Edition. Anylogic [26] is a multimethod simulation modeling tool developed by The Anylogic Company. It supports Agent Based, discrete Event and System Dynamics simulation methodologies. Anylogic was chosen over SUMO and MATSIM to model the problem discussed in this thesis due to the following reasons:

1. Built-in native java environment as well as support for extensibility using custom library codes and external sources.
2. Extensive support for GIS based ABMs including the shortest path routing and inbuilt functions for calculating the distance and time taken for each route.
3. Support for state chart and UML based process modeling.
4. Ability to dynamically create and simulate agents loaded from a database.

5. Data visualization support to analyze and present results.
6. Capability to use all CPU and GPU resources in a computer system to run optimization problems compared to other urban mobility simulators.

2.6 Conclusion

The discussion on studies based on literature review in EVCS location optimization shows that simpler exact methods of optimization are not viable for real world systems with multiple objectives to optimize and large complex data sets. Heuristic approaches on the other hand do not give the ideal accurate solution but provide reasonably accurate solutions in a fraction of time. A majority of exact solution approaches requires OD distance matrix, which require intensive computation resources to calculate the shortest distance between every combination of points. Heuristic approaches calculate the required distance values on the go, as only a fraction of the total combinations of OD distances may be required to model a scenario. Heuristic methods like ABM and Particle Swarm Optimization (PSO) offers great options to explore a highly diverse data set to find optimal solutions for single objective and multi-objective solutions with modified versions [27]. While Agent Based Modeling simulation offers a simplified modeling approach relying on the computational power while exploring the randomness of the real world system, it also offers a unique upper hand compared to other modeling solutions by queue implementation, which lets the modeler control and simulate waiting periods at each facility.

As the problem at hand – the pareto optimal location allocation of EVCS – is a Multi-facility, capacitated, deterministic, network based, discrete, maximum capture and min-sum problem, Agent Based Modeling on a GIS road network platform is the best method to deliver an optimal solution.

Chapter 3

Solution Approach

3.1 Introduction

Location Allocation (LA) problems have high relevance in the operations research field due to their wide spectrum of practical applications in shipping logistics, industrial management, urban planning, transportation infrastructure planning, etc. Owing to its stochastic and complex nature most of these problems are categorized as NP-hard problems, motivating the search for heuristic and approximated algorithms to solve them. Geographical Information System (GIS) based solutions are currently widely used to represent LA and other spatial problems. GIS plays the role of a practical tool capable of dealing with problems that are not easy to model and whose data is are available through various data management systems. This Chapter discusses the methodology adopted by this thesis to solve the LA problem of optimal EVCS through ABM.

3.2 Agent-Based Model for Location Allocation

Agent Based Modeling (ABM) is a fairly recent alternative to the mathematical modeling of the systems whose behaviors are concurrently distributed, complex, and heterogeneous. Like in the case of the LA problem discussed in this thesis – a spatially distributed discrete search space of the solutions that has the inherently complex nature of utilization factors owing to the complexity of urban mobility. ABM forms an effective tool to enhance the capabilities of the complex spatial analytic problems using GIS by minimizing the computation requirement to feed the optimization problem while providing a flexible, visual, and responsive representation of the problem itself [28].

In this thesis, the location planning of Fast Charging Stations (FCS) is modeled as a multi-objective optimization to maximize the utilization of charging stations, minimize the reroute time required to access the proposed solution of FCS and minimize the number of infeasible trips. The objectives, utilization factor and reroute factor are computed through ABM implementation. This thesis use Anylogic 8.6 to model the mobility of EVs in the city of Montreal by making use of a solution set of Charging Stations (CSs) to complete their commute. The Car Agents are moved from their

origin to destination through the shortest path provided by the Anylogic Router. The utilization factor of CSs is computed by considering the amount of time each charger was used and its convenience factor is computed considering the reroute time of each vehicle, which utilized that CS.

3.3 Overview of the datasets used

To enable the most accurate representation of the mobility of EVs in the city of Montreal, Montreal Origin-Destination (OD) survey (2013) data is used to model the Car Agent population in ABM. Charging Stations are plotted according to the information available on the website of Electric Circuit – Hydro-Quebec's public charging network – collected as of January 2020. At the time of data download, there were 6 locations with Level-3 (50kW) charging stations in the City of Montreal. For the sake of simplicity, all Level-2 and Level-3 charging station locations are assumed to be the candidate locations for future 120kW FCS

A brief overview of the Montreal OD Survey-2013 is discussed below.

3.3.1 Origin-Destination Survey 2013

The 2013 Origin-Destination survey is a joint achievement of the Agency Metropolitan Transport (AMT), the Association Québécoise du Transport Intermunicipal and Municipal (AQTIM), the Montreal Metropolitan Community(CMM), the Ministère des Transports du Québec (MTQ), the Réseau de Transport de Longueuil (RTL), the Secretariat for the Metropolitan Region (SRM), the Société de Transport de Laval (STL) and the Société de Transport de Montréal (STM), prepared and published by Secretariat for the Origin-Destination survey. This extract represents a sample of 78,800 households.

The 2013 OD survey was conducted in the fall of 2013 in the Montreal metropolitan area. This is a survey carried out through telephonic interviews that is among the most important in Quebec. It aims to draw a faithful portrait of all trips carried out by residents of the region during an average weekday, for all modes of transport used. Carried out approximately every five years since 1970, OD surveys cover an increasingly large, which spans the entire metropolitan area, major cities of Montreal, Laval, and Longueuil, passing through the north and south crowns.

The population targeted by the 2013 OD survey corresponds to all the people occupying private dwellings in each of the municipalities in the survey area. For people aged 4 years and under, no travel information is collected. The results of the survey refer to trips made on working days in the period between September 3 and December 20, 2013, inclusive. Interviews, carried out with some 78,700 households, made it possible to describe the characteristics of some 410,700 trips made by the 188,700 people who make up these households. The sampling covered in this survey is calculated to be about 3% of the total population of the Metropolitan area of Montreal.

Trip information collected, relevant to this thesis includes:

1. Household number
2. Person number
3. Reason
4. Origin
5. Destination
6. Time of departure
7. Mode of transport used.

Source : Enquête Origine-Destination 2013 de la région de Montréal, version 13.2b

Utilized by : Prof. Anjali Awasthi, Akhil Raj Kizhakkan.

3.4 Assumptions and constraints

Constraints faced in implementing an ABM using any software are many. Though Anylogic is a strong simulation modeling tool, there is a lack of technical documentation to help explore the advanced properties of the software. Implementing a highly specific model for an LA problem requires extensive use of programmable entities, knowledge of how to use them, and their inter-compatibilities in the Anylogic context, which is time-consuming to gather. The complexity of the software aside, the biggest computational requirement for any mobility model is to decide the route between two points through the GIS space. Anylogic offers the shortest route calculation as a part of its GIS module, by accessing its routing servers online. Though it takes several hours to learn all the required routes over the network, they are saved to a cache for easier and faster access in the subsequent runs. This eliminates the computational complexity involved from the system in which the model is run and thereby decreasing the model run time. This also means that mobility

between two points in GIS space is always through the shortest route obtained, which may not be the case practically.

The OD data obtained is considered as the base for all mobility events modeled in this thesis. As they are limited to only the Fall season (September-December) the model does not give an accurate picture of the seasonal variations in traffic over a whole year. As September to December notes the most change in climate and outdoor activities involved for the City of Montreal, it can be argued that for a 3-month survey aimed at representing the general mobility pattern of Montreal, the Fall season would be the best candidate.

Considering the projected growth of EVs (Figure 1.4), it can be found that EV population is currently at about 1% of the total vehicle population and is expected to reach 3-4% by the year 2025. From the sample of population covered in the OD Survey-2013 data, it can be seen that about 3% of the target population participated in the survey, and trips are defined for commute made by them using personal and public transports. This thesis assume that all the trips listed in the OD Survey-2013 are made using personal EVs so that the model has an accurate representation of charging demand for upcoming years.

As Montreal residential areas parking works on a sector parking system, this thesis assume that the bulk of the future EV owners will not have a facility to charge their EVs residentially overnight, as they may often have to park the car away from their home. Therefore, this thesis assumes that future EV charging demand rely on public charging stations to satisfy their re-charging needs. To facilitate shorter waiting periods for charging and to make EVs more acceptable to the broader public, Fast Chargers(~100kW) are required to be commonly accessible. Therefore, this thesis proposes a solution wherein total demand for charging is to be covered by Fast Charging Stations (FCS) only.

To eliminate infeasible solutions of locations to install an FCS, the sample set of solutions is gathered as the current Level-2 charging station locations. The final solution to the LA problem may be used to upgrade these Level-2 stations to an FCS of proposed capacity.

As the Vehicle Agent movement is modeled according to the departure time listed in the trips data, the peak hour of charging demand is accurately modeled in this solution. However, the model does not consider the general road traffic due to non-EVs and hence the travel times of Car Agents are

not accurately modeled in this thesis. It can be argued that, as all the agents face the same traffic constraints to travel, this assumption does not impact the final solution of the location of charging stations to be upgraded to FCS.

This thesis assumes that each EV owner will follow instructions from a higher-level decision maker to charge the car when instructed during a trip, given the origin and destination is provided to the decision-maker. This system is currently in use in several EVs where a central console provides the navigation details to the destination and informs the driver on estimated battery level upon arrival at destination. The system also urges the user to visit a charging station on-route if the estimated level is less than a certain tolerance. This thesis model considers the tolerance to be 35%.

The EVs available in Canada have a wide range of battery capacities available ranging from 16kW (Mitsubishi i-MIEV) to 100kW (Tesla Model S, Model X). For the sake of computational simplicity, this thesis assumes 3 values of battery capacities [25kW, 50kW, 75kW] are evenly distributed among the Car Agents. Also, the consumption of charge by different sizes of EVs are assumed to be very similar around 259Wh/mi or 161Wh/km, except for a few outliers (Jaguar i-Pace, Tesla's performance version cars, etc.) [29]

To summarize, the constraints are as follows:

a. Vehicle mobility through the shortest path only.
b. Limited data are available to model seasonal demand changes.
c. Only current Level-2 charging station locations are considered in solution space.
d. Real traffic congestion data not involved to reduce complexity.

The assumptions made are summarized below:

a. All trips from the database made by EVs.
b. All charging events through public FCS only.
c. Car Agents adherence to charging instruction from decision-maker
d. Generalized battery capacities and uniform consumption across all EVs

3.5 Modeling by Unified Modeling Language (UML)

The use of graphical designs to represent and explain the properties and flow of a model has always been in use. In the case of ABMs, most researchers agree that the natural way to program their models is to adopt Object-Oriented (OO) practices like abstraction, encapsulation, and inheritance. Among the OO analysis and design techniques applicable to ABMs, UML is the most favorable.

This thesis adopts the integration of OO, ABM, and UML for modeling the simulation-based LA problem optimized using the Genetic Algorithm. ABM with OO design has inherent advantages of balancing visibility and confidentiality through encapsulation. Inheritance provides a hierarchy structure that makes the design highly reusable, extendible, and easy to understand. Systems modeled with OO based ABM is more adaptive to changes over time while reducing the risk of building complex systems as they can evolve from fundamentally basic systems.

Unified Modeling Language (UML) proposes a set of well-defined and standardized diagrams (independent of any programming language or computer platform) to naturally describe and resolve problems based on high-level concepts inherent to the formulation of the problem. UML is a graphical tool that provides a highly communicative way of representing both static and dynamic aspects of a system. It is comprised of 13 basic diagrams that encourage users to focus on the end-to-end logical modeling of their solution methods, before diving into the technicalities of implementation.

The first and most important step in ABM with OO problem solving is the construction of a model. The model forms an abstract version of the relevant details from the usually complex real-world problem. The model in this thesis should facilitate a simple, comprehensible version of mobility in the city of Montreal, taking into account its major actors – EVs, Charging Stations, and the trips that direct them.

UML effectively represents the static elements and the dynamic nature of the LA problems. In our case, the Class Diagram gives an insight into the static definitions of each element in the system, while the sequence diagram gives a deeper insight into the flow of processes at different points of the simulation. While the state diagram defines the properties of each Agent state, the activity diagram defines the logical flow of triggers that result in achieving different states. The diagrams

work together to depict and describe different aspects of the system. Hence, UML diagrams are effective as a model design tool to be used for ABMs.

In the case of ABMs, it is found that 4 types of diagrams namely Class Diagram, Sequence Diagram, State Diagram, and Activity Diagram can almost accurately represent any ABMs.[30] Class diagrams are composed of classes and the multiple types of relationships among them, namely inheritance, composition, and association. This section defines all the major aspects of the model – agents, their physical sites, the resources they may consume, and their properties. Then each element is connected with the others depending on how they are related to each other or how one uses the other.

The sequence diagram is the second most common UML diagram for ABM developers. It represents how objects interact and exchange messages over time. This allows developers to trace the program while it executes and to follow the way objects interact in memory. UML is constantly evolving by adding more functionalities to these diagrams that make it closer to adapting any kind of software or modeling system.

State diagram gives all the possible states an agent can be in and all possible transitions between the states, mostly based on some conditions. This diagram always starts with an initial state and a final state, typically represented as Agent entering and leaving the ABM model. This allows perception of Agent behaviors as states and state-transitions, providing a convenient way to modularize the model code into behavioral blocks.

Activity diagrams are very similar to traditional flow charts, wherein it represents the procedural flow of code. Activity diagrams are considerably helpful in debugging the model for errors and issues, letting the developer visualize the behaviors of collaborating elements. The activity diagram gives a deeper insight into the state and sequence diagram.

For the purpose of modeling the LA optimization problem as an ABM, this thesis employs these 4 types of diagrams to model the static elements and dynamic behaviors of the system. The coming sections discuss the choice of such elements and behaviors to best represent the model through the 4 diagrams in sections 3.5.1 through 3.5.4

3.5.1 Class Diagram

The first step of an OO based model is to identify the classes involved to develop a library of agents and their related objects. In UML, class diagrams are used to represent the agents and their related objects. Once the agents and objects are identified, they can be organized into a hierarchy structure indicating their relationships. The broadness and depth of the modeling environment for specific cases are defined using the object definitions and agent relationships.

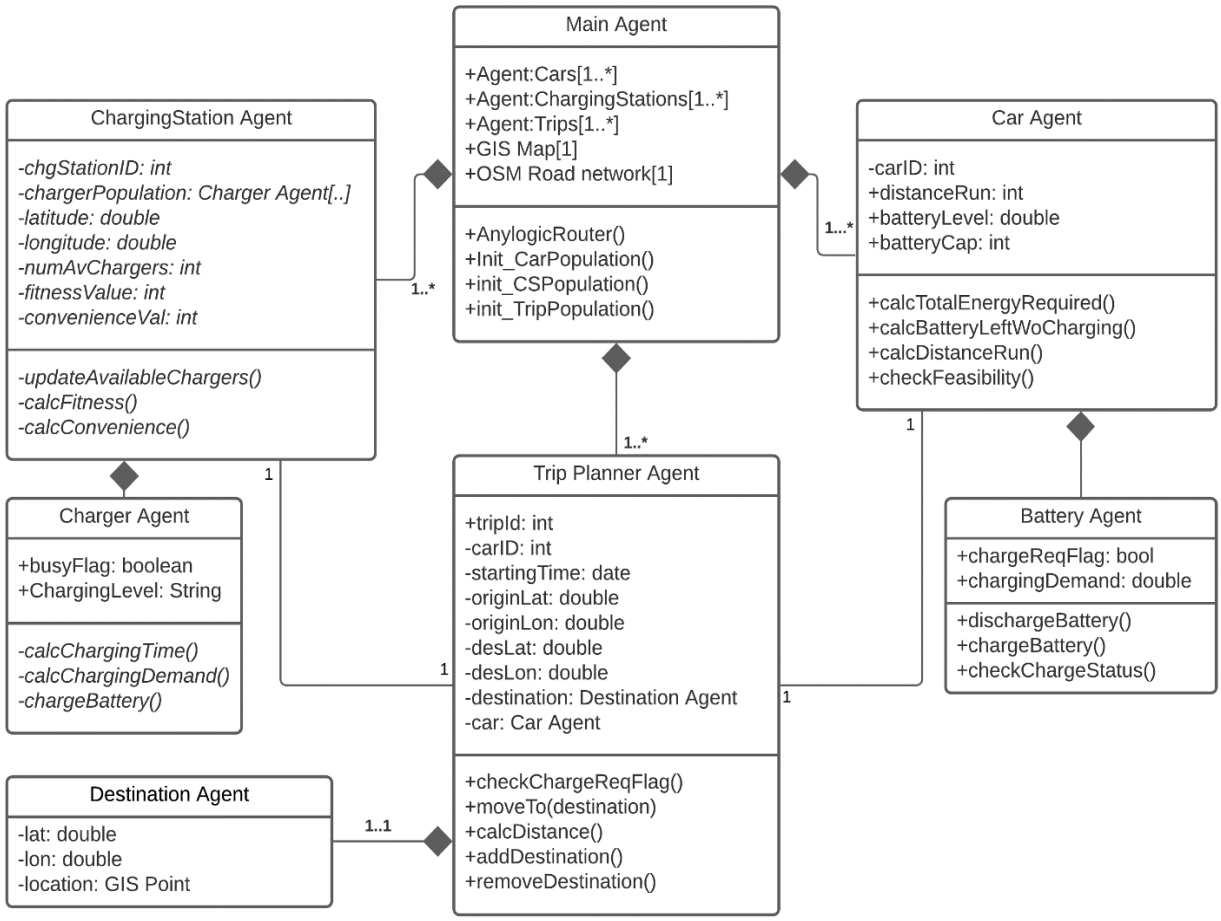

Figure 3.1 Agent Classes

This thesis defines classes according to the real-world entities each element represents. The 'main agent' acts as the base of the model, within which all other objects and Agents lie and interact with each other. Hence the GIS map and the road network upon which the population of trip, car, and charging station agents interact are present in the main agent. From Figure 3.1, it can be noted that the Main Agent is composed of Car Agent, Trip Planner Agent, and Charging Station Agent,

indicated by 1 to many relationships. Similarly Charging Station Agents (CSAs) is composed of Charger Agents (CGAs) and Car Agents (CAs) are composed of Battery Agents by a 1 to 1 relationship. In these cases, an agent is contained inside another agent using composition relation, when the second object leaves the model memory the contained agent also ceases to exist. In our case, the Main Agent (MA) takes the responsibility of creating and initializing all the agents inside that drives the simulation. At the disappearance of the MA object, the contained agents, objects, and resources are taken out of the simulation as well.

Trip Planner Agent (TPA) is related to the CSA in a 1 to 1 association, as each trip can only have one nearest charging station on the route. But TPA is related to the CA as many to 1, as multiple trips entries can call the same CA to complete the trip. Significant agents present in this class diagram are described as follows:

1. Main Agent (MA)

   The Main Agent forms the environment in which the Agents of this model interact. The shortest path router algorithm is part of the GIS map in this agent, which lets Car Agent move from origin to destination through an accurate road network and give necessary distance data to make trip decisions.

2. Trip Planner Agent (TPA)

   This Agent is responsible for the details of each trip Car Agents have to make in the model. Hence it is associated with both the Car Agent population and Charging Station Agent population. Each trip agent becomes active only at the departure time of the respective trip. Once active it fetches the corresponding Car Agent and proceeds through a decision-making process to complete the trip. The process flow detailing this decision-making process is discussed in the next section.

3. Car Agent (CA)

   Car Agents are the only non-static agent in this model. It moves according to the instructions from the trip Agent, through the path provided by the shortest path router. It dynamically consumes the battery resources as they move and keep a track of the average distance travelled a day. The car agent is composed on a Battery agent that monitors the battery state and sets a flag indicating charging required when the battery is low.

