Capacity Expansion of EV Charging Network: Model, Algorithms and A Case Study

CAPACITY EXPANSION OF ELECTRIC VEHICLE CHARGING NETWORK: MODEL, ALGORITHMS AND A CASE STUDY

BY

QIANQIAN CHEN

A THESIS

SUBMITTED TO THE SCHOOL OF COMPUTATIONAL SCIENCE AND ENGINEERING

AND THE SCHOOL OF GRADUATE STUDIES

OF MCMASTER UNIVERSITY

IN PARTIAL FULFILMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

Master of Science (2019) McMaster University

(Computational Science and Engineering) Hamilton, Ontario, Canada

TITLE:	Capacity Expansion of Electric Vehicle Charging Network: Model, Algorithms and a Case Study
AUTHOR:	Qianqian Chen Ph.D. (Operations Research and Cybernetics), Zhejiang University, Hangzhou, China
SUPERVISOR:	Dr. Kai Huang
NUMBER OF PAGES:	x, 82

Abstract

Governments in many counties are taking measures to promote electric vehicles. An important strategy is to build enough charging infrastructures so as to alleviate drivers' range anxieties. To help the governments make plans about the public charging network, we propose a multi-stage stochastic integer programming model to determine the locations and capacities of charging facilities over finite planning horizons. We use the logit choice model to estimate drivers' random choices towards different charging stations nearby. The objective of the model is to minimize the expected total cost of installing and operating the charging facilities. Two simple algorithms are designed to solve this model, an approximation algorithm and a heuristic algorithm. A branch-and-price algorithm is also designed for this model, and some implementation details and improvement methods are explained. We do some numerical experiments to test the efficiency of these algorithms. Each algorithm has advantages over the CPLEX MIP solver in terms of solution time or solution quality. A case study of Oakville is presented to demonstrate the process of designing an electric vehicle public charging network using this model in Canada.

Keywords: multi-stage stochastic integer programming, logit choice model, facility location, capacity expansion, electric vehicle charging network, algorithm.

To my husband

Acknowledgements

The two-year journey in McMaster University will be my invaluable memory. I have learned a lot and I have experienced so many great things here.

I would like to express my gratitude to my supervisor Dr. Kai Huang. His guidance and patience helped me in all the time of research and writing of this thesis. His encouragement and trust are always my driving force to continue my work. I am very lucky to meet him and have him as my supervisor.

I would also like to thank Dr. Mark Ferguson for his generous help on the development of the model and the experiment data. He gave me so many constructive suggestions throughout my research. His support helped me a lot especially when Dr. Huang was absent.

I am grateful to the rest of my thesis committee: Dr. Yun Zhou and Dr. Antoine Deza for taking the time to review my thesis and to attend my defense.

Besides, I also would like to thank Dr. John Miltenburg who taught me a lot of good work habits when I was his teaching assistant. Thanks also go to my fellow graduate students, especially Mohsen Zargoush, Shuai Zhang and Mingjie Jiang.

Last but not least, I would like to thank my husband, my parents, my parents-in-law, my grandfather, my siblings and their families for their support and love.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter, we introduce the backgrounds of the electric vehicle charging network and tools we will use to develop our multi-stage electric charging network capacity expansion (EVCE) model.

1.1 Electric Vehicle Charging Network

Electric Vehicles (EVs) are growing rapidly in the present decade due to technological developments, concerns about volatile oil prices and an increased focus on environmentally friendly and renewable energy. Governments in many countries have adopted measures to promote electric vehicles. An important action is to install more public charging facilities to reduce drivers' range anxiety, which describes the fear of running out of battery power before getting to the destination or reaching a charging station. The development of the charging network leads to a facility location and capacity expansion problem.

In most traditional facility location models, the demand of an area is defined to be

covered by a facility if the area is close to the facility within a certain coverage radius (Church and ReVelle, 1974). A number of models are proposed under this setting to select the best locations that will maximize utility, minimize cost or achieve other objectives. The p-Median and p-Center problem minimize the total travel cost and the maximum travel cost when people using the facilities respectively (Hakimi, 1964). The maximum coverage location problem maximizes the demand coverage while a given number of facilities are installed (Church and ReVelle, 1974).

Similar models are proposed for designing the public EV charging network. Frade *et al.* (2011) used the maximal covering model to maximize the demand covered by the charging facilities for Lisbon, Portugal. He *et al.* (2016) compared three facility location models for the public EV charging network in Beijing, China, including the set covering model, the maximal covering location model and the p-median model. Only the third model gives the necessary capacities of the charging stations by allocating all the charging demand of an area to its closest charging station. Huang *et al.* (2016) considered the partial covered area and they extended the polygon overlay method to split these areas.

In this thesis, we present an optimization model to investigate the design of the EV charging network for public charging. The considered area are divided into several zones. The charging demands are estimated by the number of charging events taken place in each zone. The objective is to minimize the total cost of installing charging facilities and operating them during the time period we study.

Planning EV charging network is a long-term task, which means we make discrete decisions over finite time periods. For example, we make a plan every two years. So,

there are some uncertain parameters, such as the charging demands and the charging facility expansion costs in the future. Earlier approaches use stochastic control theory to deal with the uncertainties in the stochastic capacity expansion problems. The demands were assumed to be stochastic processes (Manne, 1961). Then the use of scenarios becomes popular in stochastic programming due to the increased computational power. What is more, the scenarios should be relatively modest to make sure that the model can be solved with reasonable computational effort in practice. In this thesis, we use a scenario tree to deal with the uncertain parameters and study the capacity expansion problem by making a sequence of decisions over finite planning horizons. Therefore, the model is a multi-stage stochastic mixed-integer program. This multi-stage model can be transformed to a deterministic equivalent mixed-integer program if a finite number of scenarios and their probabilities are given (Ahmed and Sahinidis, 2003).

1.2 Customer Choice Behaviours

In the practice of designing the EV charging networks, every zone in the map is usually covered by several electric charging facilities and drivers in the same zone may choose different charging stations nearby. Therefore, it is not reasonable to allocate the whole charging demand of a zone to its closest charging station. The random choices of drivers make it difficult to estimate the number of charging events that would take place in each charging station.

The behaviours of drivers choosing different charging stations are called customer choice behaviours. Consumer behaviour analysis is widely used in marketing research and economic psychology. It is also very useful in facility location planning problems.

Serra *et al.* (1999) assumed that customers patronize the facilities in proportion to the customer-facility distances. In many literatures, researchers use multinomial logit model to consider customer choice behaviour. The model is frequently used in transportation and demographic research. When analyzing the choice of an individual among a set of alternatives, multinomial logit model focuses on the choice set of each individual and treats a choice as a function of the characteristics of the alternatives in the choice set (Hoffman and Duncan, 1988). Benati and Hansen (2002) assumed that the utility of a customer to patronize a facility can be divided in a measurable part and a random part in their model to locate facilities in order to capture more customer demand. Aros-Vera *et al.* (2013) used the cost of choosing a facility as this facility's characteristic when they use the logit choice model in their facility location problem.

Since using the logit choice model usually results in a non-linear formulation, computational difficulty becomes an important factor that hinders the widespread use of the model in facility location problems. Haase and Müller (2014) compared the solvability of three different reformulations for facility location models in which logit choice probabilities are used in the objective functions. Moreover, researchers usually use the logit formulation in the objective functions while keeping the constraints linear. This creates a lot of limitations on the use of the logit choice model.

In this thesis, we use the multinomial logit choice model to predict drivers' choices towards different charging stations nearby and then estimate the charging events that may take place at each charging station. In this way, we will get not only the locations of charging stations, but also the required capacity of charging stations.

1.3 Mixed-Integer Linear Program

An integer linear programming problem is a linear programming problem in which part or all of the variables are restricted to integers. The problem is known as a mixed-integer programming (MIP) problem when some decision variables are not discrete.

We use some techniques to make the objective function and constraints linear with respect to the decision variables when multinomial logit choice model is used in our EVCE model. Then a mixed-integer linear program (MILP) is obtained.

The linearization technique is quite useful since programs with non-linear constraints and objective function are hard to solve, especially when the nonlinear constraints and objectives are neither convex nor concave. Linear programming has been shown solvable in polynomial-time (Khachiyan, 1979). Efficient algorithms such as the simplex algorithm and algorithms based on interior point method are developed and investigated with a lot of theoretical and practical breakthroughs (Murty, 1983; Karmarkar, 1984; Vanderbei *et al.*, 1986; Renegar, 1988; Kojima *et al.*, 1989; Schrijver, 1998).

However, the linearization procedure largely increases the scale of the model since a lot of new variables are added to the model. Thus the problem we are facing now is to solve a large-scale mixed-integer linear program. We develop an approximation algorithm and a heuristic algorithm based on specific properties of this model. The approximation algorithm is able to get a relatively good solution and a lower bound of the model in a reasonable time. The heuristic algorithm can obtain a solution in seconds, but without a lower bound to estimate the quality of the solution. A

branch-and-price algorithm which is a hybrid of branch-and-bound and column generation methods to solve large-scale mixed-integer program (Barnhart *et al.*, 1998) is also designed and investigated in this thesis with some implementation details and methods to improve its efficiency. In most instances, the branch-and-price algorithm can get the optimal solution within a relatively short time compared to the solution procedure by the CPLEX MIP solver. We do numerical experiments to show the efficiency of these three algorithms and the effect of improvement methods for the branch-and-price algorithm.

To test the EVCE model in practice, we study a case by designing the EV charging network for Oakville, Canada. Details about how to collect data and how to interpret the results are explained and demonstrated.

This thesis is organized as follows. Chapter 2 describes the framework and mathematical formulations used in the EVCE model for the EV charging station location and capacity expansion problem. In Chapter 3, we present an approximation algorithm and a heuristic algorithm, and their efficiency. The branch-and-price algorithm is investigated in Chapter 4 to solve this large-scale mixed-integer linear programming model. In Chapter 5, we use this model to design and analyze the EV charging network of Oakville, Canada. Finally, we conclude the thesis with a summary in Chapter 6.

Chapter 2

Model Development

In this chapter, details about the development of the EVCE model are presented.

2.1 Framework

Designing the charging network is a long-term issue. To handle the uncertainty of the parameters and data in this problem, we use a rooted scenario tree $\mathcal{T}$ with T discrete decision stages to build the multi-stage model. Every stage corresponds to a time period during which we make locating and capacity expansion decisions. At every stage, there are several nodes that represent all the potential states of the nature. The root node is written as $n = 1$. Let $\mathcal{N}$ denote the set of all the nodes in the scenario tree. The probability of node $n \in \mathcal{N}$ is denoted by ϕ_n. Clearly, the sum of all the probabilities for nodes in the same stage is 1. For every node $n \in \mathcal{N}$, its direct predecessor is denoted by $a(n)$. We denote the direct predecessor of the root node as node 0 for notational simplicity. The set of node n and all its predecessors is denoted by $\mathcal{P}_n$. All the nodes in $\mathcal{P}_n$ can form a path from node n to the root node and $a(n)$

is the node next to n on the path. The set of node n and all its successors is denoted by $\mathcal{T}_n$ and all the nodes in $\mathcal{T}_n$ can form a sub-tree whose root is the node n. Figure 2.1 gives a scenario tree and shows $a(n)$ and $\mathcal{P}_n$ for a node $n \in \mathcal{N}$.

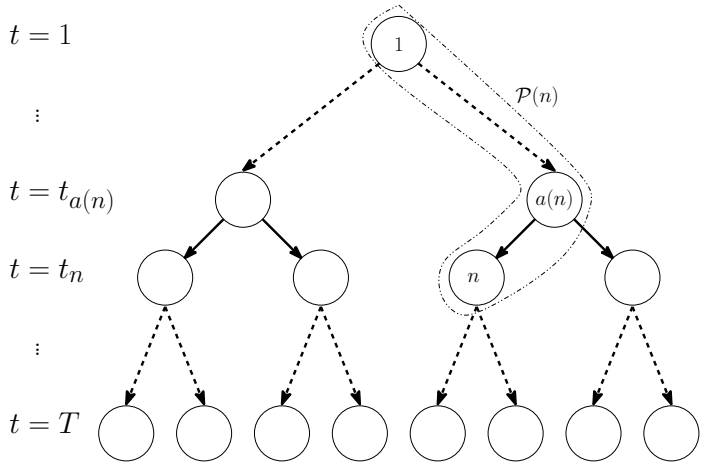

Figure 2.1: Scenario tree and predecessors

In the multi-stage model, a specific scenario tree is given with finite nodes and their probabilities. Let $\mathcal{I}$ denote the set of zones we are investigating. It is assumed that there is a set of candidate locations for the charging stations which is denoted by $\mathcal{J}$. At each node $n \in \mathcal{N}$, we need to decide whether or not a charging station is located at a candidate location $j \in \mathcal{J}$ by using the binary decision variable $x_{n,j}$. The integer decision variable $y_{n,j}$ denotes the number of chargers in the charging station located in j at node n for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$. Each node in the scenario tree is associated with parameters or data such as the charging demands, the cost of installing new charging stations and chargers, and the cost of operating these facilities. Obviously, the decisions at node n is directly influenced by the decisions at nodes in $\mathcal{P}_n$, since the installed charging stations and chargers should not disappear over time.

We follow the point demand location approach and assume that demand of a zone

is located at a distinct place in the zone (Tong and Murray, 2009; Tu *et al.*, 2016). In our EVCE model, the distinct place can be the geometric centre of the zone, or the point with densest EVs in the zone. To make the model more accurate, the area under study should be divided as fine as possible. But too many zones will make the model too difficult to solve, so tradeoff should be made between accuracy and computational difficulty. The charging demand of each zone is the predicted number of charging events in a zone. The objective is to minimize the expected total cost of installing and operating the charging facilities while the charging demands are satisfied at a certain level.

In order to make sure the charging facilities are adequate for every zone, the drivers' random choices to different charging stations nearby should be taken into consideration. A widely used model called logit choice model assumes that the characteristics of the alternatives will impact the choices most. In the paper by Hoffman and Duncan (1988), each individual has a set of candidate choices (i.e., alternatives). The probability of an individual i choosing alternative j is given by:

$$\alpha_{i,j} = \frac{e^{ad_{i,j}}}{\sum_{k \in \mathcal{J}_i} e^{ad_{i,k}}},$$

where $d_{i,j}$ is the characteristic of the jth alternative for individual i, $\mathcal{J}_i$ is individual i's choice set and a is a corresponding parameter.

We use the logit choice model in the charging network problem to predict the choices of EV drivers. Let $d_{i,j}$ denote the distance between a zone i and a candidate location j for any $i \in \mathcal{I}$ and $j \in \mathcal{J}$. Let $R_{n,i}$ denote the maximum distance that drivers of zone i are willing to go to a charging station at node $n \in \mathcal{N}$. Thus, only charging stations within $R_{n,i}$, called coverage radius, are considered by the EV drivers

of zone i at node n. We denote $\mathcal{I}_{n,j}$ as the set of zones near candidate location j within the coverage radius $R_{n,i}$. Clearly, $\mathcal{I}_{n,j} = \{i \in \mathcal{I} \mid d_{i,j} \leq R_{n,i}\}$ for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$. Similarly, $\mathcal{J}_{n,i} = \{j \in \mathcal{J} \mid d_{i,j} \leq R_{n,i}\}$ is the set of candidate charging locations near zone i within the coverage radius $R_{n,i}$ for all $n \in \mathcal{N}$ and $i \in \mathcal{I}$.

An EV driver of zone i is an individual in the logit choice model and his or her choice set is $\mathcal{J}_{n,i}$ which is defined above. We use $-d_{i,j}$ as the characteristic of candidate location j, i.e., alternative j, in the choice set $\mathcal{J}_{n,i}$, since the closer the charging station is, the more likely it is selected by the driver.

In practice, distance is definitely not the only factor that will affect drivers' choices. For example, some drivers may avoid a charging station if it is always overcrowded. The overcrowding may happen when a charging station has a limitation on the number of chargers it can accommodate but the maximum number of chargers can not satisfy the predicted charging demands. Since this factor is hard to quantify or it can make our model too complicated, we remove the upper limit on the number of chargers in each charging station. In this way, the model can freely determine the number of chargers in each charging station based on the predicted charging demands, so the overcrowding will be less likely to happen and thus will be less likely to affect drivers' choices.

In the EVCE model, a decision variable $\alpha_{n,i,j}$ is used to denote the estimated probability that an EV driver of zone i will go to charging station j to charge the vehicle at scenario node n. Let $e_{i,j} = e^{-a_i d_{i,j}}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}$ for some constants $a_i (a_i > 0)$. Then for any zone $i \in \mathcal{I}, j \in \mathcal{J}_{n,i}$ at node $n \in \mathcal{N}$,

$$\alpha_{n,i,j} = \frac{x_{n,j} \cdot e_{i,j}}{\sum\limits_{k \in \mathcal{J}_{n,i}} x_{n,k} \cdot e_{i,k}}, \tag{2.1.1}$$

where $x_{n,j}$ is binary and $x_{n,j} = 1$ if and only if there is a station built in location j at node n. Note that $\alpha_{n,i,j}$ is not well defined when all the candidate locations in $\mathcal{J}_{n,i}$ do not have a charging station (i.e., $x_{n,k} = 0$ for all $k \in \mathcal{J}_{n,i}$). Therefore, we introduce a constraint $\sum_{k \in \mathcal{J}_{n,i}} x_{n,k} \geq 1$ to make sure that drivers of zone i have at least one choice.

2.2 Notations

The notations of parameters and decision variables in this thesis are listed and are explained in Table 2.1.