4. Charging Station Agent (CSA)

This agent hosts the charger agents at the desired location and allocates incoming Car agents with one Charger agent from its pool, with a facility of keeping up to one car in the queue. The resource pool and queue implementation of chargers ensure that each charging station is met with realistic charging demand according to the number of chargers present in them.

3.5.2 State Transition Diagram

A State Transition Diagram shows the flow from one state to another within an agent. It illustrates all possible states an agent can be in and all possible transitions between those states, which can result from an event based on some conditions if any. State transition diagram, in some cases also known as Activity diagram, always starts from an initial state – indicated by a black disc – and ends at a final state – indicated by a black disc inside a white disc. Each round-corner box includes an execution of a statement or activity and the arrow from the box leads to the next step of activities. The transition between the states provides important perspectives to sophisticated operations.

The next sections discuss the State transition diagrams of agents in this model.

3.5.2.1 Trip Agent State Transition Diagram

The diagram in Figure 3.2 details the end-to-end process flow of the simulation model, as trip events are initiated from the Trip agent. Once the trips population in the Main agent is initialized from the database, the Trip agent stays idle until the departure time is provided with the trip event. As the model time approaches the departure time, the trip details are pushed to the Car Agent's trip-queue to be executed as soon as the Car Agent is released from its past trip. This ensures a Car Agent is not called into the trip process flow while it is active in another process flow. During the trip process, the feasibility of reaching the destination with a 30% battery left in the car is checked and the car is sent to the charging station on-route if it is not satisfied. Trip Agent calculated the fitness and convenience factors for Charging Station agents in use and pass it on to the Charging Station Agent. If the Car Agent does not have the required energy to travel to the nearest Charging Station, that trip is marked as infeasible and moved on to the next trip.

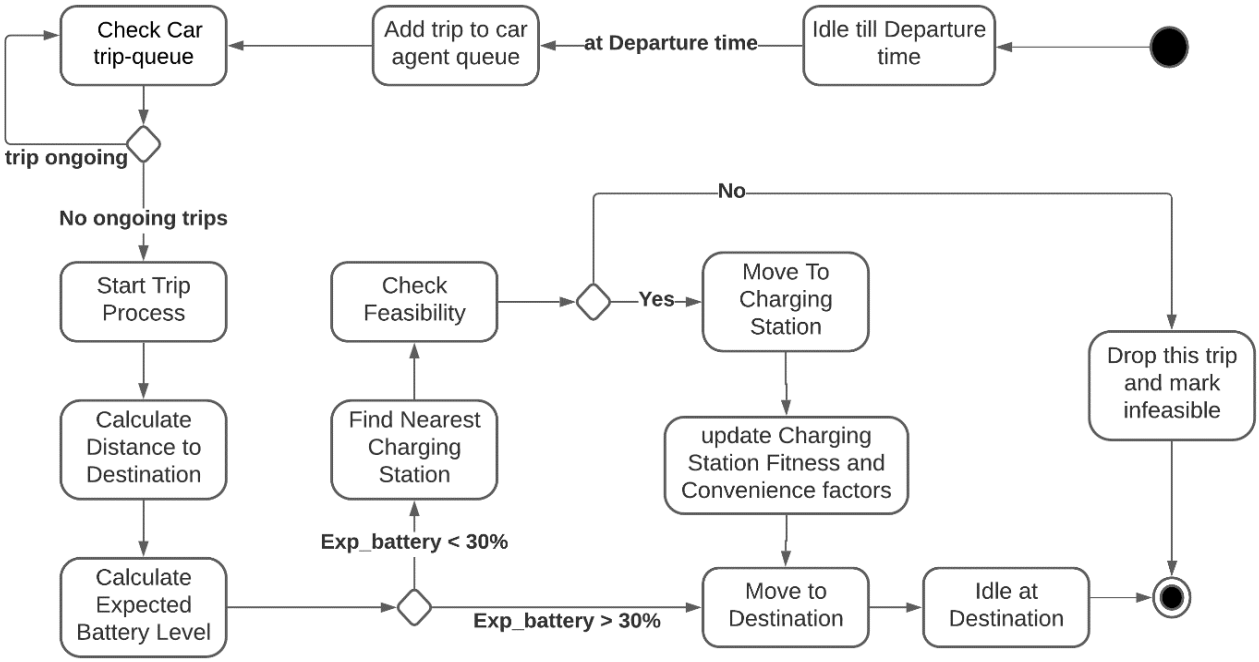

Figure 3.2 Trip State Transition Diagram

3.5.2.2 Car Agent State Transition Diagram

Car agent has a relatively simpler State Transition diagram (Figure 3.3) in this model, as it follows the decision made by TPA for its commute destination. The Car Agent is idle until there is an entry in the trip-queue. Upon receiving a trip object, the Car Agent calls for its trip process flow and

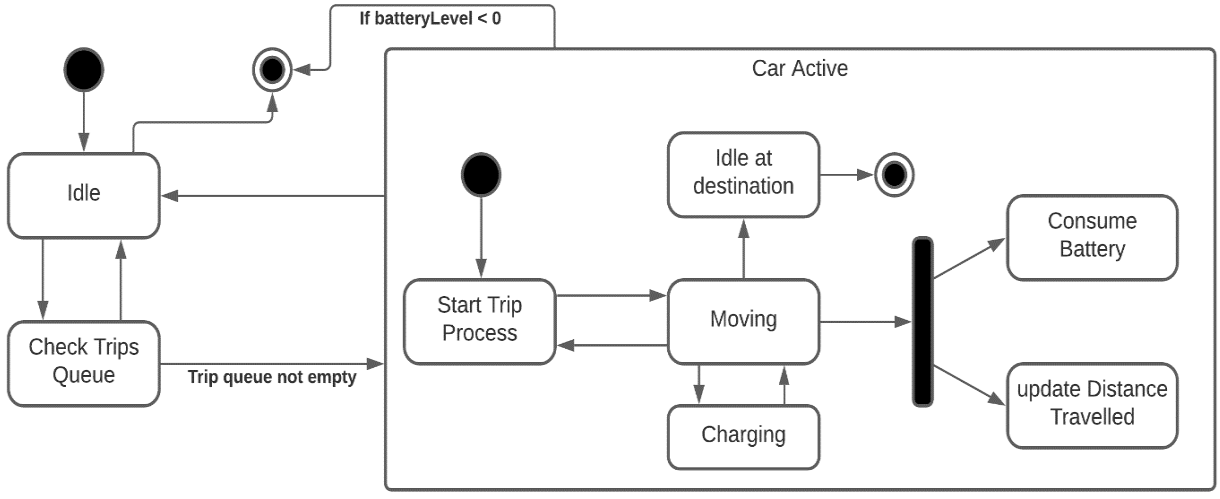

Figure 3.3 Car Agent State Transition Diagram

moves while updating its distance traveled and battery charge level through the Battery Agent. The CA is taken out of the simulation if at any point the energy level in the battery reaches zero.

3.5.2.3 Battery Agent State Transition Diagram

Battery Agent continuously monitors the state of the battery charge and sets the charge required flag dynamically, as the vehicle moves and energy in the battery is consumed. When the charge required flag is set, the battery agent interacts with the CGA in charging station to determine the time required to charge and updates the battery level accordingly (Figure 3.4).

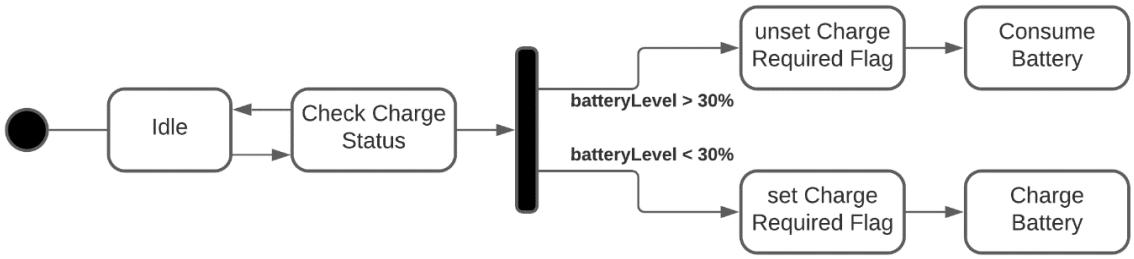

Figure 3.4 Battery State Transition Diagram

3.5.2.4 Charging Station Agent State Transition Diagram

The charging Station agent stays idle until a CA arrives. A CGA is assigned from the resource pool of chargers to the Car Agent. Once the charging process is done at the CGA, the CA is passed back to the TPA to continue with the trip. The TPA also passes data required for assessing the fitness and convenience factor of the Charging Station agent (Figure 3.5).

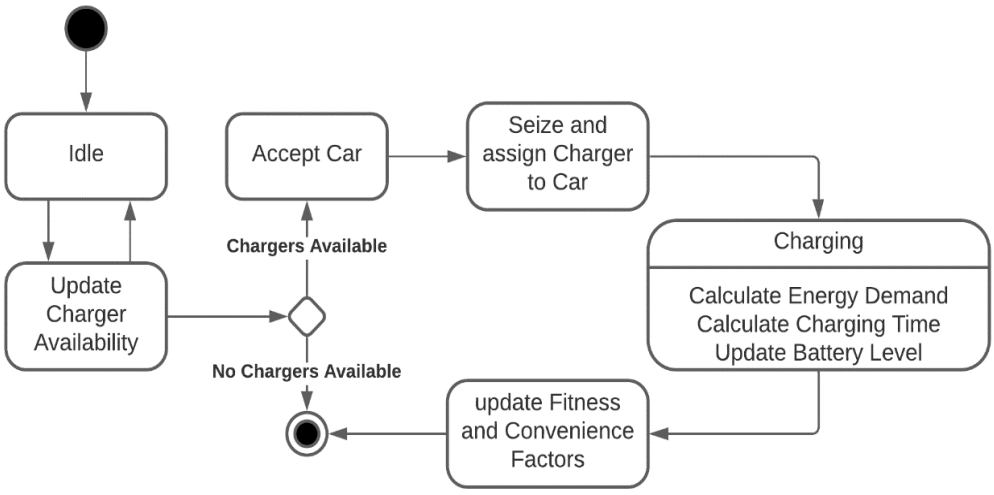

Figure 3.5 Charging Station State Transition diagram

3.5.2.5 Charger Agent State Transition Diagram

The charger agent is responsible for the actual charging process of the battery agent composed in the CA received by the Charging Station. It calculates the energy demand and corresponding charging time of the charging process. The charger agent limits the maximum charging time to 20 minutes as this thesis is modeling how FCS can serve the EV user demand as close to a conventional car refueling process (Figure 3.6)

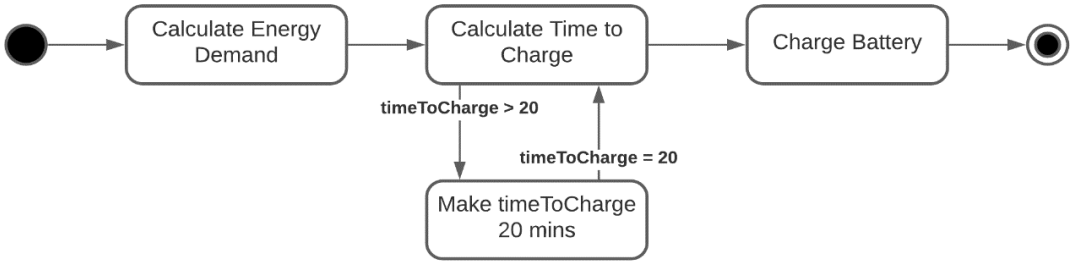

Figure 3.6 Charger State Transition diagram

3.5.3 Use Case Diagram

Use case diagrams are mainly employed in modeling the behavior of a system, a subsystem, or a class. They are useful in visualizing, specifying, and documenting the behavior and dependencies of an element of a system.

To model the EV Charging Station Location Allocation problem through Agent-Based methodology, the Agents and its behaviors must be clarified. Through UML use case diagrams, one can represent the scenarios or cases of using each agent class. The use case diagram can explain and indicate needs in the scenarios. Therefore, use case diagrams gather the behaviors and functional requirements of each system and define its associations with other external agents. Figures 3.7 through 3.12 show the use case diagram for the complete model through its various agents.