Table 2.1: Model notations

Sets	**Descriptions**
$\mathcal{I}$	Set of zones
$\mathcal{J}$	Set of all candidate locations for charging stations
$\mathcal{N}$	Set of nodes in the scenario tree
$\mathcal{I}_{n,j}$	Set of zones near candidate location j within the coverage radius $R_{n,i}$ for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$
$\mathcal{J}_{n,i}$	Set of candidate locations near zone i within the coverage radius $R_{n,i}$ for all $n \in \mathcal{N}$ and $i \in \mathcal{I}$
Parameters	**Descriptions**
ϕ_n	Probability of node n for all $n \in \mathcal{N}$
$a(n)$	Direct predecessor (a.k.a., father) of node n for all $n \in \mathcal{N}$
$s(n)$	Set of direct successors (a.k.a., children) of node n for all $n \in \mathcal{N}$
$\mathcal{P}_n$	Set of predecessors of node n (including n) for all $n \in \mathcal{N}$

$\mathcal{T}_n$	Set of successors of node n (including n) for all $n \in \mathcal{N}$
$c^s_{n,j}$	Cost of building a new charging station located in j at node n for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$
$c^p_{n,j}$	Cost of adding a charger to the charging station located in j at node n for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$
$\bar{c}^s_{n,j}$	Cost of operating a charging station located in j at node n during its time period for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$
$\bar{c}^p_{n,j}$	Cost of operating a charger in the charging station located in j at node n during its time period for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$
$x_{0,j}$	Binary parameter, initial charging station status, i.e., $x_{0,j} = 1$ if and only if there is initially a charging station located in j for all $j \in \mathcal{J}$
$y_{0,j}$	Initial number of chargers in the charging station located in j for all $j \in \mathcal{J}$
M_j	A sufficiently large positive integer number that the number of chargers in charging station located in j would never exceed in any case for all $j \in \mathcal{J}$
$W_{n,i}$	Charging demand of zone i during the busiest K hours at node n for all $n \in \mathcal{N}$ and $i \in \mathcal{I}$
$w_{n,i}$	Basic charging demand of zone i during the busiest K hours without the influence of the number of charging facilities nearby at node n for all $n \in \mathcal{N}$ and $i \in \mathcal{I}$
$b_{n,i}$	Influence coefficient of the number of charging facilities on the charging demand of zone i during the busiest K hours at node n for all $n \in \mathcal{N}$ and $i \in \mathcal{I}$

$\theta_{n,i}$	Coverage target for zone i at node n for all $n \in \mathcal{N}$ and $i \in \mathcal{I}$
$e_{i,j}$	$e_{i,j} = e^{-a_i d_{i,j}}$ for a constant $a_i > 0$ and the distance $d_{i,j}$ between zone i and the candidate location j for all $i \in \mathcal{I}$ and $j \in \mathcal{J}$
C_j	Maximum units of charging demand a charger at candidate location j can serve during the busiest K hours for all $j \in \mathcal{J}$

Variables	**Descriptions**
$x_{n,j}$	Binary variable, $x_{n,j} = 1$ if and only if there exists a charging station in location j at node n for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$
$y_{n,j}$	Integer variable, the number of chargers in the charging station located in j at node n for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$
$\alpha_{n,i,j}$	Probability of drivers in zone i going to charging station located in j to charge their EVs at node n for all $n \in \mathcal{N}$, $i \in \mathcal{I}$ and $j \in \mathcal{J}$

2.3 Optimization model

The formulations of the EVCE model are as follows.

$$\min \sum_{n\in\mathcal{N}} \phi_n \sum_{j\in\mathcal{J}} \left[c^s_{n,j}(x_{n,j} - x_{a(n),j}) + c^p_{n,j}(y_{n,j} - y_{a(n),j}) + \bar{c}^s_{n,j} x_{n,j} + \bar{c}^p_{n,j} y_{n,j} \right] \quad (2.3.1a)$$

$$\text{s.t.} \sum_{i\in\mathcal{I}_{n,j}} \theta_{n,i} W_{n,i} \alpha_{n,i,j} \le C_j y_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \qquad (2.3.1b)$$

$$\sum_{j\in\mathcal{J}_{n,i}} x_{n,j} \ge 1 \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I} \qquad (2.3.1c)$$

$$\alpha_{n,i,j} \sum_{k \in \mathcal{J}_{n,i}} e_{i,k} x_{n,k} = e_{i,j} x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i} \tag{2.3.1d}$$

$$y_{n,j} \leq M_j x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.3.1e}$$

$$x_{a(n),j} \leq x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.3.1f}$$

$$y_{a(n),j} \leq y_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.3.1g}$$

$$x_{n,j} \in \{0,1\} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.3.1h}$$

$$y_{n,j} \in \mathbb{Z}^+ \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.3.1i}$$

The objective of this model is to minimize the expected total cost of building and operating the charging facilities. The constraint (2.3.1b) ensures that the capacities of charging stations are enough to satisfy the coverage targets for each zone. Since the customer volume of a charging station is not average during a whole day, we choose the busiest K hours as the studied period. We suppose that the charging facilities are enough if the charging demands during the busiest K hours are satisfied. The constraint (2.3.1c) guarantees all the zones are covered by at least one charging station. The constraint (2.3.1d) is about drivers' estimated choice probabilities. In constraints (2.3.1e) , (2.3.1f), and (2.3.1g), we make sure there is no charger at any candidate location without a charging station and the charging stations and the chargers at any candidate locations do not disappear over time. This results in a MIP.

We know that the charging facilities in our charging network design are directly affected by the predicted charging demands of each zone. The number of charging facilities would also affect the charging demands. Since the difficulty to find a public

charging station is an important factor that people will consider while making purchases of EVs, the existence of charging facilities could directly stimulate people's willingness to buy EVs. On the other hand, a higher degree of market penetration of EVs will definitely increase the charging demands. We assume that the charging demand of a zone i is affected by the total number of charging stations near this zone within a distance $R_{n,i}$. The charging demands can be expressed as

$$W_{n,i} = w_{n,i} + b_{n,i} \sum_{j \in \mathcal{J}_{n,i}} x_{n,j}, \quad \forall n \in \mathcal{N}, \forall i \in \mathcal{I}, \tag{2.3.2}$$

where $w_{n,i}$ is the basic charging demand of zone i without the influence of the number of charging facilities at node n and $b_{n,i}$ is the influence coefficient of charging facilities on the charging demand of zone i at node n. Then the constraint (2.3.1b) becomes

$$\sum_{i \in \mathcal{I}_{n,j}} \theta_{n,i} \alpha_{n,i,j} \left(w_{n,i} + b_{n,i} \sum_{k \in \mathcal{J}_{n,i}} x_{n,k} \right) \leq C_j y_{n,j} \quad \forall n \in \mathcal{N}, \forall j \in \mathcal{J}. \tag{2.3.3}$$

2.4 Linearization

There are two non-linear constraints in the mixed-integer program presented before. In constraint (2.3.1d) and (2.3.3), both of their left-hand sides include the sum of several non-linear elements $\alpha_{n,i,j} \cdot x_{n,k}$ for $n \in \mathcal{N}, i \in \mathcal{I}, j \in \mathcal{J}_{n,i}$ and $k \in \mathcal{J}_{n,i}$. We can linearize the two constraints by introducing variables $z_{n,i,j,k} = \alpha_{n,i,j} \cdot x_{n,k}$ for all $n \in \mathcal{N}, i \in \mathcal{I}, j \in \mathcal{J}_{n,i}$ and $k \in \mathcal{J}_{n,i}$. Then the constraints become linear and four additional constraints are added to make sure that $z_{n,i,j,k} = \alpha_{n,i,j} \cdot x_{n,k}$ holds for both $x_{n,k} = 0$ and $x_{n,k} = 1$. For all $n \in \mathcal{N}, i \in \mathcal{I}, j \in \mathcal{J}_{n,i}$ and $k \in \mathcal{J}_{n,i}$, the additional

constraints are

$$z_{n,i,j,k} \leq x_{n,k} \tag{2.4.1a}$$

$$z_{n,i,j,k} \leq \alpha_{n,i,j} \tag{2.4.1b}$$

$$z_{n,i,j,k} \geq \alpha_{n,i,j} + x_{n,k} - 1 \tag{2.4.1c}$$

$$z_{n,i,j,k} \geq 0. \tag{2.4.1d}$$

When $x_{n,k} = 1$, the constraints (2.4.1b) and (2.4.1c) can make sure that $z_{n,i,j,k} = \alpha_{n,i,j} = \alpha_{n,i,j} \cdot x_{n,k}$. When $x_{n,k} = 0$, the constraints (2.4.1a) and (2.4.1d) guarantee that $z_{n,i,j,k} = 0 = \alpha_{n,i,j} \cdot x_{n,k}$.

After linearization, the model becomes

$$\min \sum_{n\in\mathcal{N}} \phi_n \sum_{j\in\mathcal{J}} [c^s_{n,j}(x_{n,j} - x_{a(n),j}) + c^p_{n,j}(y_{n,j} - y_{a(n),j}) + \bar{c}^s_{n,j} x_{n,j} + \bar{c}^p_{n,j} y_{n,j}] \tag{2.4.2a}$$

$$\text{s.t.} \sum_{i\in\mathcal{I}_{n,j}} \theta_{n,i}(w_{n,i}\alpha_{n,i,j} + b_{n,i} \sum_{k\in\mathcal{J}_{n,i}} z_{n,i,j,k}) \leq C y_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.4.2b}$$

$$\sum_{k\in\mathcal{J}_{n,i}} x_{n,k} \geq 1 \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I} \tag{2.4.2c}$$

$$\sum_{k\in\mathcal{J}_{n,i}} e_{i,k} z_{n,i,j,k} = e_{i,j} x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i} \tag{2.4.2d}$$

$$z_{n,i,j,k} \leq x_{n,k} \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i}\ \forall k \in \mathcal{J}_{n,i} \tag{2.4.2e}$$

$$\text{(EVCE)}\ z_{n,i,j,k} \leq \alpha_{n,i,j} \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i}\ \forall k \in \mathcal{J}_{n,i} \tag{2.4.2f}$$

$$z_{n,i,j,k} \geq \alpha_{n,i,j} + x_{n,k} - 1 \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i}\ \forall k \in \mathcal{J}_{n,i} \tag{2.4.2g}$$

$$y_{n,j} \leq M_j x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.4.2h}$$

$$x_{a(n),j} \leq x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.4.2i}$$

$$y_{a(n),j} \leq y_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.4.2j}$$

$$x_{n,j} \in \{0,1\} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.4.2k}$$

$$y_{n,j} \in \mathbb{Z}^+ \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{2.4.2l}$$

$$z_{n,i,j,k} \geq 0 \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i}\ \forall k \in \mathcal{J}_{n,i} \tag{2.4.2m}$$

where constraints (2.4.2e), (2.4.2f), (2.4.2g) and (2.4.2m) are linearization constraints. Then, we get a mixed-integer linear program.