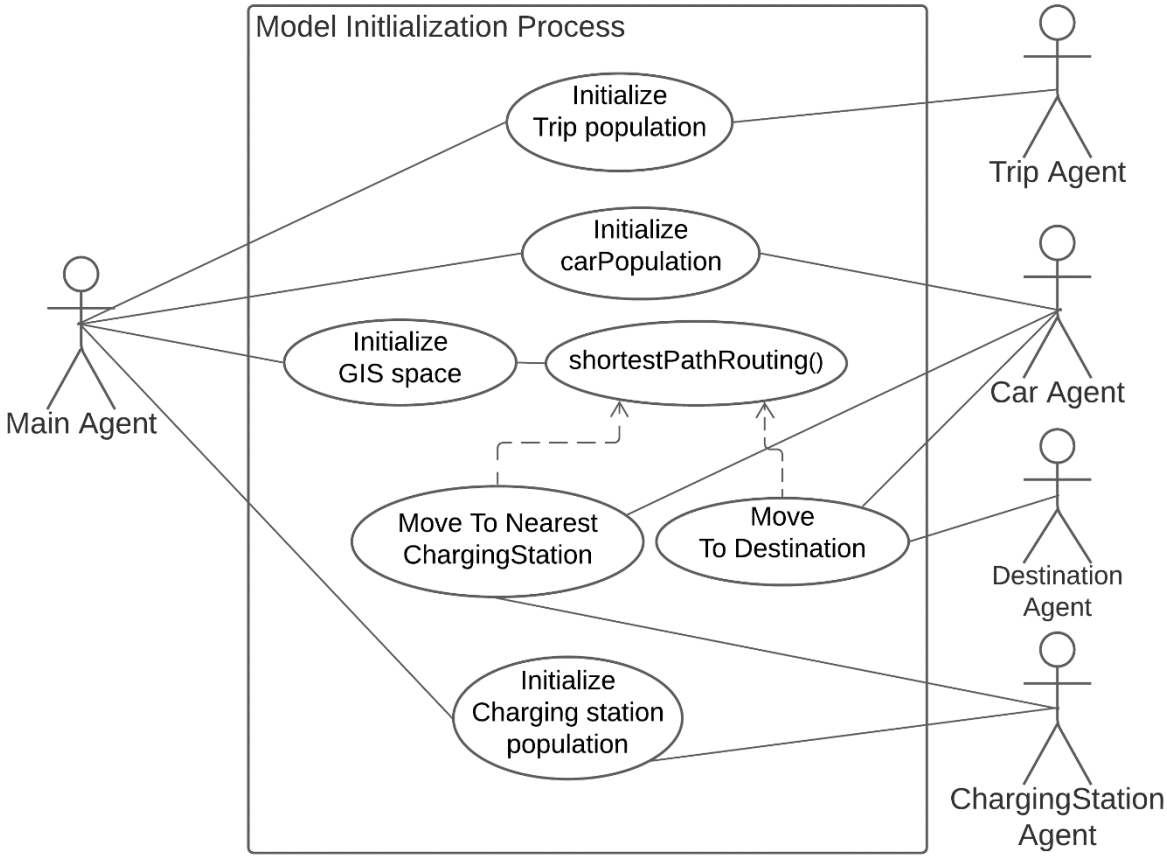

Figure 3.7 Main Agent Initialization Process Use Case Diagram

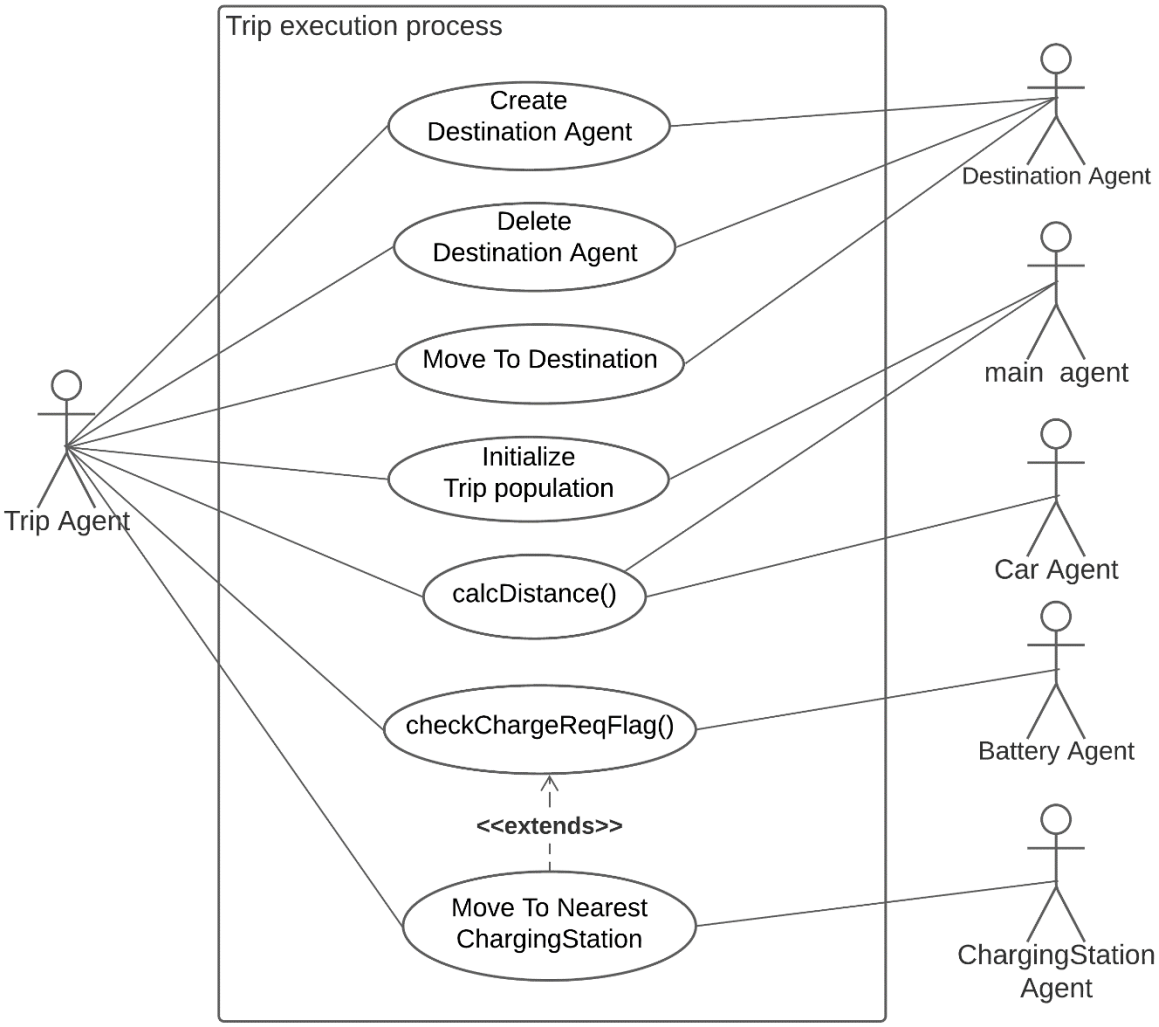

Figure 3.8 Trip Agent process use case diagram

Figure 3.9 Car Agent Process Use Case diagram

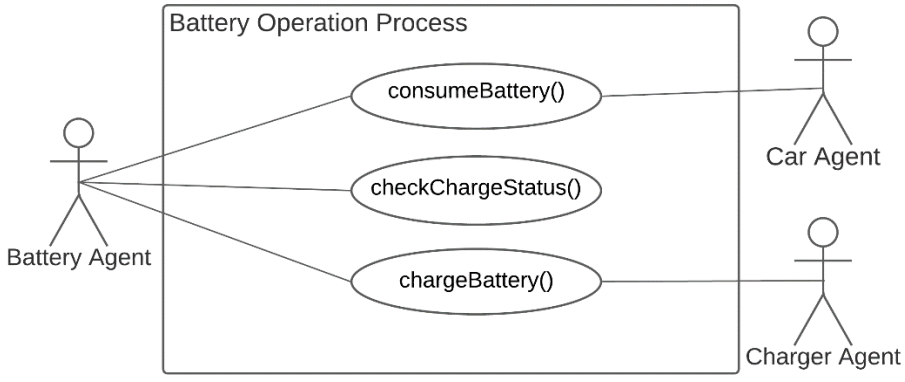

Figure 3.10 Battery Charging Process Use Case diagram

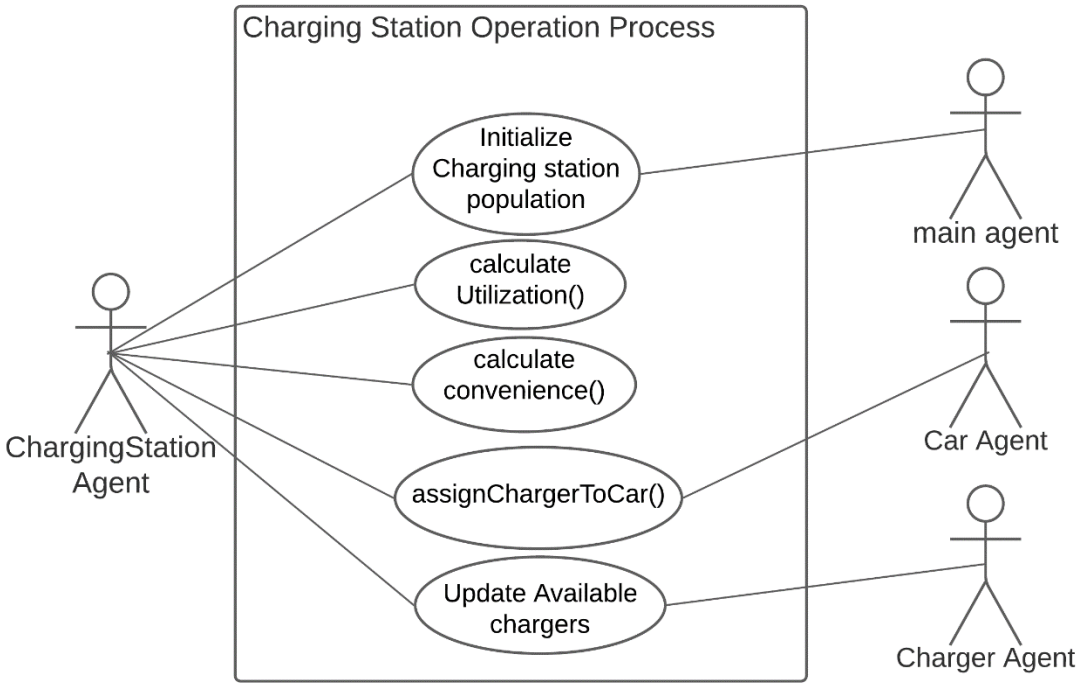

Figure 3.11 Charging Station Operation use case diagram

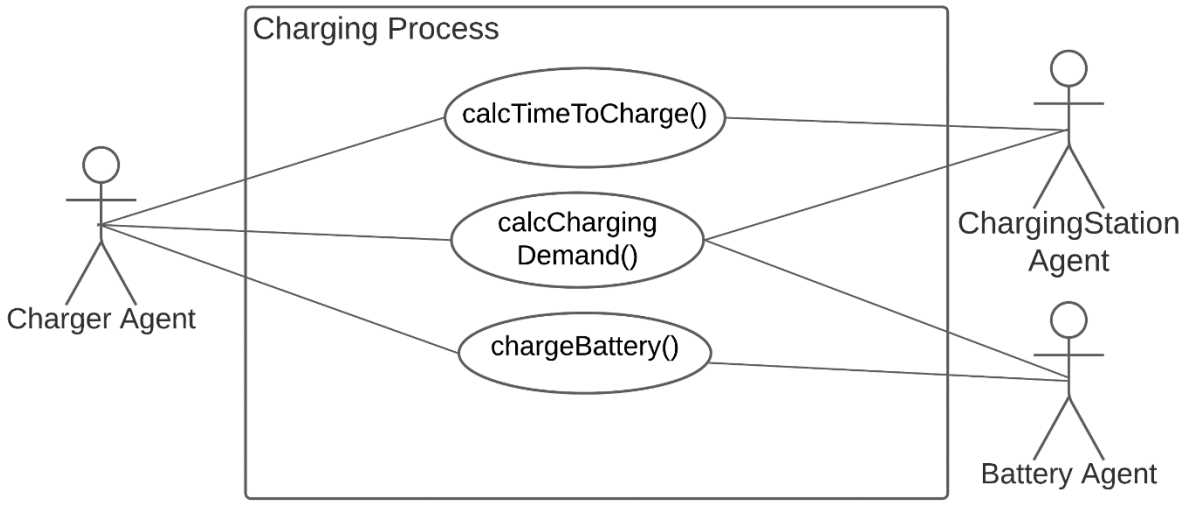

Figure 3.12 Charging Process Use Case diagram

3.5.4 Sequence Diagram

Sequence Diagram is the second most common UML diagram used by the developers and modelers, owing to its adaptability to multiple design strategies and clear representation of the process flow, incorporating all the major elements of the model/software. It shows the message exchange between objects/agents by delineating each of them in a vertical line. The messages between agents can be function calls, parameter exchange, or command messages and are represented by horizontal arrows. It can be noted that in the sequence diagram, time is represented along the vertical line progressing downwards and it can be both physical and logical.

The sequence diagram for agent-based modeling of LA problem of Charging Stations is shown in Figure 3.13. There are 6 agents involved in the process. The model starts from the MA, which initializes the population of TPA, CSA, and CA from their respective database values. Among these three main agents, only CSA has a static physical location in the GIS space. Hence CSA agents are placed in the GIS map environment permanently. TPA are then assigned with their corresponding CA according to their database values. Since TPA are an abstract agent that does not have a physical representation in real life, no symbolic icon or GIS location is assigned to them. CA on the other hand are mobile and is a physical entity in the GIS space, therefore, they are represented by a 'car' icon in the GIS map whenever they are actively participating in a TPA process flow.

After the initialization process, each TPA corresponds to one instance of a trip with an origin, destination, Time of Departure (ToD), and the instance of CA that should execute this trip. TPA stays idle till model time reaches ToD. At ToD TPA sends this instance of the trip as an object to the corresponding CA's trip-queue and instructs CA to check trip-queue. At CA if the 'current trip' parameter is empty, then it is made equal to this trip object and calls for process flow for trip execution in TPA. The TPA then places the CA at the origin, calculates the distance from the origin to destination, and checks the energy required to complete the trip by calling the 'batteryLeftWOCharging()' function from the CA, which returns the expected battery level of CA once the trip is completed. If the return value is less than 30% of total battery capacity, the 'chargeReqFlag' is set, indicating that the CA should recharge its battery first by visiting the nearest CSA before proceeding to its destination. Now at TPA, the process flow forks into two depending on whether 'chargeReqFlag' is true or false. If 'chargeReqFlag' is false, the TPA

instructs its CA to move to the destination point. If ‘chargeReqFlag’ is true, the TPA finds the nearest CSA from the available population of CSA, adds it to its CSA variable, and calculates the distance between origin to CSA. If the energy required to travel to CSA is higher than the current battery level, this instance of the trip is marked as infeasible and TPA exits the process flow. If the trip to CSA is feasible, then the CA is instructed to move to the assigned CSA. When CA reaches CSA, CA leaves the TPA process flow and joins the charging process flow in CSA. As the CA enters CSA, CSA finds an available Charger Agent (CGA) from its resource pool of chargers and assigns it to the CA. CSA calculates the Energy demand of CA and the time required to complete charging and pass it on to CGA. CGA interacts with the Battery Agent (BA) in the CA to execute the charging process for the required time. Once the charging process is done, CSA informs the CA and releases it to the TPA process flow. TPA updates the CSA convenience factor according to the reroute distance and increments the fitness factor by one to indicate the CSA utilization. TPA then instructs the CA to move towards its destination.

When CA reaches its destination, TPA updates its total distance run and looks for the next trip in its trip-queue. If there is a trip waiting in the queue, TPA instructs the CA to start the next trip and exits its process flow. If there is no trip pending, TPA sends the message ‘end’ to make CA return to its idle state and exits the process flow.

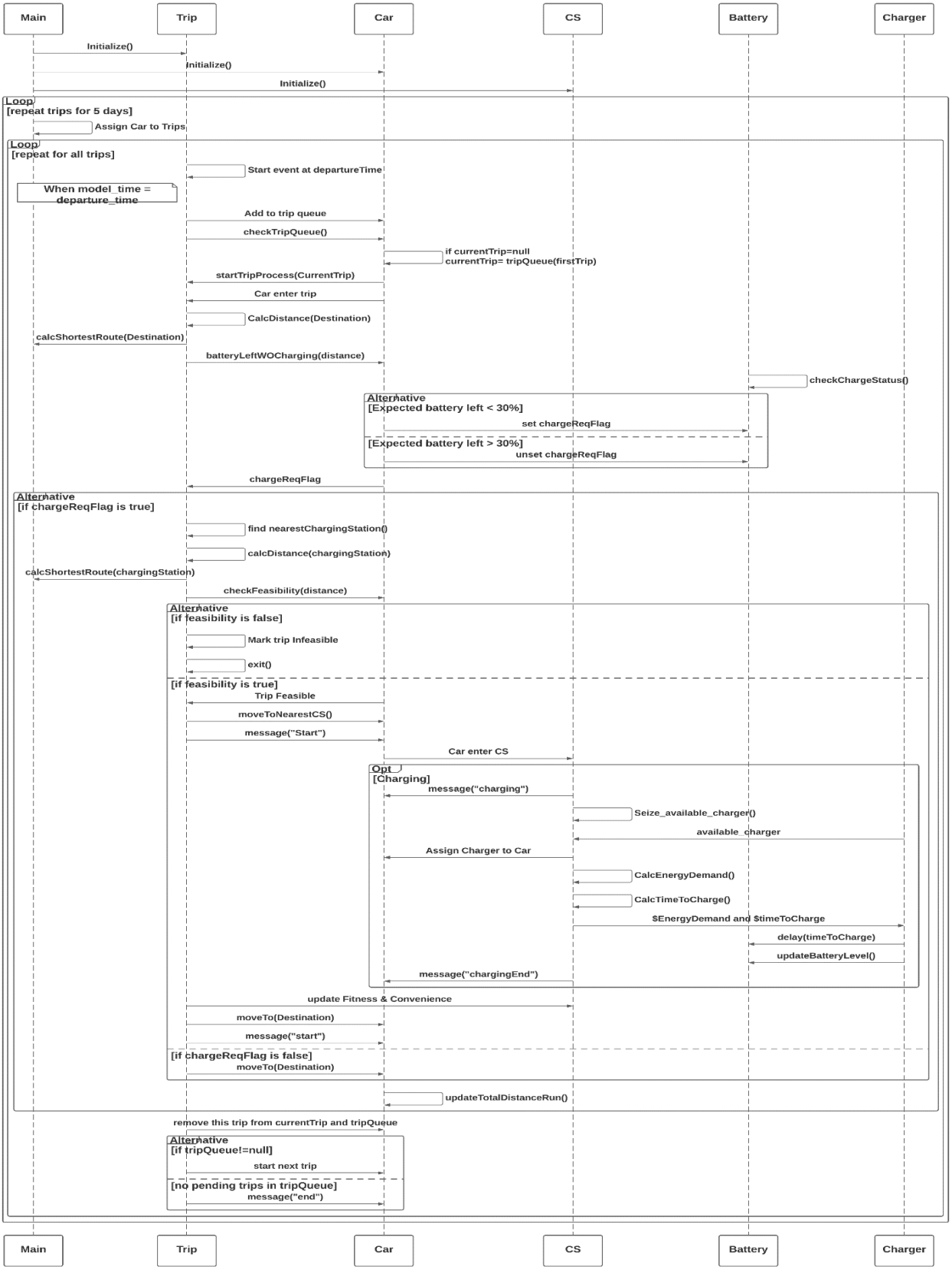

Figure 3.13 Sequence Diagram

3.6 Multi-objective optimization of ABM using Pareto Optimization

ABMs are widely used to simulate real-world systems. They act as a virtual laboratory where questions can be prompted and results analyzed, without the need for physical implementation and its associated costs and consequences. When the objective of simulation modeling is to determine the best combinations of input to achieve specific best results, they turn into an optimization problem. In general optimization problems refer to varying values of certain parameters to obtain the best values for an objective function. Hence, it should be noted that in this context, optimization refers to the optimal choice of a sequence of external inputs to achieve a specific goal[31]. This thesis makes use of the Pareto Optimization technique to arrive at the best set of Charging Station locations that will serve the EV user population of Montreal with the best convenience and minimum cost, while minimizing the number of infeasible trips. The objectives of convenience, cost, and infeasibility of trips are discussed in the next section.

One of the key factors of ABM models is its stochasticity. Hence, observations have to be made to examine how data changes over longer model times. In this thesis case, it was observed that most Car agents participating in the model had to go for at least one charging event within an average of 5 model days when started with an initial battery level, uniformly distributed between 35% to 75% of their battery capacity. Therefore, the model time required to run for optimization is determined as 5 days.

Scaling is an important concept in ABM simulation. Treating the original size and scope of the problem as true, it refers to the extent to which a model can be reduced without altering its pertinent dynamics. It is most often done to substantially reduce the run time of the ABM models and reduce the computational requirements for solving optimization problems. Thereby allowing access to a wider range of analytical tools to tackle the problem at hand[31]. For the purpose of optimal control, it is important to determine the extent to which models can be reduced while ensuring that the model is a faithful representation of the original problem. As the diversity in trips made by users is the most important in determining the significance of a charging station location, this thesis uses the total dataset provided for modeling. Significant reduction is made in fixing the model units to minutes instead of second and keeping model time accuracy to 10e-4. Both these changes result in only minor changes in model objectives calculation.

It is important to note that not all ABMs can be reduced by methods like Cohen's weighted 'k' method [32] and therefore, reduction of this model is very limited to the simulation controls used and the number of model days to be run.

3.6.1 Pareto Optimization

Heuristic methods are perhaps the most explored methods to optimize an ABM. Considering that in an ABM based LA problem, enumeration of solution space is infeasible, heuristic methods can be employed to perform a guided search through the solution space, to identify locally optimal controls. Several heuristic algorithms have been in use for ABM based optimization problems, including extensions of Simulated Annealing[33], Tabu Search[34], Squeaky Wheel optimization[35], Ant Colony Optimization, and Genetic Algorithm(GA)[36]. This thesis focuses on solving the optimization problem through a variation of GA called Simulation-based Multi-Objective Genetic Algorithm (SMOGA) to find the set of solutions for the LA problem through a Pareto optimal front based evolution.

Control problems with multiple objectives can be solved by assigning weights to each objective and treating the total of the weighted objectives as a single objective function. Single objective functions can then be solved using any appropriate method. Choosing the right amount of weights a priori becomes highly significant in this case and is up to the decision-maker to select the weights. Appropriate weights may be unknown to the decision-maker initially and multiple iterations of analysis may be required to fine-tune the weights to achieve a certain goal. It can be seen that valuable solutions and time may be lost in the process of choosing the right weights and repeating the experiment if the results are not satisfactory. Hence, treating the multi-objective problem in its original form takes advantage in choosing the desired solution from the set of solutions arrived after the analysis.

Pareto Optimization is a heuristic method that focuses on delivering multiple solutions, each with its own merits and demerits, instead of focusing on one set of solutions. Candidate solutions in the Pareto optimal front represent the solutions that are superior to the rest of the candidates in the set in at least one objective. Therefore, they cannot be improved upon an objective without sacrificing in another. Thus, giving the choice to the decision-maker to choose the right set of solutions from

the set, after the search is concluded[31]. The pseudo-code of the algorithm used is presented below in Algorithm 3.1:

- Generate initial population of charging stations (*location, n_fastChargers*) // from the dataset
- **while** *currentGeneration* < *maxGenerations* **do**
  - Run ABM and evaluate *fitness* factor and *convenience* factor for each -> *current_pop*
  - In *current_pop* determine Pareto frontier -> *current_frontier*
  - Add *current_frontier* to *next_pop*
  - **while** size(*next_pop*) < maxGenerationSize **do** // After the first iteration, reduce the maximum generation size by 5%
    - Choose 2 parents:
      - **Repeat**
        - *rand_set* = randomly select 5 solutions from *current_pop* //roulette wheel
        - **if** exactly one candidate is Pareto dominant over others **then**
          - Pareto dominant candidate becomes parent
        - **else if**
          - Dominant candidate with fewer neighbors in solution space becomes parent
            // give preference to isolated solutions (diversity)
        - **else**
          - Select a candidate at random to be parent
        - **end if**
      - **Until** 2 parents are chosen
    - Breed two new solutions A and B from parents:
      - **for all** parameters in parents **do**
        - Select parameter from random parent //roulette wheel
        - Add this parameter to *A*
        - Add corresponding parameter from other parent to B
      - **end for**
      - **set** *mutation_rate* = 0.20 * ((*maxGenerations* – *currentGeneration)/ maxGenerations)*
      - **for** n_fastCharger component in *A* and *B* **do**
        - change component with probability = *mutation_rate*
      - **end for**
    - Add *A* and *B* to *next_pop*
  - **end while**
  - **increase** *currentGeneration* by 1
- **end while**

Algorithm 3.1

The initial population is composed of 10 candidate solution sets (*chromosomes*), which are locations of Charging Stations represented through their ID number and the number of chargers in each. Each Charging Station can have 0, 2, 4 or 6 number of fast chargers. Having 0 number of chargers make any specific Charging Station inactive from the model run. So each *gene* in the candidate solution will have 2 parameters. 610 such genes would represent the total population of unique candidate locations for Charging Stations, both active and inactive, forming a candidate solution set or *chromosome.* The 10 *chromosomes* in the initial population are generated by the uniform distribution of the number of chargers among the entire solution set of charging stations. Figure 3.14 represents an example of the initial population of candidate solutions.

Candidate Solution 1

CS_ID_1	CS_ID_2	CS_ID_3	CS_ID_4	CS_ID_5	....	CS_ID_607	CS_ID_608	CS_ID_609	CS_ID_610
4	0	2	6	2	....	4	6	0	2

Candidate Solution 2

CS_ID_1	CS_ID_2	CS_ID_3	CS_ID_4	CS_ID_5	....	CS_ID_607	CS_ID_608	CS_ID_609	CS_ID_610
2	6	4	6	0	....	0	4	0	6

.
.
.

Candidate Solution 9

CS_ID_1	CS_ID_2	CS_ID_3	CS_ID_4	CS_ID_5	....	CS_ID_607	CS_ID_608	CS_ID_609	CS_ID_610
6	0	0	2	4	....	6	2	0	4

Candidate Solution 10

CS_ID_1	CS_ID_2	CS_ID_3	CS_ID_4	CS_ID_5	....	CS_ID_607	CS_ID_608	CS_ID_609	CS_ID_610
0	2	6	4	2	....	2	0	6	4

Figure 3.14 Gene representation of candidate solutions

By inputting the candidate solutions one by one to the ABM model and simulating them over the desired period of 5 model days, the fitness and convenience factor of each CS is calculated and the combined data is considered as the current population. Example of the current population representation is shown in Figure 3.15. This current population is searched for Pareto solutions and they are selected to be added straight to the next population set. In the next step, 5 candidate parents

are selected at random through roulette wheel randomization of their fitness factor. Giving preference to Pareto dominant solutions and then diversity among the 5 solutions, 2 parents are chosen to crossover. The crossover mechanism produces two new solutions A and B, by randomly exchanging the number of chargers parameter among the two parents using a 4 point crossover.

Candidate Solution 1

	CS_ID_1	CS_ID_2	CS_ID_3	CS_ID_4	CS_ID_5	....	CS_ID_607	CS_ID_608	CS_ID_609	CS_ID_610
Num_chargers	4	0	2	6	2	....	4	6	0	2
fitness	1.25	0	0	0.333	1.0		0.882	0.866	0	0
convenience	3.561	0	0	2.378	0.59		1.258	0.897	0	0

Candidate Solution 2

	CS_ID_1	CS_ID_2	CS_ID_3	CS_ID_4	CS_ID_5	....	CS_ID_607	CS_ID_608	CS_ID_609	CS_ID_610
Num_chargers	2	6	4	6	0	....	0	4	0	6
fitness	1.4	0.5	0.524	0.166	0	....	0	0.75	0	0.666
convenience	1.205	2.780	0.841	2.413	0	....	0	0.207	0	1.215

.
.
.

Candidate Solution 10

	CS_ID_1	CS_ID_2	CS_ID_3	CS_ID_4	CS_ID_5	....	CS_ID_607	CS_ID_608	CS_ID_609	CS_ID_610
Num_chargers	0	2	6	4	2	....	2	0	6	4
fitness	0	0.8	0.166	0.25	0	....	0.78	0	0.333	0.8
convenience	0	2.780	0.841	0.849	0	....	2.158	0	2.543	1.485

Figure 3.15 Population representation

The crossover points are determined randomly from 1 to 609. The solutions formed A and B are then subjected to mutation with a probability of 0.2 to randomly change the number of chargers in 20% of the Charging Stations. The mutated child solutions A and B are then added to the next population. This process of choosing parents and producing child solutions is repeated till 10 solutions are obtained for the next population set. Once the number of solutions in the next population reaches the population size. Each time a new population is generated the steps are repeated from simulating the model with each new solution set and calculating their fitness and convenience factors until there is no improvement.

The simulation model and results from this optimization experiment are discussed in the next chapter.

Chapter 4

Simulation Model Implementation and Results

4.1 Introduction

In this Chapter, the Agent-Based Modeling (ABM) of Electric Vehicle (EV) mobility through city of Montreal is implemented and discussed using Anylogic 8.6 simulation software, with emphasis on the UML models discussed in section 3.5. All the constraints and the assumptions detailed in section 3.4 are considered while constructing the simulation model. The agents in the model are created according to the Class diagram designed in section 3.5.1 and initialized with the initial data. The dynamic elements of the model design and the optimization experiment is discussed in the next sections.

4.2 Elements of simulation model and concepts

Agent based models can represent a complex system with high level of detail, due to its inherent encapsulated behavior-based design, which incorporates inheritance properties. Therefore, Objected Oriented Programming (OOP) languages can best represent them. Among the wide variety of OOP language available, Java tends to be the most popular in ABM implementation due to its compatibility to simulation based modeling and wide acceptance. As a result, Anylogic simulation software is also built on a Java development platform and provides features such as abstraction, polymorphism, encapsulation and inheritance. Similar to Java classes, Agents are implemented as Agent Classes in Anylogic. Any agent class may contain parameters and functions that define their characteristics and behaviors, state charts and process-flows that define their decision-making process and connections to other classes that interact with them.

While ABM is great at implementing the natural flow of processes from the perspective of individual objects or agents, the decision-making processes of the larger system may be the best represented by a Discrete Event (DE) process flow. DE modeling is the most efficient when the behavior of system under discussion is a sequence of operations, such as Trip planning operations in this case. As Anylogic supports the combination of both modeling strategies in a single platform, both process flow charts and statecharts are used to model the system in this thesis.

To better understand the process of ABM using Anylogic software in this thesis context, some important concepts are discussed in the next sections.

4.2.1 Anylogic Modeling elements

4.2.1.1 Statechart

A statechart in Anylogic is an extended version of UML state diagram. It provides a visual design element to define the event and time dependent behaviors of agents in the simulation. Though it is majorly used in ABMs, it works well with the process flow based and system dynamics based models. Statecharts are composed of states and transitions. According to A.Borshchev "A state can be considered as a 'concentrated history' of the agent and also as a set of reactions to external events that determine the agent's future. The reactions in a particular style are defined by transitions exiting that state. Each transition has a trigger, such as a message arrival, a condition, a timeout, or the agent arrival to the destination. When a transition is taken ('fired'), the state may change and a new set of reactions may become active. State transition is atomic and instantaneous"(37).

Anylogic supports a version of UML state diagram or state machine that supports composite states (states that contains other states), history states, transition branching and internal transitions. While orthogonal states are not supported, it is possible to define multiple statecharts in the same agent to work in parallel.

4.2.1.2 Flowchart

Flowchart is a widely adopted graphical representation of processes. A process flowchart in Anylogic combines the properties of UML sequence diagram and activity diagram. As the trip planning and execution is a sequence of operations that controls the Car Agent and Charging Station Agent states, the Trip Planner Agent is designed as a Discrete Event (DE) process flow. A process flow is a graphical representation of processes that the entities of simulation is subjected to. In this thesis context, the entities are agents: Car Agents and Charging Station Agents, which interact with the process flow in the Trip Planner Agent. Use of process flowcharts enable us to accurately represent the trip planning decisions than using a state chart to do the same.

4.2.1.3 Agent parameters

In ABMs, parameters represent the inherent characteristics of the agent or object. For a large class of agents, like Car Agents, it is their parameters that distinguish from one to another. Agent parameters are defined according to the design provided by the class diagram of model design. According to Anylogic, a parameter is used to describe the static characteristics of an agent or an object. It is normally considered constant over a complete simulation and is changed to change the behavior or outcome of the simulation by the modeler. But it can also be noted that all parameters' values are visible and changeable dynamically throughout the simulation model and hence, can be updated during runtime to adjust the model.

In the next section, model design implementation of each major agent is discussed.

4.2.2 Main Agent Modeling

Main Agent (MA) is where all the agents in the model interact (Figure 4.1) and it is the first agent that is initialized during model execution. It is comprised of a GIS space where the CSAs are placed in their respective longitude and latitude and the CAs move from origin to destination. The function *f_OnStartup* initializes the TPAs, CAs and CSAs. Destination Agents (DAs) are populated dynamically during model execution.

The static parameters of MA include

1. **kWh_PerKilometer,** which decides the battery charge consumption of EVs per distance run
2. **p_batteryLevelThreshold_ForCharging,** which decides the minimum expected battery charge percentage level required after a trip is complete, so that there is enough battery charge left for the next trip or to visit a charging station before the next trip.
3. **carSpeed** provides the average speed of movement of cars through the GIS space.

The dynamic parameters of MA are

1. **current_Population_CS** holds the set of solutions that belong to the current generation of solutions and is initialized by the optimization algorithm. It is of type Java class type *Population* and holds solutions as an array of Java class *Candidates* both discussed in section 4.2.
2. **candidate_num** depicts the individual solution that is assessed by the model at any moment. It acts as an index number to a single set of charging stations and their capacities from the current_Population_CS for model execution.

Variables **num_infeasibleTrips** and **totalChargingEvents** are dynamically populated during runtime.

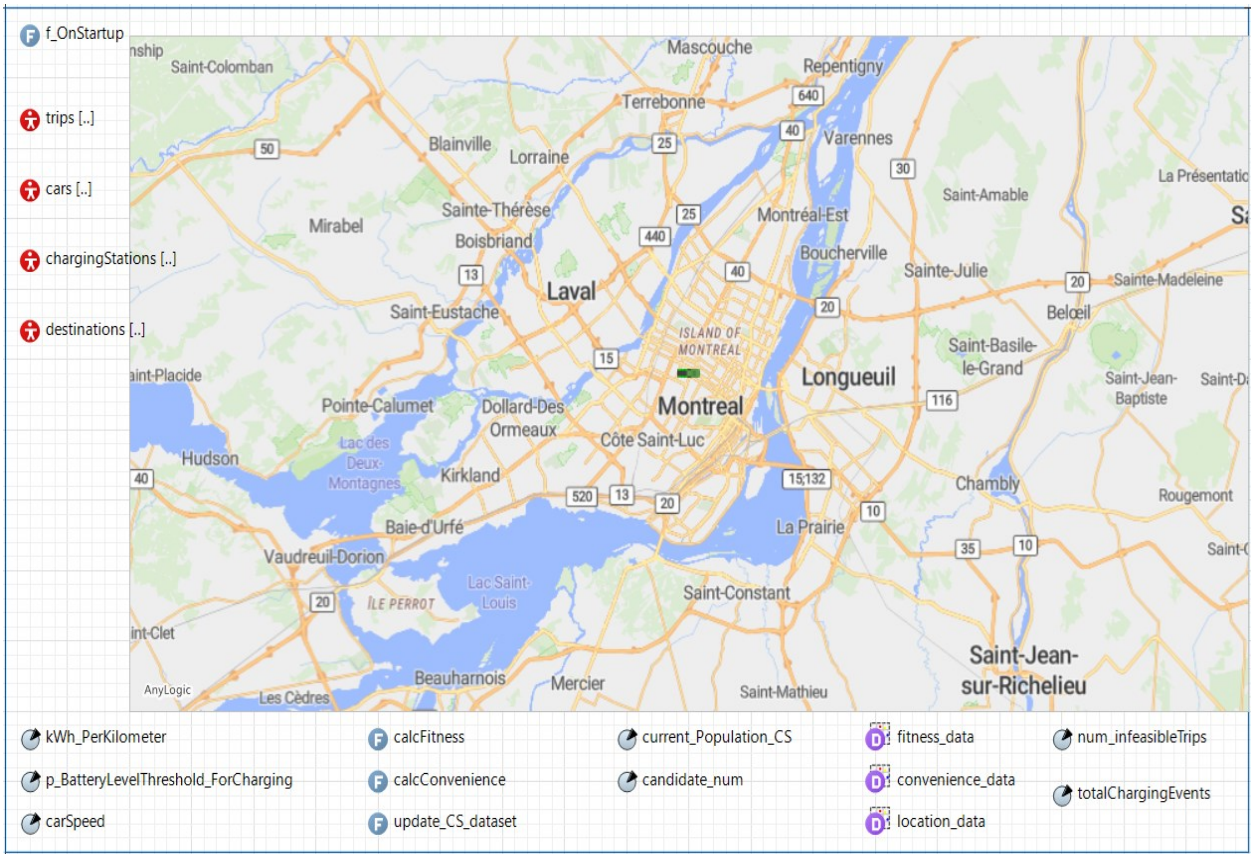

Figure 4.1 Main Agent model

Function **update_CS_dataset** populated the final values of fitness, convenience and locations of each charging station from the solution set to the data sets **fitness_data, convenience_data** and **location_data.** Functions **calcFitness** and **calcConvenience** are used to calculate the fitness factor and convenience factor of a candidate solution after each run from the dataset of individual fitness and convenience.

4.2.3 Trip Planner Agent Modeling

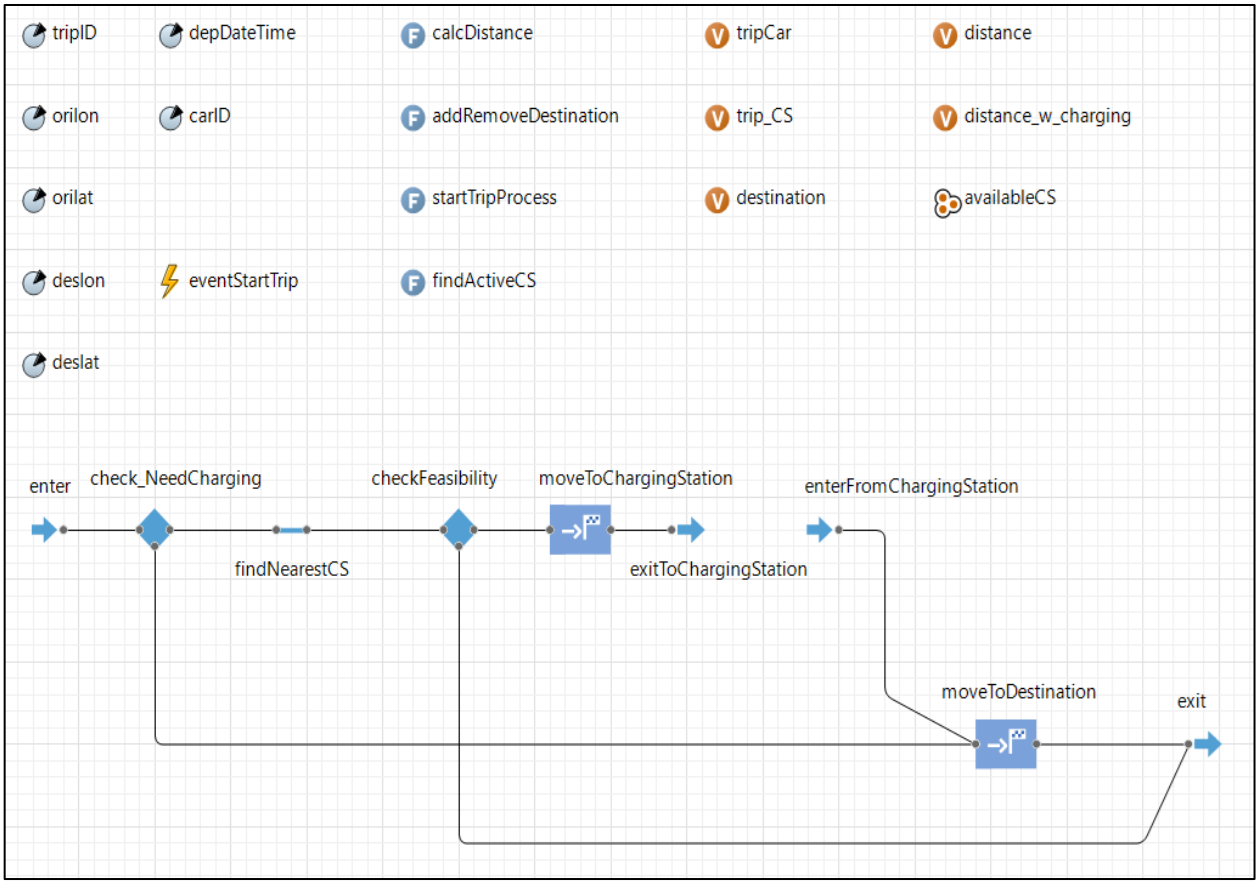

Figure 4.2 Trip Planner Agent Model

Trip Planner Agent (TPA) is the starting point of the model processes (Figure 4.2). Trip Agent parameters are initialized from the trips database, which is discussed in detail in Section 3.3. Each TPA is responsible for a single instance of trip (movement from origin to destination). The TPA acts as the decision maker of this ABM, by directing the states of both Car Agents (CA), Charging Station Agents (CSA) and their corresponding extensions, Battery Agent (BA) and Charger Agent (CA). Trip agent is defined by its parameters as follows:

1. tripID – unique value representing each trip
2. orilon – origin location longitude value
3. orilat – origin location latitude value
4. deslon – destination location longitude value
5. deslat – destination location latitude value
6. depDateTime – the time of day at which this trip should start
7. carID – unique carID that corresponds to the person/vehicle this trip belongs to

Apart from parameters, the TPA also has variables that holds dynamic values/objects during the run time. All variables are initially null or zero and are assigned values during runtime. The variables used in TPA are as follows:

1. tripCar – Car Agent reference object to corresponding carID
2. trip_CS – Charging Station Agent reference object to the selected Charging Station
3. destination – destination Agent (GIS location entity)
4. distance – distance between origin to destination without charging reroute
5. distance_w_charging – distance between origin to destination with charging reroute
6. availableCS – collection of Charging Stations that has charger slots currently unused

The process flow is initiated using an event element ***eventStartTrip*** that triggers the Trip process flow only at the specified departure time from ***depDateTime*** parameter and repeats the trigger in one day interval. Once triggered the *eventStartTrip* checks if the Car Agent is completing another trip at the moment and adds this trip to its queue. If the Car Agent is idle, then it initiates the Trip Process flowchart by passing the *tripCar* to 'enter' block of the flowchart. The flow chart then executes the process flow designed in section 3.5.2.1 as a discrete event. Functions *calcDistance*, *addRemoveDestination*, *startTripProcess* and *findActiveCS* used appropriately through the flowchart depicts the behaviors of the TPA. Underlying java code that defines each of these functions are provided in Appendices.

4.2.4 Car Agent modeling

CAs are the mobile agents in this model and their characteristics are defined by a statechart present inside the agent. The composite state of 'Car Active' from section 3.5.2.2 is modeled into the CA state 'moving', while keeping the 'charging' state outside the composite state. The state transitions are triggered through messages from the TPA, which operates this CA at any moment and the looping back state transition on the 'moving' state is triggered per minute to reduce *batteryLevel* as the CA moves through the GIS space (Figure 4.3).

The parameters that define a CA are as follows:

1. carID – unique identifier of an individual CA
2. batteryCap – battery Capacity of this CA
3. batteryLevel – current battery level of this CA

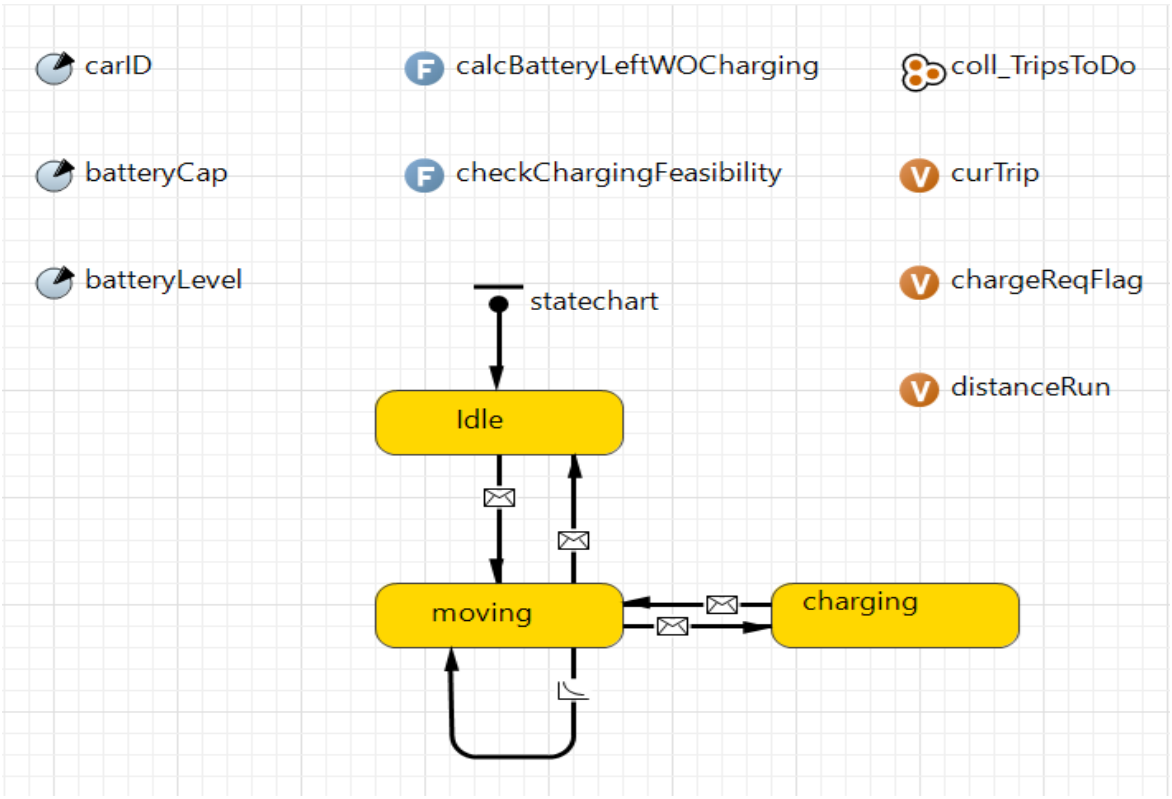

Figure 4.3 Car Agent model

Variables used in the CA are as follows :

1. coll_TripsToDo – It is a array list of TPAs that are waiting to be executed by the CA, as the CA completes an earlier trip. This feature is included to make sure that the CAs are not forced to be part of two TPA process flows simultaneously.
2. curTrip – holds the currently in progress trip that utilizes this CA
3. chargeReqFlag – if true, indicates that the battery charge level is low to make another trip and forces the TPA to direct the CA for re-charging
4. distanceRun – holds the total distance travelled by the CA. It is populated by the TPA.