Since both the number of variables $\alpha_{n,i,j}$ and $z_{n,i,j,k}$ for $n \in \mathcal{N}$, $i \in \mathcal{I}$, $j \in \mathcal{J}_{n,i}$ and $k \in \mathcal{J}_{n,i}$ and the number of constraints are quite large for any reasonable $|\mathcal{N}|$, $|\mathcal{I}|$ and $|\mathcal{J}|$, it will cost much time to solve this EVCE model directly by a MILP solver, such as CPLEX MIP solver. Therefore, algorithms are designed to solve this program.

Chapter 3

Two Simple Algorithms

In this chapter, two simple algorithms are presented to solve the model of EVCE, an approximation algorithm and a heuristic algorithm. The approximation algorithm can give a feasible solution and a lower bound of the model within a relatively short time compared to solving the model directly by the CPLEX MIP solver. The heuristic algorithm will generate a feasible solution in seconds, but there is no lower bound obtained to estimate the gap between the heuristic solution and the optimal solution. It is noted that not only the two algorithms can give a feasible solution to the model, but the properties they present will also play an important role in the branch-and-price algorithm design which is studied in the subsequent chapter.

3.1 An Approximation Algorithm

For EVCE model, it is fair to say that the difficulty of solving this problem is mainly caused by $x_{n,j}$ which is a binary variable and $y_{n,j}$ which is an integer variable for $n \in \mathcal{N}$ and $j \in \mathcal{J}$. The idea of the approximation algorithm is to solve the linear

relaxation of the model to get a solution and then modify the solution in some way to make it feasible to the original EVCE model.

3.1.1 Integer Variables

Let $(\boldsymbol{x}, \boldsymbol{y}) = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) \mid \forall n \in \mathcal{N}\}$ and $\boldsymbol{x}_n = \{x_{n,j} \mid \forall j \in \mathcal{J}\}$, $\boldsymbol{y}_n = \{y_{n,j} \mid \forall j \in \mathcal{J}\}$ for all $n \in \mathcal{N}$. The MILP is rewritten in a simpler form to show the special structure and useful properties of the EVCE model.

$$\min \quad f(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n \in \mathcal{N}} \sum_{j \in \mathcal{J}} \left(c^{(1)}_{n,j} x_{n,j} + c^{(2)}_{n,j} y_{n,j} \right) - \Psi \tag{3.1.1a}$$

$$\text{s.t.} \quad y_{n,j} \geq g_{n,j}(\boldsymbol{x}_n) \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.1b}$$

$$y_{n,j} \leq M_j x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.1c}$$

$$\text{(EVCE)} \quad \sum_{j \in \mathcal{J}_{n,i}} x_{n,j} \geq 1 \qquad \forall n \in \mathcal{N}\ \forall i \in \mathcal{I} \tag{3.1.1d}$$

$$x_{a(n),j} \leq x_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.1e}$$

$$y_{a(n),j} \leq y_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.1f}$$

$$x_{n,j} \in \{0,1\}, y_{n,j} \in \mathbb{Z}^+ \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.1g}$$

where in the objective function $c^{(1)}_{n,j} = \phi_n(c^s_{n,j} + \bar{c}^s_{n,j}) - \sum_{n' \in s(n)} \phi_{n'} c^s_{n',j}$, $c^{(2)}_{n,j} = \phi_n(c^p_{n,j} + \bar{c}^p_{n,j}) - \sum_{n' \in s(n)} \phi_{n'} c^p_{n',j}$ and $\Psi = \phi_1 \sum_{j \in \mathcal{J}} (c^s_{1,j} x_{0,j} + c^p_{1,j} y_{0,j})$ and in the first constraint

$$g_{n,j}(\boldsymbol{x}_n) = \frac{1}{C_j} \left(\sum_{i \in \mathcal{I}_{n,j}} \theta_{n,i} \Big(w_{n,i} \alpha_{n,i,j} + b_{n,i} \sum_{k \in \mathcal{J}_{n,i}} z_{n,i,j,k} \Big) \right)$$

with $\alpha_{n,i,j} = \dfrac{e_{i,j} x_{n,j}}{\sum_{k \in \mathcal{J}_{n,i}} e_{i,k} x_{n,k}}$ and $z_{n,i,j,k} = \alpha_{n,i,j} x_{n,k}, \forall i \in \mathcal{I}, j \in \mathcal{J}$.

Let REVCE denote the relaxation of EVCE model by changing (3.1.1g) from

$y_{n,j} \in \mathbb{Z}^+$ to $y_{n,j} \geq 0$.

Theorem 3.1.1 *If* $(\boldsymbol{x}, \boldsymbol{y_L}) = \{(x_{n,j}, y^L_{n,j}) \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is a feasible solution of REVCE, then* $(\boldsymbol{x}, \boldsymbol{y_I}) = \{(x_{n,j}, y^I_{n,j}) \mid y^I_{n,j} = \lceil y^L_{n,j} \rceil, \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is a feasible solution of EVCE.*

Proof. For REVCE, the constraints (3.1.1d), (3.1.1e), (3.1.1g) are automatically satisfied. Other constraints are also obviously satisfied by $(\boldsymbol{x}, \boldsymbol{y_I})$ since

$$y^I_{n,j} = \lceil y^L_{n,j} \rceil \geq y^L_{n,j} \geq g_{n,j}(\boldsymbol{x}_n),$$

$$y^I_{n,j} = \lceil y^L_{n,j} \rceil \leq \lceil M_j x_{n,j} \rceil = M_j x_{n,j}$$

and

$$y^I_{a(n),j} = \lceil y^L_{a(n),j} \rceil \leq \lceil y^L_{n,j} \rceil = y^I_{n,j}.$$

□

Let EVCE($\bar{\boldsymbol{x}}$) and REVCE($\bar{\boldsymbol{x}}$) denote the programs where $x_{n,j}$ is given and fixed to $\bar{x}_{n,j}$ for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$ in EVCE and REVCE, respectively. These two programs are further simplified as

$$\begin{aligned}
\min \quad & f_{\bar{x}}(\boldsymbol{y}) = \sum_{n \in \mathcal{N}} \sum_{j \in \mathcal{J}} c^{\bar{x}}_{n,j} y_{n,j} + \Psi_{\bar{x}} && && \text{(3.1.2a)} \\
\text{s.t.} \quad & L^{\bar{x}}_{n,j} \leq y_{n,j} \leq U^{\bar{x}}_{n,j} && \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} && \text{(3.1.2b)} \\
(\text{EVCE}(\bar{\boldsymbol{x}})) \quad & y_{a(n),j} \leq y_{n,j} && \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} && \text{(3.1.2c)} \\
& y_{n,j} \in \mathbb{Z}^+ && \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} && \text{(3.1.2d)}
\end{aligned}$$

and

$$\min \quad f_{\bar{x}}(\boldsymbol{y}) = \sum_{n\in\mathcal{N}}\sum_{j\in\mathcal{J}} c_{n,j}^{\bar{x}} y_{n,j} + \Psi_{\bar{x}} \tag{3.1.3a}$$

$$\text{s.t.} \quad L_{n,j}^{\bar{x}} \le y_{n,j} \le U_{n,j}^{\bar{x}} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.3b}$$

$$(\text{REVCE}(\bar{\boldsymbol{x}})) \quad y_{a(n),j} \le y_{n,j} \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.3c}$$

$$y_{n,j} \ge 0 \qquad \forall n \in \mathcal{N}\ \forall j \in \mathcal{J} \tag{3.1.3d}$$

by setting

$$L_{n,j}^{\bar{x}} = g_{n,j}(\bar{\boldsymbol{x}}_n) \ge 0$$

$$U_{n,j}^{\bar{x}} = M_j \bar{x}_{n,j} \in \mathbb{Z}^+$$

and denoting the coefficient of $y_{n,j}$ in the objective function as $c_{n,j}^{\bar{x}}$.

Theorem 3.1.2 *If* $\boldsymbol{y_L^*} = \{y_{n,j}^* \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is the optimal solution of REVCE(*$\bar{\boldsymbol{x}}$*), then* $\boldsymbol{y_I^*} = \{\lceil y_{n,j}^* \rceil \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is an optimal solution of EVCE(*$\bar{\boldsymbol{x}}$*).*

The proof is omitted here and can be found in Appendix B.

3.1.2 Approximation Algorithm

Inspired by the special properties shown in Theorem 3.1.1 and Theorem 3.1.2, we designed a simple approximation algorithm (Algorithm 1). Let p^{EVCE} and p^{REVCE} denote an instance of the EVCE and the REVCE, respectively.

Algorithm 1 Approximation algorithm for EVCE

Input: p^{EVCE}

Output: Approximation solution $(\boldsymbol{x_I}, \boldsymbol{y_I})$

1: $p^{REVCE} \leftarrow p^{EVCE}$ ▷ Remove the integrality limitation of variable $\boldsymbol{y}$

2: $(\boldsymbol{x_I}, \boldsymbol{y_L}) = solve(p^{REVCE})$

3: $\boldsymbol{y_I} = \{\lceil y_{n,j} \rceil \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\} \leftarrow \boldsymbol{y_L} = \{y_{n,j} \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$

4: **return** $(\boldsymbol{x_I}, \boldsymbol{y_I})$

This approximation algorithm is very easy to implement and can save a lot of time. During the implementation of the algorithm, the optimal solution of REVCE is obtained as $(\boldsymbol{x_I}, \boldsymbol{y_L})$. We can get a lower bound of EVCE from this solution.

Theorem 3.1.3 *If* $(\boldsymbol{x_I}, \boldsymbol{y_L}) = \{(x^I_{n,j}, y^L_{n,j}) \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is the optimal solution of REVCE,* $(\boldsymbol{x_I}, \boldsymbol{y_I}) = \{(x^I_{n,j}, y^I_{n,j}) \mid y^I_{n,j} = \lceil y^L_{n,j} \rceil, \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is a feasible solution of EVCE and* $(\boldsymbol{x}^*, \boldsymbol{y}^*)$ *is the optimal solution of EVCE, then*

$$f(\boldsymbol{x_I}, \boldsymbol{y_L}) \leq f(\boldsymbol{x}^*, \boldsymbol{y}^*) \leq f(\boldsymbol{x_I}, \boldsymbol{y_I}).$$

Proof. It is obvious that $f(\boldsymbol{x}^*, \boldsymbol{y}^*) \leq f(\boldsymbol{x_I}, \boldsymbol{y_I})$ since EVCE is a minimization problem. Because REVCE is the relaxation form of EVCE which is obtained by removing the integer constraints for the decision variables $y_{n,j}, \forall n \in \mathcal{N}, \forall j \in \mathcal{J}$, the optimal objective value of REVCE is less than or equal to the optimal objective value of EVCE. Thus $f(\boldsymbol{x_I}, \boldsymbol{y_L}) \leq f(\boldsymbol{x}^*, \boldsymbol{y}^*)$. □

Therefore, we can get not only an approximation solution $(\boldsymbol{x_I}, \boldsymbol{y_I})$ but also a lower bound $f(\boldsymbol{x_I}, \boldsymbol{y_L})$ for EVCE by Algorithm 1.

3.2 A Heuristic Algorithm

This heuristic algorithm is based on the special structure of the EVCE model. The constraints of the linear program (3.1.1) will be divided into $n+1$ groups, including n subproblem groups and a coupling constraint group. For each $n \in \mathcal{N}$, the subproblem group SP(n) has constraints $y_{n,j} \geq g_{n,j}(\boldsymbol{x}_n)$, $y_{n,j} \leq M_j x_{n,j}$ and $\sum_{j \in \mathcal{J}_{n,i}} x_{n,j} \geq 1$, $\forall j \in \mathcal{J}$. A solution $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ which satisfies all constraints in SP(n) will be guaranteed that every zone is covered by at least one charging station and the capacity of any charging station is enough for the charging demand at scenario node n. The remaining constraints are coupling constraints which make sure that charging stations and chargers will not disappear over time.

The main idea of the heuristic algorithm is to generate a feasible solutions $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ for each SP(n) and make sure that these solutions do not go against the principle that stations and chargers won't disappear over time. Then $(\boldsymbol{x}, \boldsymbol{y}) = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) \mid \forall n \in \mathcal{N}\}$ will make a heuristic solution of EVCE.

In fact, it is quite easy to generate a feasible $\boldsymbol{x}_n$ for SP(n). Since the capacity of any charging station is unlimited, the only limitation we shall consider about $\boldsymbol{x}_n$ is to cover all the zones by charging stations at scenario node n. For example, we can randomly select a candidate location j which can cover at least one uncovered zone and set x_j to 1, and then repeat this procedure until all the zones are covered. Of course, we can try to get a better solution by applying a specific rule instead of picking it randomly when choosing a candidate location.

Let $\mathcal{J}_n^{fix0}$ and $\mathcal{J}_n^{fix1}$ denote two disjoint subsets of $\mathcal{J}$, $\forall n \in \mathcal{N}$. It is not hard to generate a solution $\boldsymbol{x}_n$ for SP(n) greedily under the condition that for any $j \in \mathcal{J}_n^{fix0}$, $x_{n,j}$ is fixed to 0 and for any $j \in \mathcal{J}_n^{fix1}$, $x_{n,j}$ is fixed to 1. The GREEDY function is

explained in Algorithm 2.

Algorithm 2 Generate a feasible $\boldsymbol{x}_n$ for SP(n) greedily

Input: SP(n), $\mathcal{J}_n^{fix0}$, $\mathcal{J}_n^{fix1}$

Output: $\boldsymbol{x_n}$

1: **function** GREEDY(SP(n), $\mathcal{J}_n^{fix0}$, $\mathcal{J}_n^{fix1}$)
2: $x_{n,j} := 1, \forall j \in \mathcal{J}_n^{fix1}$
3: $x_{n,j} := 0, \forall j \in \mathcal{J} \setminus \mathcal{J}_n^{fix1}$
4: $\mathcal{J}_n^{now} := \mathcal{J} \setminus (\mathcal{J}_n^{fix0} \bigcup \mathcal{J}_n^{fix1})$ ▷ Set of available candidate locations
5: $\mathcal{I}_n^{covered} := \{i \in \mathcal{I} \mid \exists j \text{ s.t. } x_{n,j} = 1 \text{ and } d_{i,j} \leq R_{n,i}\}$
6: $\mathcal{I}_n^{now} := \mathcal{I} \setminus \mathcal{I}_n^{covered}$ ▷ Set of temporarily uncovered zones
7: **while** $\mathcal{I}_n^{now}$ is not empty **do**
8: $\mathcal{J}_n^{now} \leftarrow \mathcal{J}_n^{now} \setminus \{j \mid d_{i,j} > R_{n,i}, \forall i \in \mathcal{I}_n^{now}\}$ ▷ Delete useless locations
9: **if** $\mathcal{J}_n^{now}$ is empty **then**
10: **return** No feasible solution exists!
11: **end if**
12: Choose a j_0 in $\mathcal{J}_n^{now}$ ▷ Choose randomly or under a specific rule
13: $x_{n,j_0} := 1$
14: $\mathcal{J}_n^{now} \leftarrow \mathcal{J}_n^{now} \setminus \{j_0\}$
15: $\mathcal{I}_n^{now} \leftarrow \mathcal{I}_n^{now} \setminus \{i \mid d_{i,j_0} \leq R_{n,i}\}$ ▷ Delete covered zones
16: **end while**
17: **return** $\boldsymbol{x}_n = \{x_{n,j} \mid j \in \mathcal{J}\}$
18: **end function**

At step 12 in the GREEDY function, the candidate location j_0 is selected randomly

or under a specific rule. A recommended rule is to choose the candidate location with lowest $\frac{1}{n_z}\left(c^s_{n,j} + \bar{c}^s_{n,j} + \tilde{y}(c^s_{n,j} + c^p_{n,j})\right)$, where n_z is the number of zones that j_0 covers in $\mathcal{I}_{now}$ and $\bar{y}$ is a parameter set to be as close as possible to the average number of chargers a charging station holds.

For any SP(n), once a feasible $\boldsymbol{x}_n$ is given, it is not difficult to get a feasible $\boldsymbol{y}_n$ for this $\boldsymbol{x}_n$. The generation of $\boldsymbol{y}_n = \bar{g}(\boldsymbol{x}_n)$ can be expressed as

$$y_{n,j} = \bar{g}_{n,j}(\boldsymbol{x}_n) = \lceil g_{n,j}(\boldsymbol{x}_n) \rceil, \forall j \in \mathcal{J}.$$

It is obvious that $y_{n,j} = 0$ when $x_{n,j} = 0$ and $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ is feasible to SP(n).

To make sure the final solution for EVCE, which is an MILP, is not against the principle that stations and chargers can not disappear over time, the heuristic algorithm constructs $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ based on the assignment at its father node $(\boldsymbol{x}_{a(n)}, \boldsymbol{y}_{a(n)})$. Specifically, $x_{n,j}$ is fixed to 1 manually if $x_{a(n),j}$ is 1 and $y_{n,j}$ will take the maximum value from $\bar{g}_{n,j}(\boldsymbol{x}_n)$ and $y_{a(n),j}$. For any given feasible solution of SP(1) at the root node of the scenario tree, a feasible solution of SP(n) is generated based on the assignment of SP($a(n)$) by the GREEDY function in Algorithm 2 node by node from top of the scenario tree to the end. Then, a final feasible solution for EVCE is generated by combining $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ for all $n \in \mathcal{N}$.

The detailed heuristic algorithm is as follows.