Function **calcBatteryLeftWOCharging** calculates the energy demand(charge required) of the trip and sets the **chargeReqFlag** to true, if the expected battery charge level after the trip would be less than *p_batteryLevelThreshold_ForCharging.*

Function **checkChargingFeasibility** checks if the CA has enough energy to reach the nearest CSA to recharge before continuing to the destination and instructs the TPA to mark the trip as infeasible if there is not enough energy left to make the trip. Underlying java code that defines each of these functions are provided in Appendices.

4.2.5 Battery Agent modeling

BA is modeled as an extension of CA and its behavior is dictated by the state transition messages triggered from state transitions from the CA. BA keeps track of the battery charge level in each idle, moving and charging state and updates the CA battery charge level, accordingly (Figure 4.4).

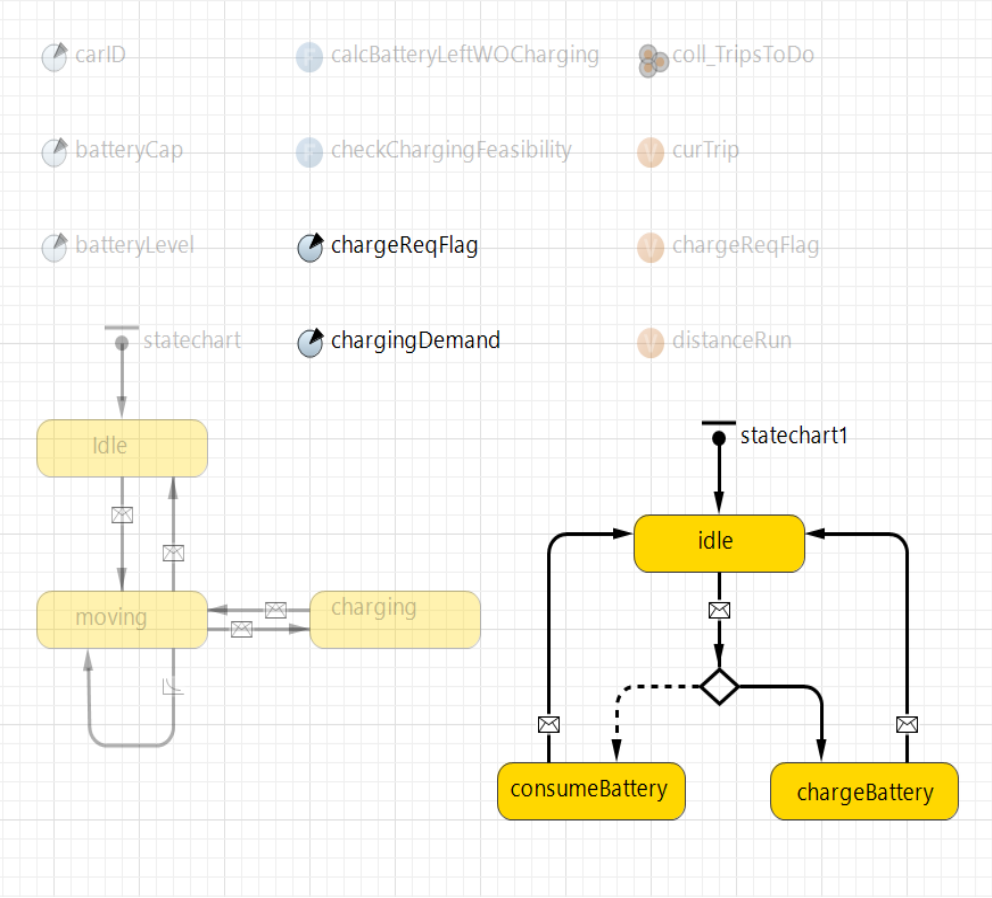

Figure 4.4 Battery Agent model

4.2.6 Charging Station Agent modeling

CSA (Figure 4.5) is initialized by the optimization experiment algorithm by randomly assigning number of chargers to each candidate Charging Station location. However, the *chargingLevel* parameter of CSA is initialized from the MA. CSA is modeled using the process flow chart block that enables the modeler to include functionalities of seize the resource (CA), execute charging action and update the fitness and convenience value together into on 'service' block in Anylogic.

Function ***calcChargingTime*** is responsible to calculate the energy demand of the CA that arrives at the CSA and determine the charging time required. Since this thesis aims to make EV recharging process as close to a gas station refueling process, it is assumed that any EV user would not want

to spend more than 20 minutes at the charging station and hence, caps the ***chargingTime*** to 20 minutes.

Function ***updateAvailableChargers*** keeps track of the FCS capacity, by monitoring the expected number of cars to arrive at the station, current queue of cars waiting for charging and number of available chargers. It sets the *chargers_available* according to the formula below as a measure of how many more cars it can accommodate at that moment.

```
chargers_available = (number of idle chargers) + (queue capacity)
                   - (current queue size) - (number of incoming cars)
```

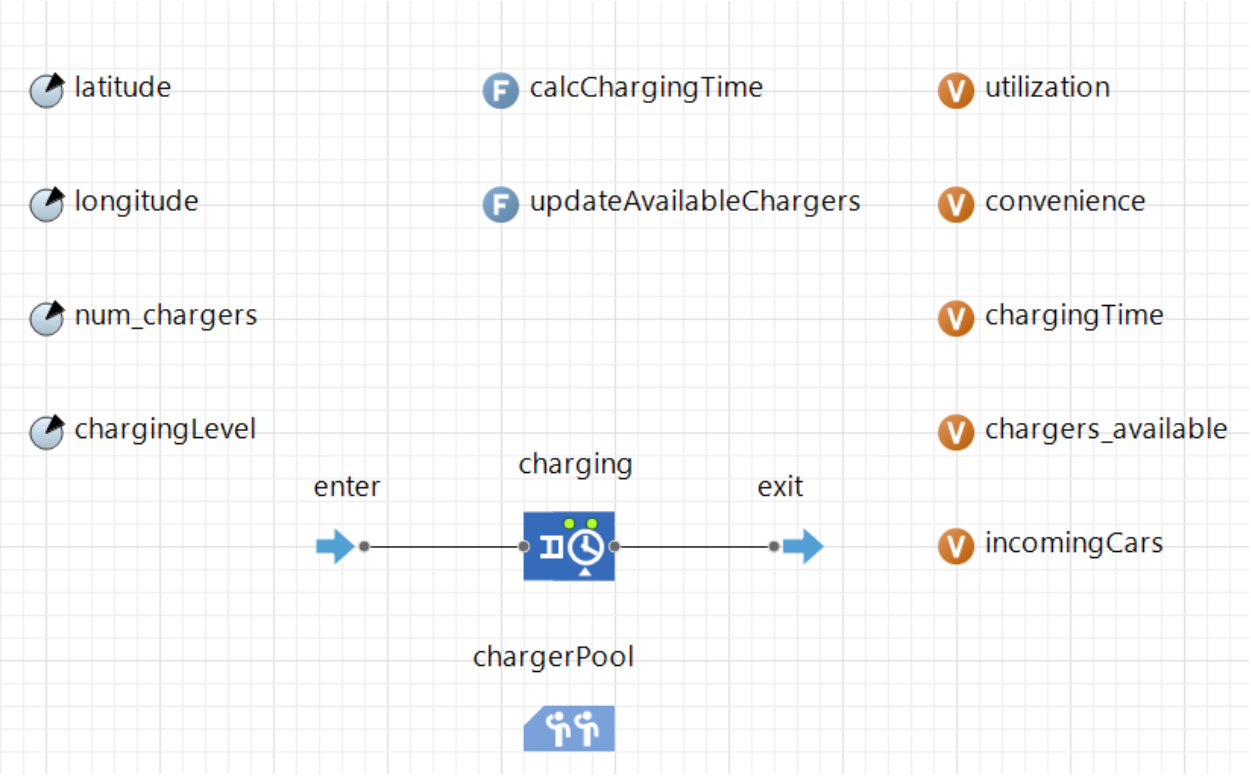

Figure 4.5 Charging Station model

4.2.7 Charger Agent modeling

Charger agent is modeled as an extension to the Charging station agent with parameters *chargingSpeed,* which provides its peak charging rate and *busyFlag*, which if set true initiates the Charger statechart as defined in section 3.5.2.5

4.3 Optimization experiment

The optimization process to find the best solutions of FCS locations and their capacities is implemented using Java classes and custom experiments developed specifically for this model. The design of optimization experiment follows algorithm 3.1, with the use of separate java classes for Candidate solution, population of solutions in a given generation and Genetic Algorithm (GA) implementor class to aggregate the evolutionary behaviors of GA. The individual components of this process are discussed in the coming sections in the order of lowest level of encapsulation.

4.3.1 Candidate Java class

A candidate solution is a set of charging stations mapped to its exact location through the database, whose number of chargers are determined by a roulette wheel randomizer. The number of chargers can be 0,2,4,6 or 8 in each of the FCS. Any FCS with zero number of chargers would indicate that this particular charging station is not in service and hence, not a part of the solution. Apart from the number of chargers, other parameters of each candidate solution set include a candidate id, fitness factor, convenience factor, total number of infeasible trips and number of neighbors (other candidate solutions with similar objective values).

Java code implementation of this class is provided in appendices for further detail.

4.3.2 Population Java class

A population is a collection of Candidate solutions that forms a generation of solutions. The population class implements the characteristic behaviors of the population such as initialize population, check for dominance between two candidate solutions, finding Pareto front candidates out of the solution, find neighbors of each candidate solution etc.

The java code implementation of this class is provided in appendices for further detail.

4.3.3 Multi-objective Genetic Algorithm (MultiObjGA) Java class

This class incorporates the Candidate and Population class. It performs the evolutionary Multi objective optimization process by implementing the procedures from algorithm 3.1. It includes population crossover (process of selecting parent candidates and producing child candidates) and mutating a population (introducing random parameters in a new child population to increase exploration of search space).

The java code implementation of this class is provided in appendices for further detail.

4.3.4 Optimization experiment driver Java code

Custom experiment is a feature of Anylogic that lets the modeler write custom Java code that can modify, execute and iterate the simulation model. In this thesis, the ABM simulation implemented in Anylogic following modeling procedure in section 4.1 is subjected to iterative optimization using the custom experiment.

Each generation of solutions, i.e each 'Population' is evaluated by running the simulation model with each Candidate solution in the population to determine its objective values, i.e., average utilization of chargers, average distance of reroute to re-charge and number of infeasible trips. Once an entire generation of candidate solutions are evaluated, that population is subjected to the evolution. Evolution is composed of crossover of population to generate new candidate solutions and mutating them with a pre-determined probability to introduce randomness to the solution.

The java code implementation of this class is provided in appendices for further detail.

4.4 Model execution (Runtime)

In this section the model execution at runtime is described step by step. The complete list of Main Agent parameters and their corresponding values is given in Table 4.1.

Table 4-1 Runtime Parameter values

Parameter	Value
kWh_perKilometer	0.161 kWh/km
P_BatteryLevelThreshold_ForCharging	0.35 => 35% of battery capacity
carSpeed	40kmph
current_Population_CS	null
candidate_num	0
chargingSpeed	120kW
Population of Destination	Initially null
Population of Trips	Initialized from database
Population of Cars	Initialized from database
Population of Charging Stations	Initialized from database

TPA and CA are initialized from database, with constant random values of battery capacity and initial battery level for Car Agents.

The simulation model is initially run for the required amount of model time, which is 5 days and 2 hours to model the traffic of a typical working weekdays. The additional 2 hours are allocated to let all vehicles who started their journey close to the midnight to reach their destinations. This is done using routes fetched from network, which allows caching of routing data to enable faster run times in the subsequent runs. For this purpose, all candidate charging station locations are enabled and the simulation is run with the complete set of trips data (Figure 4.6 Simulation model runtime view).

Once the required number of routes are learned and stored in cache, CSA is initialized through roulette wheel randomizer from Candidate java class and optimization experiment is conducted iteratively. The pareto frontier solutions from each generation are sorted for the best solutions and a maximum of half the population size of pareto front solutions are carried to the next generation. At the end of each generation, a generation output file complete details of all candidates from the

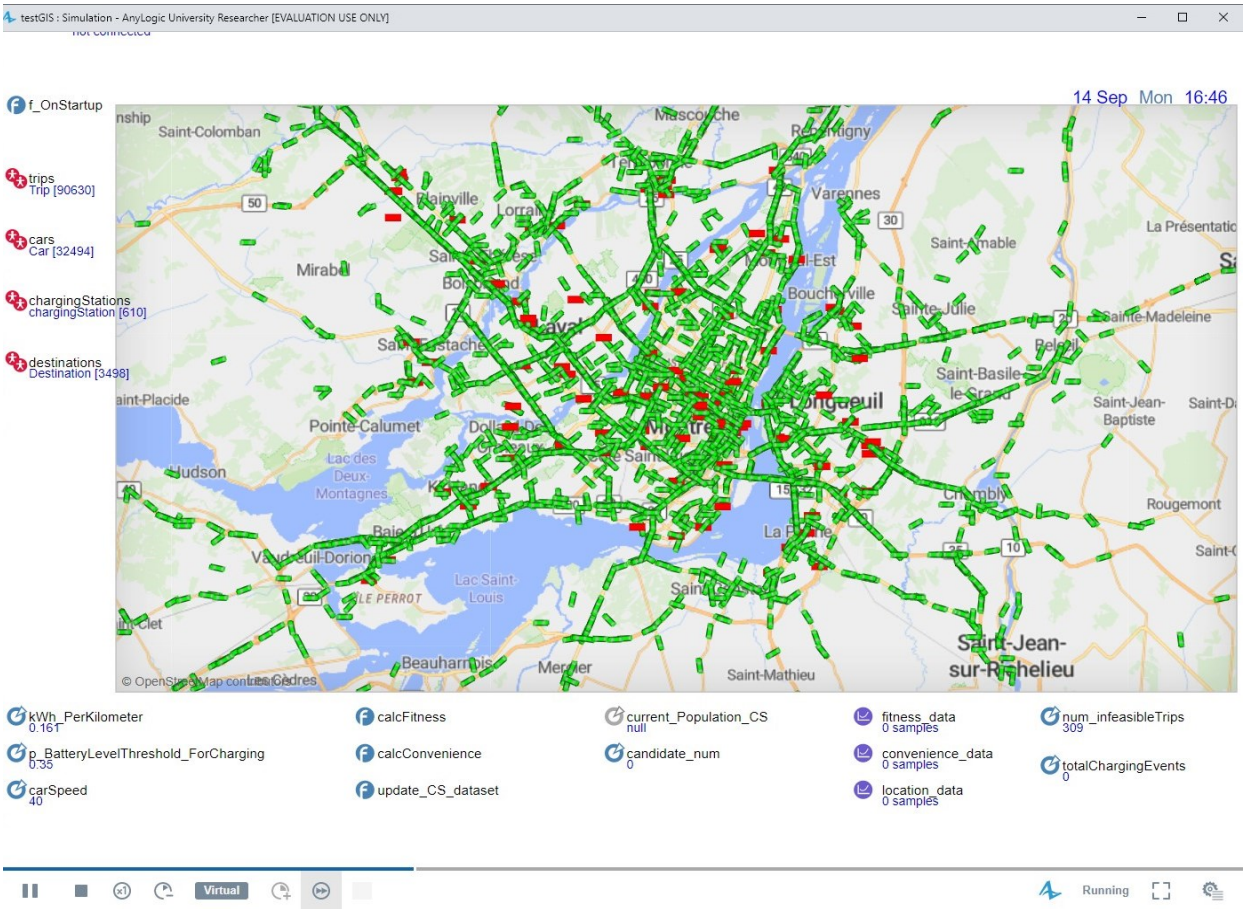

Figure 4.6 Simulation model runtime view

solution is generated and saved. A summary of the fitness, convenience and trip-drop factor of each Candidate solution is saved to another file for complete summary.

4.5 Results of Optimization experiment model

Matlab 2019R is used to plot the summary of results from the optimization experiment in an isometric plot (Figure 4.7). The axis for the Fitness factor is reversed to negative direction so as to comprehend the plot easily. With the reversed direction of the fitness factor, the direction of optimization is towards the negative direction in each axis. Hence the solution points plotted towards the origin of the cube are better than the points farther away from the origin.

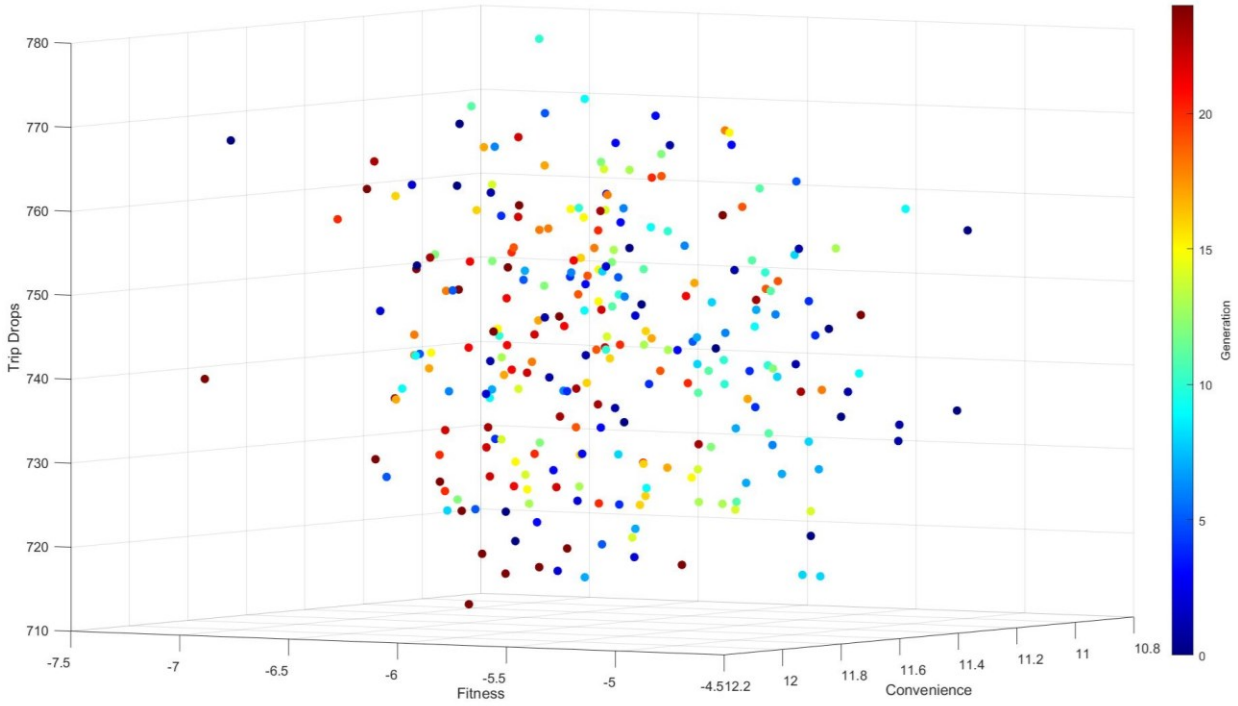

Figure 4.7 Optimized solutions Isometric view

The generation of solutions is distinguished by the color of the dots as given by the legend. It can be seen that as the model proceeds to higher generation, the solution converges towards the axes and forms a boundary of pareto front. It may be noticed that a few lower generation solutions are also close to the pareto front. This occurs due to the limit set on the number of pareto front solutions that get to be forwarded to the next generation, causing some good solutions to be lost in selection.