Algorithm 3 Heuristic Algorithm

Input: p, an instance of EVCE

Output: $(\boldsymbol{x}_{heur}, \boldsymbol{y}_{heur}) = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) \mid \forall n \in \mathcal{N}\}$

1: $\mathcal{J}_n^{fix0}(p) := \{j \in \mathcal{J} \mid x_{n,j}$ is fixed to 0 in p at node $n\ \}, \forall n \in \mathcal{N} \setminus \{1\}$

2: $\mathcal{J}_n^{fix1}(p) := \{j \in \mathcal{J} \mid x_{n,j} \text{ is fixed to 1 in } p \text{ at node } n\ \}, \forall n \in \mathcal{N} \setminus \{1\}$

3: Generate several solutions $\{\boldsymbol{x}_1^q \mid 1 \leq q \leq Q\}$ for SP(1) of p

4: **for** every q from 1 to Q **do**

5: $\quad \boldsymbol{y}_1^q \leftarrow \bar{g}(\boldsymbol{x}_1^q)$

6: $\quad$ **for** every $n \in \mathcal{N} \setminus \{1\}$ **do** ▷ In a order that n is not earlier than $a(n)$

7: $\quad\quad \mathcal{J}_n^{q,fix0} := \mathcal{J}_n^{fix0}(p)$

8: $\quad\quad \mathcal{J}_n^{q,fix1} := \mathcal{J}_n^{fix1}(p) \bigcup \{j \mid x_{a(n),j}^q = 1\}$

9: $\quad\quad \boldsymbol{x}_n^q := GREEDY(SP(n), \mathcal{J}_n^{q,fix0}, \mathcal{J}_n^{q,fix1})$

10: $\quad\quad y_{n,j}^q := \max\{y_{a(n),j}^q, \bar{g}_{n,j}(\boldsymbol{x}_n^q)\}, \forall j \in \mathcal{J}$

11: $\quad$ **end for**

12: $\quad$ Calculate the objective value $f(\boldsymbol{x}^q, \boldsymbol{y}^q)$ where $(\boldsymbol{x}^q, \boldsymbol{y}^q) = \{(\boldsymbol{x}_n^q, \boldsymbol{y}_n^q) \mid \forall n \in \mathcal{N}\}$

13: **end for**

14: Choose the solution $(\boldsymbol{x}^{q*}, \boldsymbol{y}^{q*})$ with lowest objective value among all Q feasible solutions

15: **return** $(\boldsymbol{x}_{heur}, \boldsymbol{y}_{heur}) = (\boldsymbol{x}^{q*}, \boldsymbol{y}^{q*})$

At step 3 in the heuristic algorithm (Algorithm 3), the method to generate Q feasible solutions $\{\boldsymbol{x}_1^q \mid 1 \leq q \leq Q\}$ for SP(1) is not mentioned. In fact, this step is very important to the whole algorithm, which directly affects the quality of the solution and the running time of the algorithm. Clearly, the more solutions given in this step for SP(1), the more likely the final solution to the problem is close to the optimal solution, but it also means that the running time will be longer and more computer memory is occupied.

A method we use to make a compromise between the number of solutions explored and the running time of the algorithm is explained in Algorithm 4 in Appendix A.

It combines the ideas of "greedy" and "enumeration" and was implemented by the recursive method. It chooses a candidate location greedily, but explores both the cases where a charging station is or is not built at this location. In each step, some "useless" candidate locations will be removed and will no longer be selected in the future. Here, a candidate location is useless if it covers none of currently uncovered zones. The removal largely reduces the number of the potential solutions.

3.3 Numerical Experiments

In this section, we generate some instances of different sizes with randomized data and test the efficiency of the approximation algorithm (Algorithm 1) and the heuristic algorithm (Algorithm 3). When generating the instances with randomized data, we limit the costs of building and operating stations and chargers in a reasonable range and guarantee the charging demand in each zone increases as time goes by. The scenario tree in Figure 3.1 is used in all instances. Table 3.1 shows the number of variables, constraints and non-zero elements for all instances we will use. Every instance has $|\mathcal{N}| \cdot |\mathcal{J}|$ binary variables ($x_{n,j}$) and the same number of integer variables ($y_{n,j}$). It is also shown in the table.

For the approximation algorithm (Algorithm 1), we show the solution time t_{appr}, the objective value of the approximation solution $z_{appr} = f(\boldsymbol{x_I}, \boldsymbol{y_I})$, the lower bound $LB_{appr} = f(\boldsymbol{x_I}, \boldsymbol{y_L})$ and the solution gap $GAP_{LB} = \dfrac{z_{appr} - LB_{appr}}{z_{appr}}$. For the heuristic algorithm (Algorithm 3), we show the solution time t_{heur} and the objective value of the heuristic solution $z_{heur} = f(\boldsymbol{x}_{heur}, \boldsymbol{y}_{heur})$. They are all compared with CPLEX MIP solver's solution time t^* and optimal objective value z^*. The time saving percentages

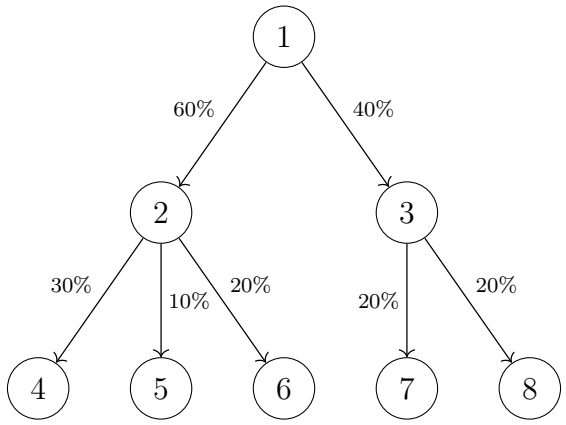

Figure 3.1: The scenario tree in numerical experiments

$TS_{appr} = \dfrac{t^* - t_{appr}}{t^*}$, $TS_{heur} = \dfrac{t^* - t_{heur}}{t^*}$ and objective gaps $GAP_{appr} = \dfrac{z_{appr} - z^*}{z_{appr}}$, $GAP_{heur} = \dfrac{z_{heur} - z^*}{z_{heur}}$ are also reported.

In Table 3.2 and Table 3.3, the maximal solution time is set to 7200 seconds. The CPLEX MIP solver or the algorithms are stopped when the time is out. The currently best solution is reported together with the current objective value gap for the CPLEX MIP solver. For the approximation algorithm (Algorithm 1), the currently best solution is reported. The current lower bound of REVCE is reported as the lower bound of EVCE. Let us take Instance 5 in Table 3.2 for an example. When solving Instance 5, the CPLEX MIP solver can not solve to optimality in 7200 seconds. It obtains a feasible solution with an objective value of 16,611 and a gap of 5.6%. The approximation algorithm gets the same feasible solution, but it also obtains a better lower bound 15,728 with a gap of 5.3% within 3,205 seconds. The approximation algorithm saves more than 55.5% of the time compared to the CPLEX MIP solver. We are not able to get the gap of the objective values between the approximation solution and the optimal solution since the latter is not found by the CPLEX MIP solver, but we can estimate the gap from the current lower bound and claim that the

Table 3.1: Scale of the instances

Instance Number	Instance Size $\|\mathcal{I}\|$	$\|\mathcal{J}\|$	$\|\mathcal{N}\|$	# of Var.	# of Con.	# of Non-Zeros	# of $x_{n,j}(y_{n,j})$
1	10	10	8	1,579	4,293	11,330	80
2	10	10	8	1,843	5,217	13,478	80
3	10	15	8	3,753	10,179	29,550	120
4	10	15	8	4,256	11,600	33,775	120
5	10	20	8	5,090	13,896	40,554	160
6	10	20	8	6,770	18,758	55,141	160
7	15	15	8	2,774	7,562	20,438	120
8	15	15	8	2,643	6,955	19,463	120
9	15	20	8	5,070	13,674	39,731	160
10	15	20	8	4,568	12,264	35,380	160
11	15	30	8	10,000	27,672	81,336	240
12	15	30	8	9,506	26,246	77,058	240

gap can not exceed 5.3%.

From Table 3.2, we can see the the approximation algorithm (Algorithm 1) will give an approximation solution which is very close to the optimal solution. The gap between the lower bound and the objective value of the approximation solution is no more than 10% in the tested instances, but this is not guaranteed since the results vary greatly for different instances. For the last two instances where both the CPLEX MIP solver and the approximation algorithm can not get the optimal solution in 7200 seconds, the approximation algorithm obtain better solutions than the CPLEX MIP solver.

In Table 3.3, the heuristic algorithm (Algorithm 3) gets feasible and relatively good solutions in seconds. But since it does not offer any lower bound for the problem, the gap between the heuristic solution and the optimal solution is not obtained during the algorithm process. So it is best to use the heuristic solution as an initial feasible

solution for some other optimization algorithms, for example, the branch-and-price which we will investigate in the next chapter.

Table 3.2: Efficiency of the approximation algorithm (Algorithm 1)

Inst.	Inst. Size		CPLEX MIP solver		Algorithm 1				Comparison	
#	$\|\mathcal{I}\|$	$\|\mathcal{J}\|$	$t^*(s)$	z^*	$t_{appr}(s)$	z_{appr}	LB_{appr}	GAP_{LB}	TS_{appr}	GAP_{appr}
1	10	10	15	18,648	2	18,648	17,452	6.4%	86.7%	0%
2	10	10	42	19,765	7	19,797	18,695	5.6%	83.3%	0.2%
3	10	15	2,579	16,683	214	16,683	16,005	4.1%	91.7%	0%
4	10	15	468	16,376	200	16,376	15,586	4.8%	57.2%	0%
5	10	20	$> 7{,}200$	16,611(5.6%)	3,205	16,611	15,728	5.3%	$> 55.5\%$	$\leq 5.3\%$
6	10	20	$> 7{,}200$	17,546(6.6%)	4,650	17,771	16,456	7.4%	$> 35.4\%$	$\leq 7.4\%$
7	15	15	192	27,177	10	27,177	25,596	5.8%	94.8%	0%
8	15	15	530	26,335	54	26,335	24,945	5.3%	89.8%	0%
9	15	20	3,924	26,186	929	26,186	24,662	5.8%	76.3%	0%
10	15	20	$> 7{,}200$	27,048(6.7%)	2,196	27,254	25,327	7.1%	$> 69.5\%$	$\leq 7.1\%$
11	15	30	$> 7,200$	27,332(35.2%)	$> 7,200$	25,909	17,179	33.7%	-	$\leq 31.7\%$
12	15	30	$> 7,200$	26,125(26.2%)	$> 7,200$	26,032	19,096	26.6%	-	$\leq 25.9\%$

Table 3.3: Efficiency of the heuristic algorithm (Algorithm 3)

Inst.	Inst. Size		CPLEX MIP solver		Algorithm 3		Comparison	
#	$\|\mathcal{I}\|$	$\|\mathcal{J}\|$	$t^*(s)$	z^*	$t_{heur}(s)$	z_{heur}	TS_{heur}	GAP_{heur}
1	10	10	15	18,648	0.010	18,957	99.9%	1.6%
2	10	10	42	19,765	0.001	19,765	100%	0%
3	10	15	2,579	16,683	0.026	17,051	100%	2.2%
4	10	15	468	16,376	0.041	16,430	100%	0.3%
5	10	20	$> 7,200$	16,611(5.6%)	0.110	16,914	100%	$\leq 7.3\%$
6	10	20	$> 7,200$	17,546(6.6%)	0.085	17,546	100%	$\leq 6.6\%$
7	15	15	192	27,177	0.045	27,177	100%	0%
8	15	15	530	26,335	0.011	26,979	100%	2.4%
9	15	20	3,924	26,186	0.443	26,186	100%	0%
10	15	20	$> 7,200$	27,048(6.7%)	0.111	27,307	100%	$\leq 7.3\%$
11	15	30	$> 7,200$	27,332(35.2%)	8.175	26,009	$> 99.9\%$	$\leq 31.9\%$
12	15	30	$> 7,200$	26,125(26.2%)	7.438	26,032	$> 99.9\%$	$\leq 25.9\%$

Chapter 4

Branch-and-Price Algorithm

In this chapter, a branch-and-price algorithm is designed for EVCE. This algorithm is based on the branch-and-bound algorithm and uses column generation method to solve the linear program at each node in the branch-and-bound tree. We also performed the Dantzig-Wolfe decomposition procedure. All these lead to a branch-and-price algorithm, which transforms the original EVCE model into several smaller subproblems in order to reduce the solution time.

4.1 An Overview of Algorithms for Mixed Integer Linear Program

Linear programming is a technique for the optimization of a linear objective function, subject to linear constrains. Linear programs can be transformed in some ways and then expressed as

$$\min \quad \boldsymbol{c}^T \boldsymbol{x} \tag{4.1.1a}$$

$$\text{s.t.} \quad A\boldsymbol{x} = \boldsymbol{b} \tag{4.1.1b}$$

$$\boldsymbol{x} \geq 0 \tag{4.1.1c}$$

The most important algorithms to solve linear programming are Simplex algorithm and Interior point methods. A brief introduction about Simplex is presented below since it provides the basic theory of the algorithms we will use later. We talk only about the bounded liner programs here.

4.1.1 Simplex Algorithm

The Simplex algorithm is a basis exchange algorithm. It solves linear programs by constructing a series of basic feasible solutions which are corners (extreme points) of the polyhedron composed of all feasible solutions. Since an optimal basic feasible solution always exists for the linear program which has an optimal solution, it is sufficient to consider the basic feasible solutions only. The Simplex algorithm travels along a path on the edges of the polytope to the corners with non-decreasing objective values until an optimal basic feasible solution is found.

For any basic feasible solution, all non-basic variables are set to 0. Let us suppose that $\boldsymbol{x}_B$ are basic variables and $\boldsymbol{x}_N$ are non-basic variables of a basic feasible solution. Then $\boldsymbol{c}_B$ and $\boldsymbol{c}_N$ are their coefficients in the objective function, respectively. The coefficient matrix is divided into two matrices B and N according to the basic and non-basic variables, where B is always non-singular. The linear program is also expressed as

$$\min \quad \boldsymbol{c}_B^T\boldsymbol{x}_B + \boldsymbol{c}_N^T\boldsymbol{x}_N \tag{4.1.2a}$$

$$\text{s.t.} \quad B\boldsymbol{x}_B + N\boldsymbol{x}_N = \boldsymbol{b} \tag{4.1.2b}$$

$$\boldsymbol{x} \geq 0. \tag{4.1.2c}$$

Let $\boldsymbol{\pi} = \boldsymbol{c}_B^T B^{-1}$, the linear program will be expressed even further as

$$\min \quad \mathbf{0}^T \boldsymbol{x}_B + (\boldsymbol{c}_N^T - \boldsymbol{\pi} N)\boldsymbol{x}_N - \boldsymbol{\pi}\boldsymbol{b} \tag{4.1.3a}$$

$$\text{s.t.} \quad \boldsymbol{x}_B + B^{-1}N\boldsymbol{x}_N = B^{-1}\boldsymbol{b} \tag{4.1.3b}$$

$$\boldsymbol{x} \geq 0. \tag{4.1.3c}$$

The basic feasible solution is $\boldsymbol{x}_B = B^{-1}\boldsymbol{b} \geq 0$ and $\boldsymbol{x}_N = \mathbf{0}$. The constrains are obviously satisfied and the objective value of this basic feasible solution is $-\boldsymbol{\pi}\boldsymbol{b}$. If there is at least one element of $\boldsymbol{c}_N^T - \boldsymbol{\pi}N$ with a negative value, increasing the corresponding non-basic variable from 0 to a positive number will decrease the objective function value. The values of basic variables can be adjusted at the same time to make sure the constraints are satisfied if the amount of growth of the non-basic variable is within a certain range. A better solution is then obtained for this linear program. Therefore, it is important to check the value of $\boldsymbol{c}_N^T - \boldsymbol{\pi}N$. The multiplier $\boldsymbol{\pi}$ is called the dual solution of the program and $\boldsymbol{c}_N^T - \boldsymbol{\pi}N$ are called the reduced costs of non-basic variables (Bradley *et al.*, 1977). A basic feasible solution is optimal if and only if the reduced costs are all non-negative.

The set of all basic variables is called the basis of a linear program. The final goal of the Simplex algorithm is to find out the basis for which all the reduced costs are non-negative. This idea is used in the column generation method, which is a popular method for solving large-scale linear programs.

There are always applied problems which have too many variables and constraints and they greatly exceed the computational limit at current time no matter how fast the computer capabilities grow. Two methods are presented to help solve large-scale linear programs, column generation and decomposition.

4.1.2 Column Generation

The column generation method is based on the theory of the basis and a representation property.

Theorem 4.1.1 (Representation Property) *Let $\boldsymbol{x}^1$, $\boldsymbol{x}^2$, ..., $\boldsymbol{x}^K$ be the extreme points of the feasible region of the linear program (4.1.1) and assume that the points in this feasible region are bounded. Then any feasible solution $\boldsymbol{x}$ can be expressed as a convex combination of the extreme points as*

$$\boldsymbol{x} = \lambda_1 \boldsymbol{x}^1 + \lambda_2 \boldsymbol{x}^2 + \cdots + \lambda_K \boldsymbol{x}^K$$

with

$$\lambda_1 + \lambda_2 + \cdots + \lambda_K = 1, \quad \lambda_k \geq 0 \quad (k = 1, 2, \ldots, K)$$

Suppose that we have got all the extreme points of linear program (4.1.1) as $\{\boldsymbol{x}^q, q = 1, 2, \cdots, Q\}$. The program is reformulated as

$$\min \quad \sum_{q=1}^{Q} (\boldsymbol{c}^T \boldsymbol{x}^q) \lambda_q \tag{4.1.4a}$$

$$\text{s.t.} \quad \sum_{q=1}^{Q} (A\boldsymbol{x}^q) \lambda_q = \boldsymbol{b} \tag{4.1.4b}$$

$$\sum_{q=1}^{Q} \lambda_q = 1 \tag{4.1.4c}$$

$$\lambda_q \geq 0, \quad q = 1, 2, \cdots, Q \tag{4.1.4d}$$

This program is totally equivalent to the original linear program, but the variables are $\lambda_q, 1 \leq q \leq Q$. In this program, all extreme points are called columns. Each column is a feasible solution of the original program and has a corresponding variable λ_q for some q. Usually, the number of columns are very large and thus there are too many variables in this program. Most of the variables are non-basic, which will take the value 0 in an optimal basic feasible solution.

The main idea of the column generation method is to generate a column pool with limited number of columns and consider only these columns in the program. Other variables are set to 0 automatically if their columns are not included in the column pool. An optimal solution is found when all variables whose columns are not in the column pool have non-negative reduced costs. Before reaching the optimal solution, columns with negative reduced costs are continuously added to the column pool until no such column is found. The details are omitted here since we will explain it later in the context of the Dantzig-Wolfe decomposition.

4.1.3 Dantzig-Wolfe Decomposition

The Dantzig-Wolfe decomposition is used when the coefficient matrix of the linear program is large, sparse and has special block structures. The blocks help divide the decision variables and constraints into several disjoint sets. Each block forms a subproblem where only a set of variables and a set of constraints are involved.