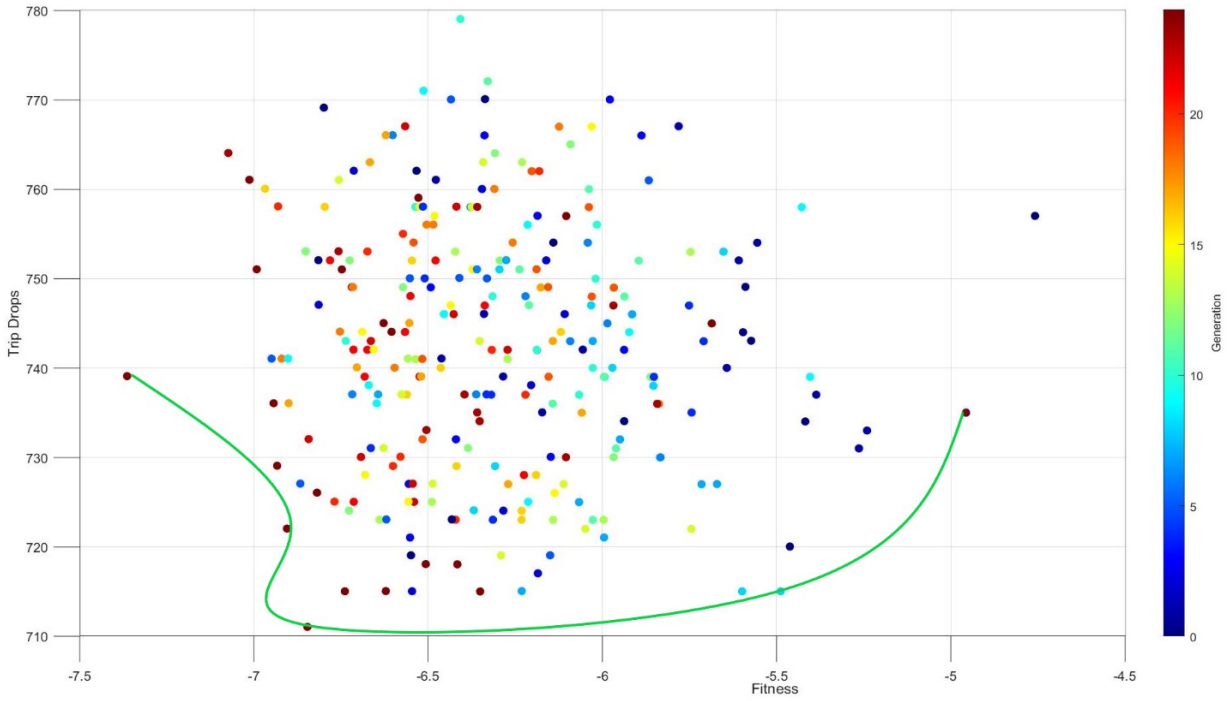

Figure 4.8 2-Dimensional view of TripDrops VS Fitness Factor

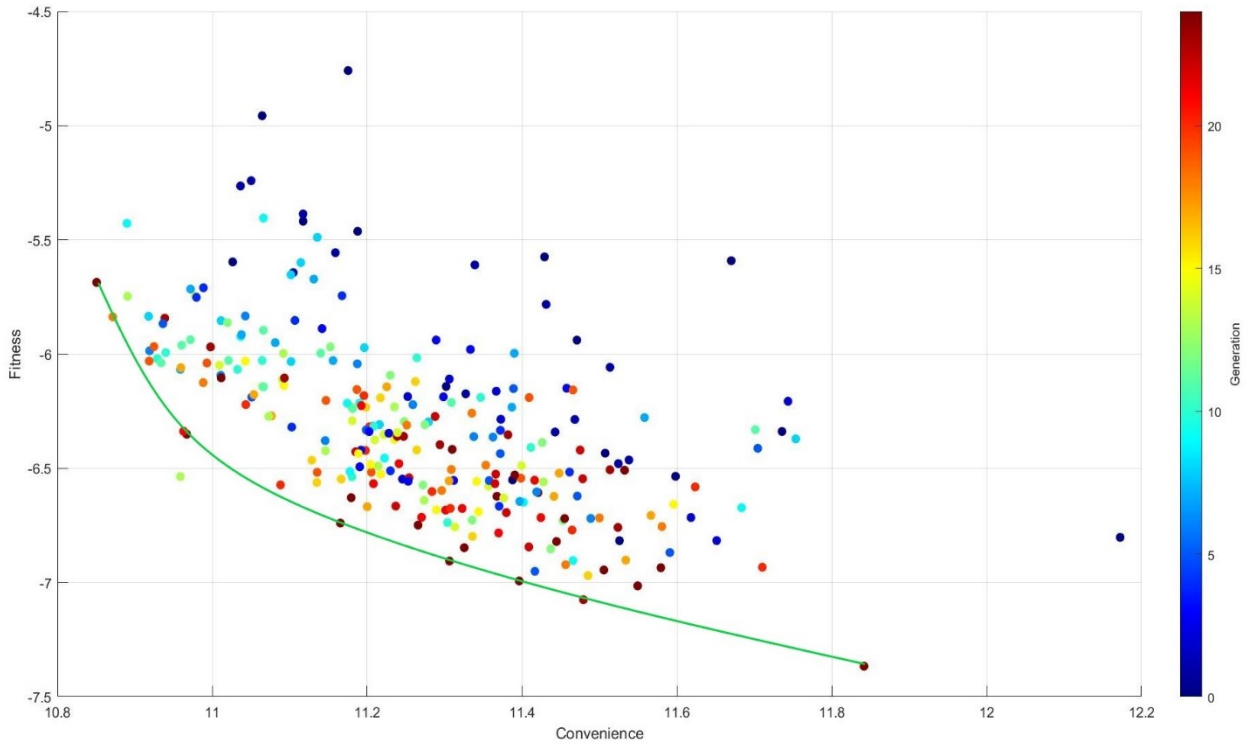

Figure 4.9 2-dimensional view of Fitness VS Convenience factor

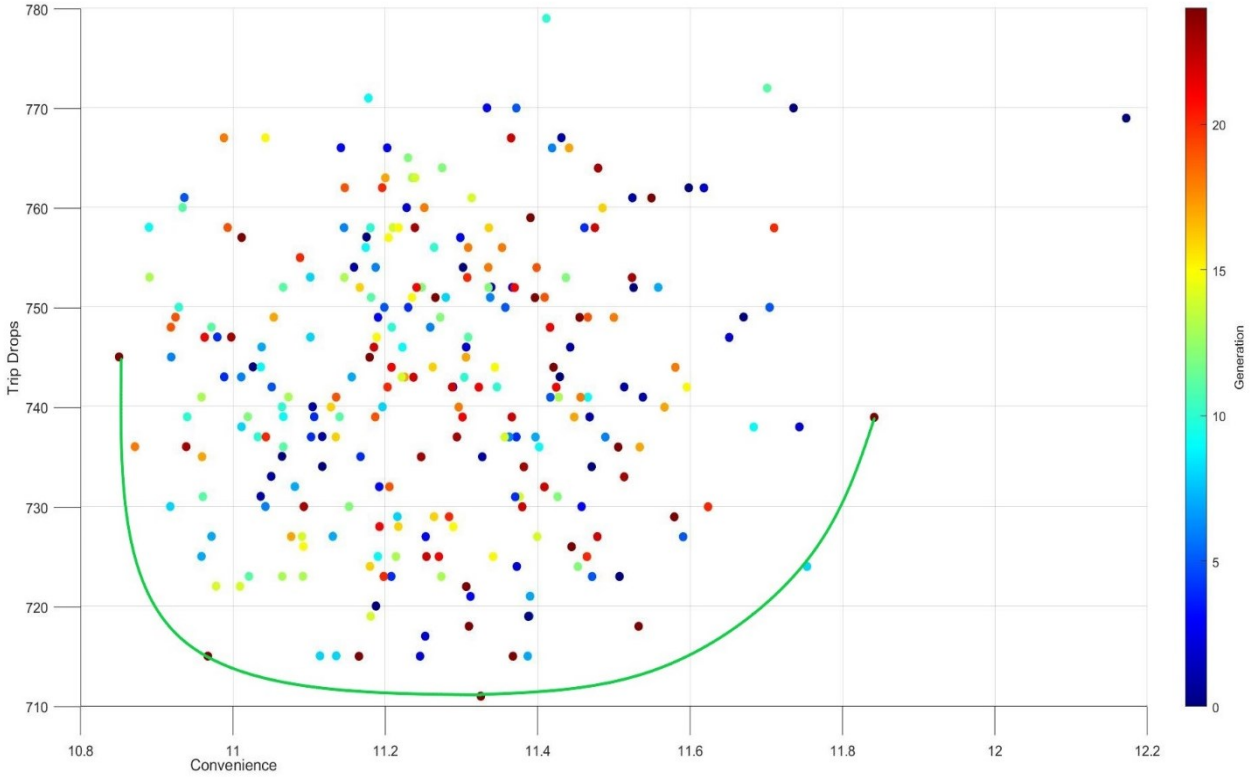

Figure 4.10 2-Dimensional view of Trip Drops VS Convenience factor

The optimality of the last generation solutions can be visualized by viewing the isometric plot from either of the planes Trip Drops Vs Fitness or Fitness vs Convenience Figure 4.8, Figure 4.9 and Figure 4.10 gives the 2-dimensional plots between 2 objective values. The pareto front is marked by connecting the solution points from the last generation by a bright green line. It can be observed that the higher generation points dominate the boundary of the pareto front and give the best trade offs between the two objective values each plot.

From the final generation of solutions, 3 candidate solutions with the best individual fitness values can be plotted. As the pareto front retains the best feasible solutions from each objective value, the decision maker can choose from any available solution present in the final pareto front to decide the optimum trade-off between the 3 objective values, as per the requirement.

Figure 4.11 provides the solution set of charging station locations with their respective number of chargers for the most optimal trip drops factor. Implementing this solution predicts only 711 trips

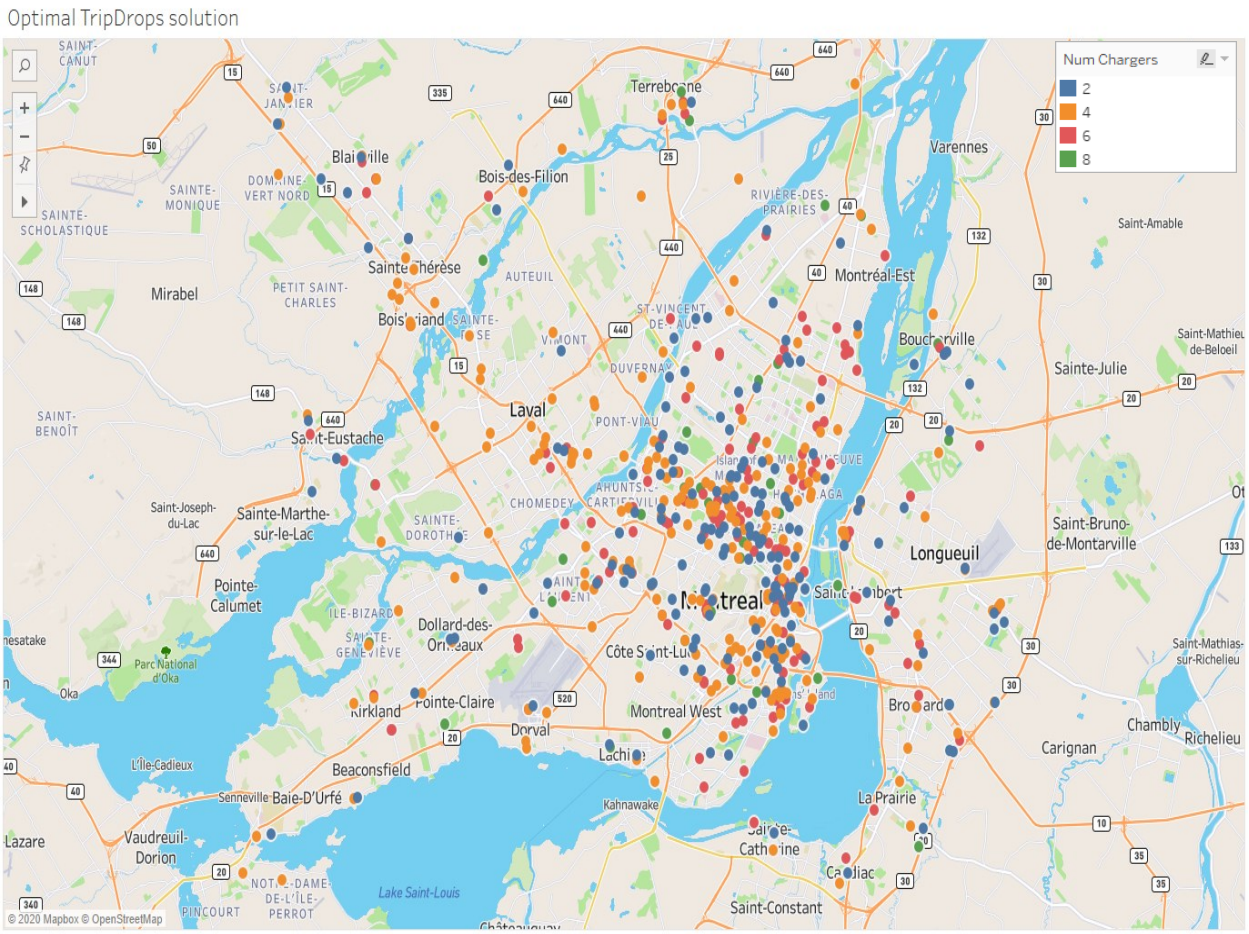

Figure 4.11 Locations of Charging Station for minimum trip drops

being dropped due to infeasible charging station distances. This solution, however, trades-off to Fitness and Convenience factor by 7.03% and 4.37%, respectively from the optimal solution.

Figure 4.12 provides the solution set of charging station locations and number of chargers in each station for the optimal convenience factor. By implementing this solution, the EV users are expected to travel only an average of 10.85 km to recharge their EV. However, the tradeoffs in Trip drop factor and Fitness factor of this solution to the optimal values are 4.7% and 20.4% respectively.

Figure 4.13 provides the solution set of charging station locations and their respective number of chargers for the optimal fitness factor. By implementing this solution, average charging event per day is maximized to 7.36 times per charger. The trade offs in this solution can be found as 9.13% deviation of convenience factor and 3.93% deviation in trip drops from the optimal values.

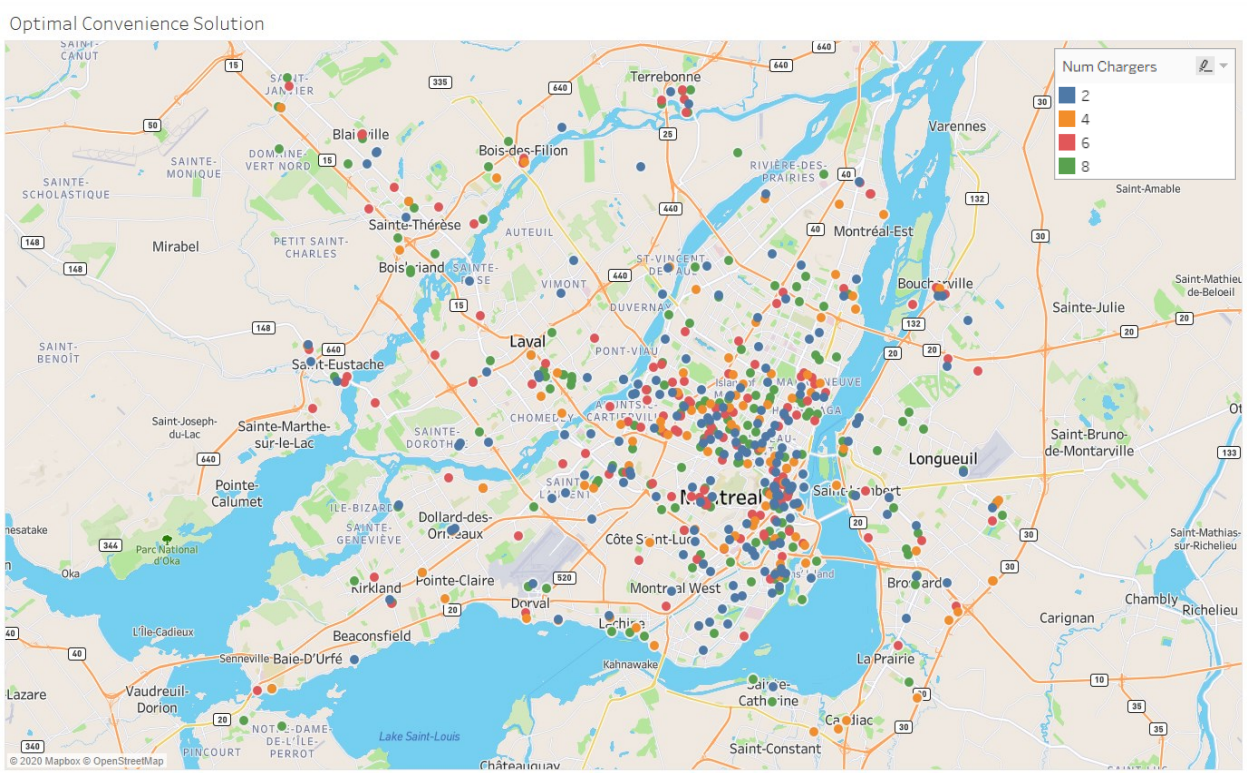

Figure 4.12 Locations of Charging Station for optimal convenience

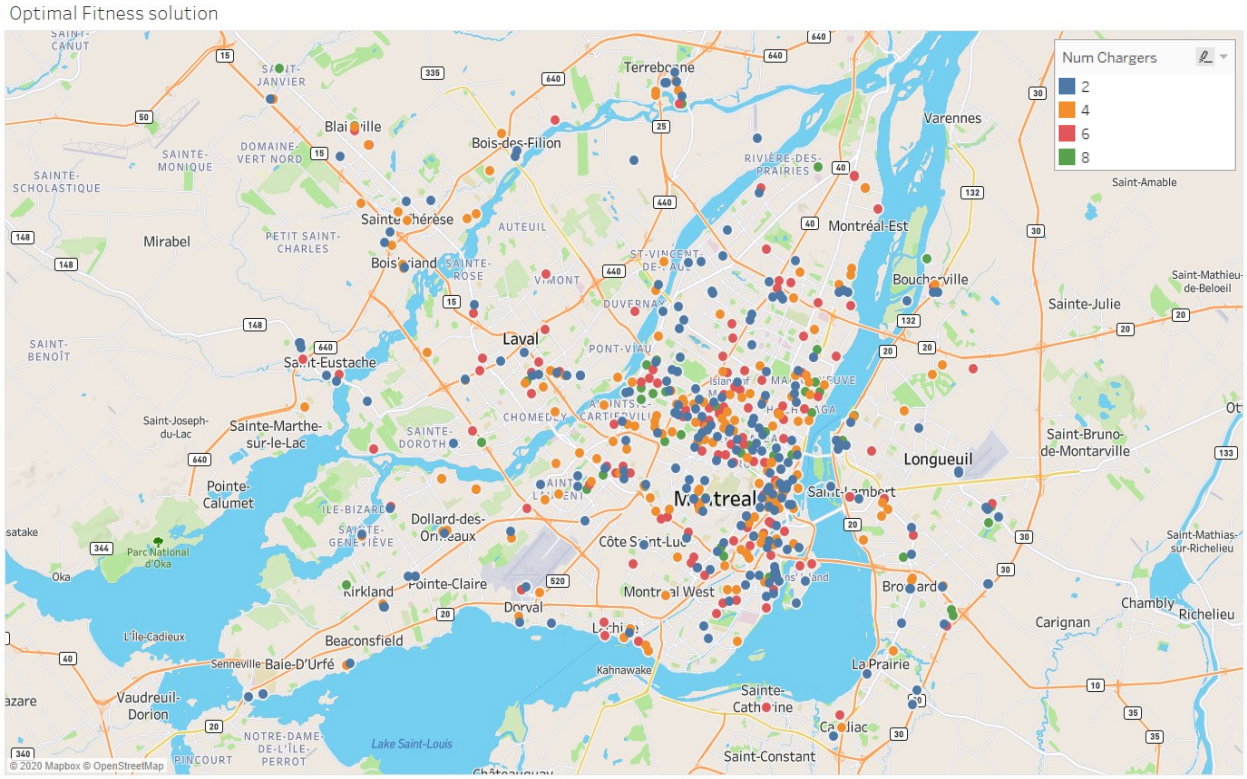

Figure 4.13 Locations of Charging Station for maximum utilization of chargers

In addition to finding the individual solutions of the most optimal objective values, pareto optimization lets a decision maker choose from the best trade offs among the possible solutions by choosing candidate solutions from the pareto front. For example, from Figure 4.10 it can be found that a solution exists above the most optimal solution that carries objective values show in Table 4-2 from Generation 14, candidate 12. Its devation from optimal convenience and trip drops are marginal while delivering a reasonable devation in fitness value. For a decision maker who wants to cover most trips with the least drops and the best convenience, can choose this solution with a reasonable increase in the cost of installation. The locations for the same solution is plotted in Figure 4.14.

Table 4-2 Example pareto optimal solution

Objective	**Value**	**Deviation from optimum**
Fitness	6.35122	13.7%
Convenience	10.96712	1.07%
Trip Drops	715	0.56%

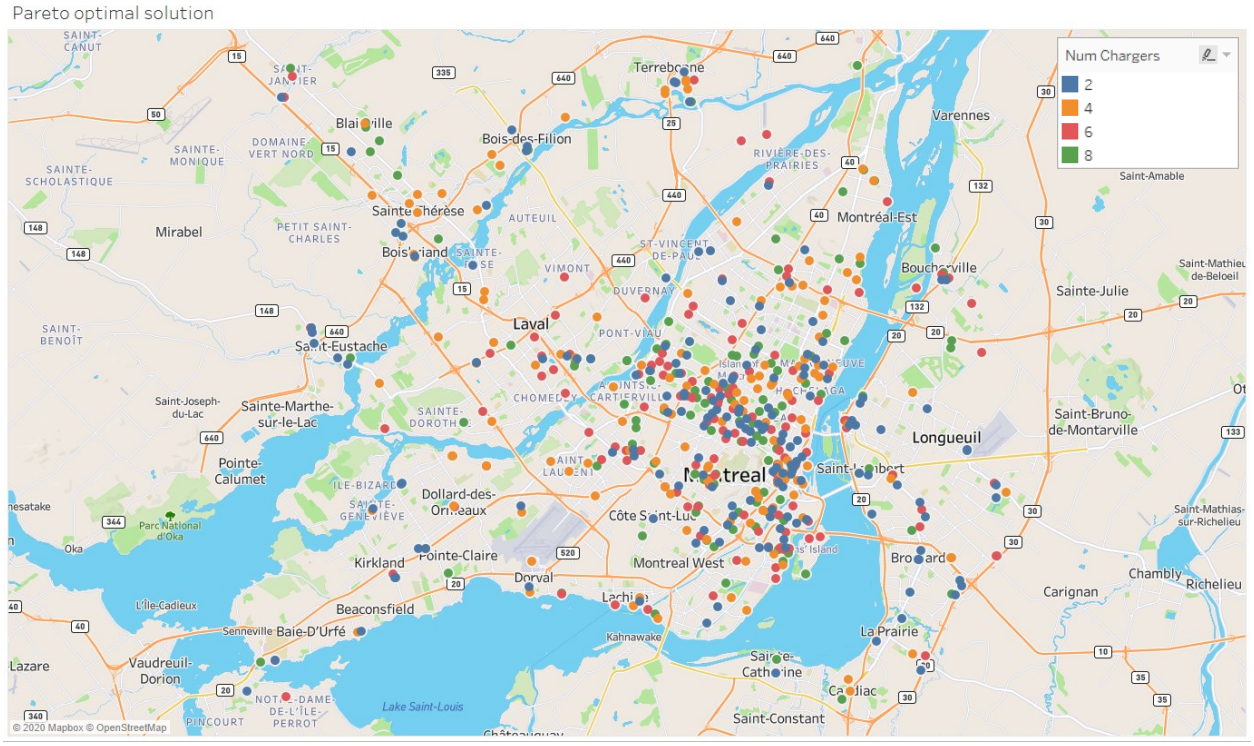

Figure 4.14 Example Pareto optimal solution

4.6 Comparison of results

This thesis implemented an ABM optimized through elitist pareto optimization GA to find pareto optimal solutions for the locating new fast charging stations in the city of Montreal. In this section, this approach is compared with other Location allocation methods in use qualitatively considering the parameters involved and assumptions made.

Exact solution-based methods like FCLM and Fuel-Travel back methods required a OD-Distance matrix, that required high computation resources to calculate the shortest distance between each origin and destination points, considering all combinations. Using ABM, this model could minimize the OD-Distance calculation to only the required routes, during the simulation is performed, thereby minimizing the computational requirement and time taken to run the model. ABM provides superior accuracy in demand modeling by simulating the travel of each EV through the road network and their charging demands in model time. This allows a more practical approach to decide the capacities of each charging station, depending on charging demand.

Table 4-3 Comparison with different LA methods

Method	**FCLM**	**Voronoi Diagram**	**Fuel-Travel-Back**	**TM-BPSO**	**GA-ABM**
Input Data	OD Distance Matrix	Traffic flow Density V Population Density	Traffic flow Heatmap	OD Pair data	**OD Pair data**
Computational intensity	Medium	Low	High	Medium	**Medium**
Demand Modeling	Aggregate	Approximate Aggregate	Averaged over time	Dynamic	**Dynamic**
Queue Support	NA	NA	NA	NA	**Queue with wait time limit**
GIS Based	No	Yes	Yes	Yes	**Yes**
Portability	Yes	Limited	Limited	Limited	**Yes**

ABM also stands apart from many Heuristic algorithms that offer dynamic demand modeling, by its support for queue implementation. This thesis models CA and CSA with their corresponding queues, that accepts trip requests and charging requests respectively. The trip queue in CA allows each EV in the model to execute one trip after another, even if the first trip did not finish in expected time (due to traffic delays, charging event delays etc.). The charging demand queue in CSA allows an accurate depiction of how many EVs can be charged at a time and how many EVs can be kept waiting depending on a maximum wait time tolerance. This allows a realistic management of peak time charging demand, ensuring that EVs will not search for far away CS with immediately available charger, rather than waiting a few minutes to charge at a nearby CS. The summary of these comparisons is listed in Table 4-3.

As no other research has conducted Location allocation problem analysis in the context of Montreal with similar objectives, a quantitative comparison of results is not possible between different methods.

4.7 Conclusive remarks

This Chapter discusses about the elements present in the simulation model, their concepts and implementation. The discussion of simulation model is emphasized on Anylogic modeling software as it allows multi-method modeling. The model was implemented according to the UML designs discussed in Chapter 3. Implementation of optimization algorithm through the custom experiment and Java classes in Anylogic is discussed in detail as well. Further, the important runtime parameters are provided and briefly discussed the process of running the model.

The results proved that the optimization algorithm is capable of converging random initial solutions to optimal final solutions through simulation modeling and pareto optimization. Results are plotted to provide detailed information on deriving meaningful conclusions.

Analysis from this Chapter achieved an integration of UML, agent-based modeling and evolutionary algorithm-based optimization through simulation modeling to solve the problem of optimal location allocation of capacitated charging stations in the city of Montreal.

Chapter 5

Conclusions and Future works

This Chapter discusses the conclusions derived from designing and implementing the optimal Location Allocation (LA) problem of EV charging stations using real traffic data from the city of Montreal. The chapter also proposes future works in the area of Agent Based Modeling optimization for LA problems.

5.1 Conclusions

This thesis reviewed current status of research in LA methods concerning EV charging station allocation and general trends in location allocation problems-based research. Special focus was made on heuristic algorithms based LA optimization problems and their superior performance compared to exact solution methods. Agent based modeling was discussed in detail along with the optimization methodology adopted to solve the LA problem under discussion using ABM.

The design for modeling the EV mobility through urban environments is explained with the help of UML diagrams. A clear and concise pictorial representation of the decision-making model is delivered by UML diagrams. The diagrams make it easier to develop, implement and maintain complex systems such as the problem discussed in this thesis, while allowing it to be comprehendible for users who are not experts in modeling. UML diagrams also saves time and effort in implementation on platforms where UML based modeling and development is supported.

Anylogic was chosen to conduct the research as it supports a wide variety of applications for ABM with its Object-Oriented design and implementation. Anylogic stands apart from other ABM modeling software due to its support for multi-method modeling, which was utilized by this thesis. The model design discussed through UML diagrams is implemented in Anylogic. The model was made more efficient in computation and practical by implementing the Trip Planner Agent's activity sequence through a Discrete Event (DE) process flow utility.

A simple genetic evolutionary algorithm is adopted and implemented to suite the need for a heuristic optimization algorithm that is adapted to be used in ABM. The logic that drives the

optimization algorithm to obtain the optimal pareto set of solutions is explained in detail and implemented using Anylogic's custom Java classes and Custom experiment fields.

Using the developed ABM for urban mobility and using real data of vehicle user's travels in the city of Montreal, this system was able to model the EV traffic through the city, during different times of the day with great accuracy. The ABM also models the internal battery states of each car as it travels according to its assigned trips and its charging events as and when required. The model reflects the charging demand in each charging station by limiting the number of vehicles that can remain in queue to charge. The trip planner agent assigns utilization values based on the number of times a charger is used and convenience values based on the reroute distance to reach the location to each charging station. The ABM model was able to successfully simulate the traffic cyclically for 5 days to model energy demand of cars over a working week, which is translated as charging demand in the optimization experiment.

The optimization experiment was run using 20 Candidate solutions over 25 generations and was proven to be converging to the optimal values. The results of the optimization algorithm were plotted to discuss the convergence properties and pareto front was obtained in an isometric plot with the three objectives. Pareto front solutions were also demonstrated using the 2-dimensional plots between pairs of objective values. Solutions for optimal trip drops case, convenience case and fitness case were plotted in the map of Montreal to show the exact locations provided by the solution along with their respective number of suggested chargers value.

5.2 Contributions

The model developed in this thesis is one of the first attempts at utilizing ABM in an optimization problem. Use of ABM and real traffic data of the city of Montreal makes this model highly accurate in predicting the expected charging demand of EVs in the city of Montreal. Pareto optimization based genetic algorithm was used to find the optimal solution sets of charging station locations and capacities with a wide variety of trade offs between each objective. Therefore, the results discussed in this thesis may be adopted by City of Montreal to implement planned upgrades to FCS from the existing Level-2 charging stations.

The developed model may also be utilized by EV fleet owners to asses the charging demands of their EVs using their origin – destination data.

5.3 Future works

While the results achieved are well defined and practical and reflects a general trend towards the selection of charging station locations and number of chargers in different scenarios, some of the assumptions made in selecting the traffic data may make the real world solution different from the suggested solution. Below are a few improvements and advancements that can be done to make the results of the model closer to accurate real world solution.

1. Cost of installation of power input to the Fast Charging Stations are not considered in this thesis and may vary in a non-linear fashion compared to the cost of chargers itself. A hybrid approach to location allocation problem by ranking the Charging Station location according to its installation cost and proximity to high energy electricity distribution can improve the results towards better cost estimation.
2. All trips in the OD survey data was assumed to be an EV trip, to cover a broader range to future EV users. More targeted OD data for specific use cases can be modelled using the same design to derive solutions for the certain use cases.
3. For more choices in candidate locations, existing gas stations locations may be included as possible solutions for Fast Charging Stations as suggested by M.Nicholas[14]
4. Currently a single simulation run takes over one week to learn all the routes from network server and store it cache for optimization through iterative simulation, as Anylogic currently does not support caching locally computed routes. This time constraint is a major bottleneck in using computers with higher computation power to model a system from the scratch. Anylogic is expected to release this feature in early 2021 and can be used to model larger systems on better computers/servers.
5. Larger systems with higher number of candidate solutions and agents will also need higher candidate population and generations of solutions to converge on to the optimal pareto front. A larger population of candidates in each generation can make results from this thesis's model much more distinguishable between pareto front solutions and dominated solutions.

References

[1] T. Hodges, "Public Transportation's Role in Responding to Climate Change," *U.S. Dep. Transp.*, no. January, p. 20, 2010.

[2] IEA, *Global EV Outlook 2018*. 2018.

[3] S. Deb, K. Kalita, and P. Mahanta, "Review of impact of electric vehicle charging station on the power grid," *Proc. 2017 IEEE Int. Conf. Technol. Adv. Power Energy Explor. Energy Solut. an Intell. Power Grid, TAP Energy 2017*, vol. 1, pp. 1–6, 2018.

[4] Z. Lin, J. Ogden, Y. Fan, and C.-W. W. Chen, "The fuel-travel-back approach to hydrogen station siting," *Int. J. Hydrogen Energy*, vol. 33, no. 12, pp. 3096–3101, Jun. 2008.

[5] R. F. Love, J. G. Morris, and G. O. Wesolowsky, *Facilities Location: Models & Methods*. North-Holland, 1988.

[6] C. S. ReVelle and H. A. Eiselt, "Location analysis: A synthesis and survey," *Eur. J. Oper. Res.*, vol. 165, no. 1, pp. 1–19, 2005.

[7] Y.-W. Wang and C.-C. Lin, "Locating road-vehicle refueling stations," *Transportation Research Part E: Logistics and Transportation Review*, vol. 45, no. 5. pp. 821–829, Sep-2009.

[8] C. M. Hosage and M. F. Goodchild, "Discrete space location-allocation solutions from genetic algorithms," *Ann. Oper. Res.*, vol. 6, no. 2, pp. 35–46, 1986.

[9] M. J. Hodgson, "A Flow-Capturing Location-Allocation Model," *Geogr. Anal.*, vol. 22, no. 3, pp. 270–279, 2010.

[10] M. Kuby and S. Lim, "The flow-refueling location problem for alternative-fuel vehicles," *Socio-Economic Planning Sciences*, vol. 39, no. 2. pp. 125–145, Jun-2005.

[11] Y.-W. Wang and C.-R. Wang, "Locating passenger vehicle refueling stations," *Transportation Research Part E: Logistics and Transportation Review*, vol. 46, no. 5. pp. 791–801, Sep-2010.

[12] Y.-W. Wang, "Locating battery exchange stations to serve tourism transport: A note,"

Transportation Research Part D: Transport and Environment, vol. 13, no. 3. pp. 193–197, May-2008.

[13] Y.-W. Wang, "An optimal location choice model for recreation-oriented scooter recharge stations," *Transportation Research Part D: Transport and Environment*, vol. 12, no. 3. pp. 231–237, May-2007.

[14] M. A. Nicholas, "Driving demand: What can gasoline refueling patterns tell us about planning an alternative fuel network?," *Journal of Transport Geography*, vol. 18, no. 6. pp. 738–749, Nov-2010.

[15] A. Shukla, J. Pekny, and V. Venkatasubramanian, "An optimization framework for cost effective design of refueling station infrastructure for alternative fuel vehicles," *Computers & Chemical Engineering*, vol. 35, no. 8. pp. 1431–1438, 2011.

[16] Z. Lin, J. Ogden, Y. Fan, and C. W. Chen, "The fuel-travel-back approach to hydrogen station siting," *Int. J. Hydrogen Energy*, vol. 33, no. 12, pp. 3096–3101, 2008.

[17] Charles S. ReVelle Ralph W. Swain, *Central Facilities Location; Geographical Analysis*. 1970.

[18] J. Li, X. Sun, Q. Liu, W. Zheng, H. Liu, and J. A. Stankovic, "Planning electric vehicle charging stations based on user charging behavior," *Proc. - ACM/IEEE Int. Conf. Internet Things Des. Implementation, IoTDI 2018*, pp. 225–236, 2018.

[19] S. Lim and M. Kuby, "Heuristic algorithms for siting alternative-fuel stations using the Flow-Refueling Location Model," *European Journal of Operational Research*, vol. 204, no. 1. pp. 51–61, 2010.

[20] D. M.S., *Network and Discrete Location: Models, Algorithms and Applications*, 1995th ed. New York: John Wiley and Sons, Inc., 1995.

[21] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. Cambridge, MA, USA: MIT Press, 1992.

[22] M. Mitchell, *An Introduction to Genetic Algorithms*. Cambridge, MA, USA: MIT Press,

1998.

[23] O. Alp, E. Erkut, and Z. V. I. Drezner, “An Efficient Genetic Algorithm for the p -Median,” pp. 21–42, 2003.

[24] H. Xu, S. Miao, C. Zhang, and D. Shi, “Optimal placement of charging infrastructures for large-scale integration of pure electric vehicles into grid,” *International Journal of Electrical Power & Energy Systems*, vol. 53. pp. 159–165, Dec-2013.

[25] C. M. MacAl and M. J. North, “Tutorial on agent-based modelling and simulation,” *J. Simul.*, vol. 4, no. 3, pp. 151–162, 2010.

[26] Wikipedia contributors, “AnyLogic --- {Wikipedia}{,} The Free Encyclopedia.” 2020.

[27] A. Awasthi, K. Venkitusamy, S. Padmanaban, R. Selvamuthukumaran, F. Blaabjerg, and A. K. Singh, “Optimal planning of electric vehicle charging station at the distribution system using hybrid optimization algorithm,” *Energy*, vol. 133, pp. 70–78, Aug. 2017.

[28] D. G. Brown, R. Riolo, D. T. Robinson, M. North, and W. Rand, “Spatial process and data models: Toward integration of agent-based models and GIS,” *J. Geogr. Syst.*, vol. 7, no. 1, pp. 25–47, 2005.

[29] M. Kane, “Electric Car Energy Consumption (EPA) Compared – April 1, 2019,” *InsideEVs*, 2019. [Online]. Available: https://insideevs.com/reviews/343702/electric-car-energy-consumption-epa-compared-april-1-2019/. [Accessed: 21-Oct-2020].

[30] H. Bersini, “UML for ABM,” *J. Artif. Soc. Soc. Simul.*, vol. 15, no. 1, p. 9, 2012.

[31] M. Oremland and R. Laubenbacher, “Optimization of Agent-Based Models: Scaling Methods and Heuristic Algorithms,” *J. Artif. Soc. Soc. Simul.*, vol. 17, no. 2, p. 6, 2014.

[32] J. Cohen, “Weighted kappa: Nominal scale agreement provision for scaled disagreement or partial credit.,” *Psychological Bulletin*, vol. 70, no. 4. American Psychological Association, US, pp. 213–220, 1968.

[33] M. Pennisi, R. Catanuto, F. Pappalardo, and S. Motta, “Optimal vaccination schedules using simulated annealing,” *Bioinformatics*, vol. 24, no. 15, pp. 1740–1742, 2008.

[34] T. Wang and X. Zhang, “3D protein structure prediction with genetic Tabu search algorithm

in off-lattice AB model," *2009 2nd Int. Symp. Knowl. Acquis. Model. KAM 2009*, vol. 1, pp. 43–46, 2009.

[35] J. Li, A. J. Parkes, and E. K. Burke, "Evolutionary Squeaky Wheel Optimization: A New Framework for Analysis," *Evol. Comput.*, vol. 19, no. 3, pp. 405–428, 2011.

[36] M. Oremland, "Optimization and Optimal Control of Agent-Based Models," pp. 1–66, 2011.

[37] A. Borshchev, "The big book of simulation modeling : multimethod modeling with AnyLogic 8," in *TA - TT* -, [Lisle, IL] SE - 612 pages : illustrations (some color) ; 25 cm: AnyLogic North America, 2019, p. "Designing state-based behavior: statecharts."

Appendices

Java Code implementation

Trip Planner Agent

calcDistance

```
double distance = tripCar.distanceByRoute(agent)/1000;
return distance;
```

addRemoveDestination

```
if (action == "add") {
destination = main.add_destinations(deslat, deslon);
destination.latitude = deslat;
destination.longitude = deslon;
destination.jumpTo(deslat,deslon);
} else if (action == "remove") {
main.remove_destinations(destination);
```

startTripProcess

```
tripCar.curTrip = this;
tripCar.coll_TripsToDo.remove(this);
tripCar.car.setVisible(true);

tripCar.jumpTo(orilat, orilon);
addRemoveDestination("add");
distance = calcDistance(destination);
traceToDB("trip" + this.getIndex() +" - " + distance + "km @" + date());
if(distance > 150) {
traceToDB("Distance too long to be in-city travel.. Skipping this trip !!");
} else {
tripCar.calcBatteryLeftWOCharging(distance);
enter.take(tripCar);
}
```

findActiveCS

```
int i =0;
for(chargingStation cs: main.chargingStations) {
	cs.updateAvailableChargers();
	if(cs.availableChgs > 0) {
		availableCS.add(i,cs);
		i++;
	}
}
```

Car Agent

calcBatteyLeftWOCharging

```
double expectedLevel = batteryLevel;

expectedLevel = expectedLevel - distance * main.kWh_PerKilometer;
double expLevelPercent = (expectedLevel/batteryCap);
if (expLevelPercent < main.p_BatteryLevelThreshold_ForCharging)
{
	chargeReqFlag = true;
}
```

checkChargingFeasibility

```
double expectedLevel1 = batteryLevel - distance1 * main.kWh_PerKilometer;
double expLevelPercent1 = (expectedLevel1/batteryCap);
if (expLevelPercent1 < 0)
{
	traceToDB("Car" + this.getIndex() + " Trip" + curTrip.getIndex() + " Time " +
date());
	traceToDB("Not enough charge to reach nearest Charging Station !");
	traceToDB(" Trip" + curTrip.getIndex() + " Added to infeasible trips");
	return false;
} else {
	return true;
}
```

Candidate Java Class

```
import java.util.Random;
public class Candidate implements Serializable {

	int[] cs_id;
	int[] gene;
	double fitness;
	double convenience;
	int tripDrops;
	int totCE;
	int neighbors;
	boolean newCandidate = true;
	boolean newCandidateFlag = true;
	//int num_genes = GA_Driver.numCS;

    public Candidate(int num_genes) {
	   this.gene = new int[num_genes];
	   this.cs_id = new int[num_genes];
		this.fitness = 0.0;
		this.convenience = 0.0;
		this.tripDrops = 0;
		this.totCE = 0;
		this.neighbors = 0;
		this.init_Candidate();
    }

    public Candidate init_Candidate() {
	   Random r = new Random();
	   //initialise the candidate population with random proportions of number of
chargers
	   int numzeros = r.nextInt(100);
	   int numtwos = r.nextInt(100);
	   int numfours = r.nextInt(100);
	   int numsixs = r.nextInt(100);
	   int numeights = r.nextInt(100);
	   int numTotal = numzeros + numtwos + numfours + numsixs + numeights;
	   double zeroFraction = 100.0 * numzeros/numTotal;
	   double twoFraction = 100.0 * (numzeros + numtwos)/numTotal;
	   double fourFraction = 100.0 * (numzeros + numtwos + numfours)/ numTotal;
	   double sixFraction = 100.0 * (numzeros + numtwos + numfours + numsixs)/
numTotal;
	   int[] rouletteWheel = new int[100];
	   for (int i=0; i<100; i++ ) {
		if ( i < zeroFraction) {
			rouletteWheel[i] = 0;
		} else if ( i < twoFraction) {
			rouletteWheel[i] = 2;
		} else if ( i< fourFraction) {
			rouletteWheel[i] = 4;
		} else if ( i< sixFraction) {
			rouletteWheel[i] = 6;
		} else {
			rouletteWheel[i] = 8;
		}
```

```
        }
                for (int i =0;i < cs_id.length; i++) {
                        cs_id[i] = i;
                        gene[i] = rouletteWheel[r.nextInt(100)]; // randomly assign
number of chargers 0,2,4,6,8
                }
                return this;
        }

        public Candidate getCandidate () {
           return this;
        }

        public List<Integer> printCandidate() {
           List<Integer> geneList = new ArrayList();
           for(int g: gene) geneList.add(g);
           return geneList;
        }

        public int numActiveChargers() {
           int count = 0;
           for(int numChg: gene) {
                if (numChg !=0) count++;
           }
           return count;
        }

        public void setRandomObjectiveVals () {
           Random r = new Random();
           this.fitness = r.nextDouble();
           this.convenience = r.nextDouble();
           this.tripDrops = r.nextInt(50);
           this.newCandidate = false;
        }

        public void printCandidateSummary() {
                System.out.println("Number of active CS: " + this.numActiveChargers());
                System.out.println("Fitness Factor: " + this.fitness);
                System.out.println("Convenience Factor: " + this.convenience);
                System.out.println("Trips Drop Factor : " + this.tripDrops);
                System.out.println("Neighbors : " + this.neighbors);
                //System.out.println(this.printCandidate());
        }


           /**
            * This number is here for model snapshot storing purpose<br>
            * It needs to be changed when this class gets changed
            */
           private static final long serialVersionUID = 1L;

}
```

Population Java Class

```
/**
 * Population
 */
import java.util.ArrayList;
import java.util.List;
public class Population implements Serializable {

	Candidate[] candidates;
    //int populationSize = GA_Driver.popSize;

    public Population(int populationSize) {
	this.candidates = new Candidate[populationSize];
    }

    public Population init_population(int numG) {
	for (int i =0; i< candidates.length ; i++) {
		candidates[i] = new Candidate(numG).init_Candidate();
	}
	return this;
    }

    public Candidate[] getParetoFront() {
	//initialise pareto front with whole population
	List<Candidate> paretoFront = new ArrayList<>(Arrays.asList(candidates));
	List<Candidate> ndCandidates = new ArrayList<Candidate>();

	//find non-dominated solutions
	for (Candidate c1 : candidates) {
		for (Candidate c2 : candidates) {
			if (c1 != c2 ) {
				if(checkDominance(c1,c2)) ndCandidates.add(c2);
			}
		}
	}

	//substract non-dominated solutions from population
	for(int i=0; i < ndCandidates.size(); i ++) {
		for(int j =0; j<paretoFront.size(); j++) {
			if(paretoFront.get(j).equals(ndCandidates.get(i)))
paretoFront.remove(j);
		}
	}

	List<Candidate> rank_fitness = sortDescending(paretoFront, p -> p.fitness);
	List<Candidate> rank_convenience = sortAscending(paretoFront, p ->
p.convenience);
	List<Candidate> rank_tripDrops = sortAscending(paretoFront, p -> p.tripDrops);
	List<Candidate> rank_aggregate = paretoFront;
	rank_aggregate.remove(rank_fitness.get(0));
	rank_aggregate.remove(rank_convenience.get(0));
	rank_aggregate.remove(rank_tripDrops.get(0));
	rank_aggregate = sortAscending(rank_aggregate, p -> (p.tripDrops/700 -
p.fitness/5 + p.convenience/10));
```

```
        List<Candidate> sortedPareto = new ArrayList<Candidate>();
        sortedPareto.add(rank_fitness.get(0));
        if (!sortedPareto.contains(rank_convenience.get(0))) {
                sortedPareto.add(rank_convenience.get(0));
        }
        if (!sortedPareto.contains(rank_tripDrops.get(0))) {
                sortedPareto.add(rank_tripDrops.get(0));
        }
        for(Candidate c: rank_aggregate) {
                if (!sortedPareto.contains(c)) {
                        sortedPareto.add(c);
                }
        }
        if (sortedPareto.size() != paretoFront.size()) {
                System.out.println("Sorted pareto front length doesn't match with real
pareto front");
        }
        Candidate[] paretoF = new Candidate[sortedPareto.size()];
            for (int i= 0; i < sortedPareto.size(); i++) {
                paretoF[i] = sortedPareto.get(i);
            }

        return paretoF;
    }

    public boolean checkDominance(Candidate c1, Candidate c2) {
        boolean checkFitness = (c1.fitness >= c2.fitness);
                boolean checkTripDrops = (c1.tripDrops <= c2.tripDrops);
                boolean checkConvenience = (c1.convenience <= c2.convenience);
                if (checkFitness && checkConvenience && checkTripDrops) {
                        //check for dominance
                boolean domFitness = (c1.fitness > c2.fitness);
                        boolean domTripDrops = (c1.tripDrops < c2.tripDrops);
                        boolean domConvenience = (c1.convenience < c2.convenience);
                        if (domFitness || domConvenience || domTripDrops) {
                                return true;
                        } else {
                                return false;
                        }
                } else {
                        return false;
                }
    }

    public void setNeighbors () {
        for (int i =0; i < candidates.length; i++) {
                candidates[i].neighbors = 0;
                for (int j=0; j < candidates.length; j++) {
                        boolean fitness_n = (Math.abs(candidates[i].fitness -
candidates[j].fitness) < 0.1) ;
                        boolean convenience_n = (Math.abs(candidates[i].convenience -
candidates[j].convenience) < 0.1) ;
                        boolean tripDrops_n = (Math.abs(candidates[i].tripDrops -
candidates[j].tripDrops) < 0.1) ;
```

```
                        if (fitness_n && convenience_n && tripDrops_n) {
                                candidates[i].neighbors++;
                        }
                }
        }
    }

    public Candidate[] getPopulation() {
        return candidates;
    }

        public void printPopulationSummary () {
                System.out.println("----------------------------------------");
                for (int i=0; i< candidates.length; i++) {
                        System.out.println("Candidate[" + i + "]" );
                        System.out.println("Number of active CS: " +
candidates[i].numActiveChargers());
                        System.out.println("Fitness Factor: " + candidates[i].fitness);
                        System.out.println("Convenience Factor: " +
candidates[i].convenience);
                        System.out.println("Trips Drop Factor : " +
candidates[i].tripDrops);
                        System.out.println("Neighbors : " + candidates[i].neighbors);
                        System.out.println(candidates[i].printCandidate());
                }
        }

        public void setRandomPopVals() {
                for(Candidate c: candidates) {
                        if (c.newCandidate) c.setRandomObjectiveVals();
                }
        }



        /**
         * This number is here for model snapshot storing purpose<br>
         * It needs to be changed when this class gets changed
         */
        private static final long serialVersionUID = 1L;

}
```

MultiObjGA Java Class

```
public class MultiObjGA implements Serializable {

    public static final int populationSize = 10;
    public MultiObjGA() {
    }

      @Override
      public String toString() {
            return super.toString();
      }

      public static Population init_GA_experiment (int numC, int numG) {
            Population newPop = new Population(numC).init_population(numG);
            return newPop;
      }

      public Population crossover (Population population) {
            int numCandidates = population.candidates.length;
            int numCS = population.candidates[0].gene.length;
            int num_elites = roundToInt(numCandidates / 2.0);
            Population newPopulation = new
Population(numCandidates).init_population(numCS);
            List<Candidate> nextGenCandidates = new ArrayList<Candidate>();
            Candidate[] paretoPop = population.getParetoFront();
            if (paretoPop.length < num_elites) {
                  num_elites = paretoPop.length;
            }
            //add Candidates in the pareto front to next generation
            for (int i =0; i < num_elites; i++) {
                  nextGenCandidates.add(paretoPop[i]);
            }

            //add children candidates until population size
            while (nextGenCandidates.size() < numCandidates) {

                  //choose 2 parents to crossover
                  Candidate[] parents = new Candidate[2];
                  for (int i = 0; i <2 ; i++) {
                        //select 5 random candidates from the current population
                        Population crossoverPop = new Population(5);
                        Random r = new Random();
                        for (int j=0; j < 5; j++) {
                              int indx = r.nextInt(10);
                              crossoverPop.candidates[j] =
population.candidates[indx];
                        }

                        //check for dominant solutions
                        Candidate[] paretoFront = crossoverPop.getParetoFront();

                        if(paretoFront.length == 1) {
                              parents[i] = paretoFront[0];
                        } else {
```

```
                                    Candidate diverseCandidate = paretoFront[0];
                                    for (int j = 1; j<paretoFront.length; j++) {
                                          if(diverseCandidate.neighbors >
paretoFront[i].neighbors) diverseCandidate = paretoFront[i];
                                    }
                                    parents[i] = diverseCandidate;
                              }
                        }

                        Random r = new Random();
                        Candidate childA = new Candidate(numCS);
                        Candidate childB = new Candidate(numCS);
                        /*
                        // Random crossover
                        for (int i =0; i < numCS; i++) {
                              boolean pSelect = r.nextBoolean();
                              if (pSelect == true) {
                                    childA.gene[i] = parents[0].gene[i];
                                    childB.gene[i] = parents[1].gene[i];
                              } else {
                                    childA.gene[i] = parents[1].gene[i];
                                    childB.gene[i] = parents[0].gene[i];
                              }
                        }
                        */
                        //2 point crossover
                        int point1 = roundToInt(numCS/3.0);
                        int point2 = roundToInt(numCS/1.5);
                        for(int i =0; i< point1 ; i ++) {
                              childA.gene[i] = parents[0].gene[i];
                              childB.gene[i] = parents[1].gene[i];
                        }
                        for (int i = point1; i <point2; i++) {
                              childA.gene[i] = parents[1].gene[i];
                              childB.gene[i] = parents[0].gene[i];
                        }
                        for (int i = point2; i< numCS; i++) {
                              childA.gene[i] = parents[0].gene[i];
                              childB.gene[i] = parents[1].gene[i];
                        }

                        nextGenCandidates.add(childA);
                        if(nextGenCandidates.size() != population.candidates.length)
nextGenCandidates.add(childB);
                  }

                  for (int i =0; i < population.candidates.length; i++ ) {
                        newPopulation.candidates[i] = nextGenCandidates.get(i);
                  }

                  return newPopulation;
            }

            public Population mutate (Population population, double p_mutation) {
                  Random r = new Random();
```

```
		Population mutatedPopulation = population;
		for (Candidate c: mutatedPopulation.candidates) {
			if(c.newCandidate) {
				for(int g=0; g< c.gene.length; g++ ) {
					double mFlag = r.nextDouble();
					if (mFlag < p_mutation) {
						c.gene[g] = Math.abs((r.nextInt(5)) *2);
					}
				}
			}
		}
		return mutatedPopulation;
	}

	public Population evolve (Population population, double p_mutation) {
		return mutate(crossover(population), p_mutation);
	}


	/**
	 * This number is here for model snapshot storing purpose<br>
	 * It needs to be changed when this class gets changed
	 */
	private static final long serialVersionUID = 1L;

}
```

Custom Optimization Experiment Java Code in Anylogic

```
int popSize = 30;
int numCS = 731;
double p_mutation = 0.2;
int num_generations = 10;
TextFile textFile = null;
TextFile EGA_Data = null;
TextFile EGA_Data_gen = null;

Population currentPop = new Population(popSize).init_population(numCS);
MultiObjGA ga = new MultiObjGA();

int generationNum = 0;
while (generationNum < num_generations) {
	//Run simulation for all Candidate solutions in population
	traceln("#########################################################################
##");
	traceln("###################   Generation "+ generationNum + "
######################");
	traceln("#########################################################################
##");
	for (int i = 0; i < popSize; i++) {
		traceln("-----------------------------------------------------------------
");
		traceln ("Experiment GA_loop Generation: "+ generationNum + " Candidate:
" + i);
		traceln("-----------------------------------------------------------------
");
		// Create Engine, initialize random number generator:
		Engine engine = createEngine();
		engine.setTimeUnit( MINUTE );
		// Fixed seed (reproducible simulation runs)
		engine.getDefaultRandomGenerator().setSeed( 1 );
		engine.setStartTime( 0.0 );
		engine.setStartDate( toDate( 2020, SEPTEMBER, 14, 0, 0, 0 ) );
		// Set stop time:
		engine.setStopDate( toDate( 2020, SEPTEMBER, 15, 2, 0, 0 ) );
		// Create new root object:
		Main root = new Main( engine, null, null );
		// TODO Setup parameters of root object here
		root.setParametersToDefaultValues();
		// root.kWh_PerKilometer = 123;
		// ...
		root.current_Population_CS = currentPop;
		if(i==0) {
			//root.current_Population_CS = currentPop;
			root.current_Population_CS.printPopulationSummary();
			traceln("##########   Generation "+ generationNum +"
############");
			textFile = new TextFile( root, null, TextFile.WRITE, "out"+
generationNum +".csv", null, null );
			textFile.println("Candidate
Number,CS_ID,num_chargers,fitness,convenience,latitude,longitude");
			if (generationNum == 0) {
```

```
                        EGA_Data = new TextFile( root, null, TextFile.WRITE,
"Evolution_data.csv", null, null );

      EGA_Data.println("Generation,Candidate_Num,ActiveChargers,Fitness,Convenience,
TripDrops,Total_charging_events,newCandidateFlag");
                  }
                  EGA_Data_gen = new TextFile( root, null, TextFile.WRITE,
"Evolution_data"+ generationNum +".csv", null, null );

      EGA_Data_gen.println("Generation,Candidate_Num,ActiveChargers,Fitness,Convenie
nce,TripDrops,Total_charging_events,newCandidateFlag");
            }

            if (currentPop.candidates[i].newCandidate) {
                  root.candidate_num = i;

                  // Prepare Engine for simulation:
                  engine.start( root );
                  // Start simulation in fast mode:
                  engine.runFast();
                  // TODO Process results of simulation here
                  // traceToDB( "chargingStations:" );
                  // traceToDB( inspectOf( root.chargingStations ) );
                  // ...
                  root.update_CS_dataset();
                  traceln("Candidate"+ i + " done");
                  for (int j = 0 ; j<root.fitness_data.size() ; j++) {
                        textFile.println(i+","+ root.fitness_data.getX(j) +"," +
root.convenience_data.getX(j) +","+root.fitness_data.getY(j) + "," +
root.convenience_data.getY(j) + "," + root.location_data.getX(j) + "," +
root.location_data.getY(j));
                  }
            } else {
                  traceln("Candidate Solution already simulated, skipping !!");
            }
            //write contents of dataset of current run:

            currentPop.candidates[i] = root.current_Population_CS.candidates[i];
            currentPop.candidates[i].printCandidateSummary();
            //currentPop = root.current_Population_CS;
            // Destroy the model:
            engine.stop();
      } // end of for(candidates)
      traceln("Generation "+ generationNum +" File written");
      textFile.close();
      for(int c=0; c < currentPop.candidates.length; c++) {
            Candidate ca = currentPop.candidates[c];
            EGA_Data.println(generationNum+","+ c +","+ ca.numActiveChargers() +","+
ca.fitness +","+ ca.convenience +","+ ca.tripDrops +","+ ca.totCE +","+
ca.newCandidateFlag);
            EGA_Data_gen.println(generationNum+","+ c +","+ ca.numActiveChargers()
+","+ ca.fitness +","+ ca.convenience +","+ ca.tripDrops +","+ ca.totCE +","+
ca.newCandidateFlag);
            ca.newCandidateFlag = false;
      }
```

```
        EGA_Data_gen.close();
        currentPop.setNeighbors();
        currentPop.printPopulationSummary();
        currentPop = ga.evolve(currentPop, p_mutation);
        generationNum++;
} // end of while(generation)
EGA_Data.close();
traceln("Simulation Completed, Evolution Data file written");
```