A typical linear program for the Dantzig-Wolfe decomposition is expressed as

$$\min\ (\boldsymbol{c}_1)^T\boldsymbol{x}_1 + (\boldsymbol{c}_2)^T\boldsymbol{x}_2 + \cdots + (\boldsymbol{c}_K)^T\boldsymbol{x}_K \tag{4.1.5a}$$

$$\text{s.t.} \quad D_1\boldsymbol{x}_1 \leq \boldsymbol{d}_1 \tag{4.1.5b}$$

$$D_2\boldsymbol{x}_2 \leq \boldsymbol{d}_2 \tag{4.1.5c}$$

$$\cdots \leq \cdots \tag{4.1.5d}$$

$$D_K\boldsymbol{x}_K \leq \boldsymbol{d}_K \tag{4.1.5e}$$

$$A_1\boldsymbol{x}_1 + A_2\boldsymbol{x}_2 + \cdots + A_K\boldsymbol{x}_K = \boldsymbol{b} \tag{4.1.5f}$$

$$\boldsymbol{x}_k \in \mathbb{R}_+^{n_k}, \quad k = 1, 2, \cdots, K. \tag{4.1.5g}$$

Then a feasible solution $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_K)$ is generated by the solutions of the subproblems $\{\boldsymbol{x}_k \mid D_k\boldsymbol{x}_k \leq \boldsymbol{d}_k\}$ which satisfy the coupling constraint (4.1.5f).

The column generation method is well used based on the Dantzig-Wolfe decomposition. First let us reformulate the program which includes only the coupling constraints. It is also called the Master Problem (MP).

$$\min\ \sum_{q=1}^{Q_1}(\boldsymbol{c}_1^T\boldsymbol{x}_1^q)\lambda_1^q + \sum_{q=1}^{Q_2}(\boldsymbol{c}_2^T\boldsymbol{x}_2^q)\lambda_1^q + \cdots + \sum_{q=1}^{Q_K}(\boldsymbol{c}_K^T\boldsymbol{x}_K^q)\lambda_K^q \tag{4.1.6a}$$

$$\text{s.t.}\ \sum_{q=1}^{Q_1}(A_1\boldsymbol{x}_1^q)\lambda_1^q + \sum_{q=1}^{Q_2}(A_2\boldsymbol{x}_2^q)\lambda_2^q + \cdots + \sum_{q=1}^{Q_K}(A_K\boldsymbol{x}_K^q)\lambda_K^q = \boldsymbol{b} \tag{4.1.6b}$$

$$\text{(MP)}\ \sum_{q=1}^{Q_k}\lambda_k^q = 1, \quad k = 1, 2, \cdots, K \tag{4.1.6c}$$

$$\lambda_k^q \geq 0, \quad \forall q = 1, \cdots, Q_k, \quad k = 1, 2, \cdots, K, . \tag{4.1.6d}$$

where $\{\boldsymbol{x}_k^q \mid q = 1, \cdots, Q_k\}$ are all the extreme points of subproblem $\{\boldsymbol{x}_k \mid D_k\boldsymbol{x}_k \leq$

$\boldsymbol{d}_k\}$ for all $k = 1, \cdots, K$. A restricted version of MP is called the Restricted Master Problem (RMP) which contains only limited number of columns in the column pools $\{\boldsymbol{x}_k^q \mid q \in \mathcal{Q}_k\}$ for $k = 1, 2, \cdots, K$.

$$\min \sum_{q\in\mathcal{Q}_1} (\boldsymbol{c}_1^T \boldsymbol{x}_1^q)\lambda_1^q + \sum_{q\in\mathcal{Q}_2} (\boldsymbol{c}_2^T \boldsymbol{x}_2^q)\lambda_1^q + \cdots + \sum_{q\in\mathcal{Q}_K} (\boldsymbol{c}_K^T \boldsymbol{x}_K^q)\lambda_K^q \tag{4.1.7a}$$

$$\text{s.t.} \sum_{q\in\mathcal{Q}_1} (A_1 \boldsymbol{x}_1^q)\lambda_1^q + \sum_{q\in\mathcal{Q}_2} (A_2 \boldsymbol{x}_2^q)\lambda_2^q + \cdots + \sum_{q\in\mathcal{Q}_K} (A_K \boldsymbol{x}_K^q)\lambda_K^q = \boldsymbol{b} \tag{4.1.7b}$$

$$\text{(RMP)} \quad \sum_{q\in\mathcal{Q}_k} \lambda_k^q = 1, \quad k = 1, 2, \cdots, K \tag{4.1.7c}$$

$$\lambda_k^q \geq 0, \quad \forall q \in \mathcal{Q}_k, \quad k = 1, 2, \cdots, K, . \tag{4.1.7d}$$

The subproblems are equipped with objective functions to get new columns with negative reduced costs or to prove that all reduced costs of the columns outside the column pools are non-negative. The subproblem k is

$$\min \quad (\boldsymbol{c}_k^T - \boldsymbol{\pi}_k A_k)\boldsymbol{x}_k - \mu_k \tag{4.1.8a}$$

$$\text{SP}(k) \qquad \text{s.t.} \quad D_k \boldsymbol{x}_k \leq \boldsymbol{d}_k \tag{4.1.8b}$$

$$\boldsymbol{x}_k \in \mathbb{R}_+^{n_k} \tag{4.1.8c}$$

where $\boldsymbol{\pi}_k$ and μ_k are dual solutions corresponding to the coupling constraints and the convex constraint in the MP, respectively. The columns with negative reduced costs got from the subproblems are gradually added to the column pool until the optimal solution of the original problem is reached when all the objective values of the subproblems are non-negative.

4.1.4 Branch-and-Bound

A very popular algorithm to solve the integer programming problems is the branch-and-bound method. In the branch-and-bound algorithm, the total set of feasible solutions are partitioned into smaller subsets of solutions by adding some constraints, usually fixing the value or range of an integer variable. Then the smaller subsets are searched systematically until the optimal solution is found. It is called branch-and-price because the most important elements are the branch operation and the bound operation.

The branch operation creates a rooted branch-and-bound tree. The root of the tree is the original problem together with the total set of feasible solutions. When fixing the value or range of an integer variable, several branches grow from its father node and these branches are subproblems with disjoint sets of solutions.

The algorithm explores branches of this branch-and-price tree to find the optimal solution. However, the algorithm would degenerate to a brute-force search without the bound operation. Any branch is checked before further exploration against upper and lower estimated bounds on the optimal solution and is "pruned" if it has no chance to produce a better solution. For a minimizing problem, a global upper bound is the objective value of the currently best feasible solution the algorithm has found so far. Each branch has its own lower bound which is the optimal objective value of the linear relaxation of the subproblem for this branch if it is feasible. Clearly, any feasible solution of the subproblem of the branch must have an objective value greater than or equal to the branch's lower bound. In this way, a branch is "pruned" if its lower bound is greater than the global upper bound since it is not promising to produce a better solution. A branch is also necessary to be "pruned" if its subproblem

is infeasible, i.e., its set of solution is empty.

Using the branch operation and bound operation, the algorithm searches from top to down of the branch-and-bound tree. Upon visiting a subproblem of a node in the branch-and-bound tree, the linear relaxation of the subproblem is solved to get the lower bound of this branch. The global upper bound and the currently best solution are both updated if the optimal solution of the linear relaxation of the subproblem is a better and feasible solution to the original problem. Only branches that are not "pruned" will be further explored. The global lower bound of the optimal solution is the smallest lower bound among all the existing branches. Clearly, the global upper bound will stay the same or decrease as the process of exploration. Since the lower bound of any node in the branch-and-bound tree will be greater than or equal to the lower bound of its father node, the global lower bound will stay the same or increase gradually as the process of the exploration. The algorithm can stop when the global upper and lower bound are close enough to each other.

4.1.5 Branch-and-Price

The branch-and-price algorithm is a hybrid of branch-and-bound and column generation methods. It uses the column generation method to help solve the linear relaxation of the program in each node of the branch-and-bound tree. In this thesis, we use the branch-and-price algorithm in which the column generation method is based on the Dantzig-Wolfe decomposition to solve our EVCE model.

Generally speaking, the Dantzig-Wolfe decomposition reformulates the original problem into a master problem and several pricing problems. A column in the master problem is a solution to one of the subproblem. The column generation method is

used to generate columns from the pricing problems and then add the columns to the master problem to improve the overall objective of the linear relaxation of the subproblem at the branch-and-bound tree.

Details about the branch-and-price algorithm is presented in the subsequent section.

4.2 Branch-and-Price Algorithm for EVCE

As we have shown in Table 3.1 in Section 3.3, the EVCE model is a large-scale MILP. Apart from the integrity of the variables, the large coefficient matrix also makes it difficult to solve this optimization model directly. Although the coefficient matrix is huge, it is sparse and has special structure, which make it possible for us to use the Dantzig-Wolfe decomposition method to reduce the difficulty of solving this model.

Figure 4.1 shows the visualized sparsity pattern of the coefficient matrix for an instance. The order of variables and constraints are reorganized to make the sparsity pattern more clear.

The blocks of the coefficient matrix show a special structure of this linear programming problem and this kind of structure is typical for Dantzig-Wolfe decomposition. Decision variables can be divided into several disjoint sets. The blocks are coefficient matrixes of pricing problems in which only variables in the same set are involved. The submatrix in the bottom of the figure are very different from the blocks. It is the coefficient matrix of coupling constraints, in which variables from different sets are connected together.

The constraints (2.4.2b), (2.4.2c), (2.4.2d), (2.4.2e), (2.4.2f), (2.4.2g), (2.4.2h) are specific to scenario-tree node n. The constraints (2.4.2i), (2.4.2j) are coupling

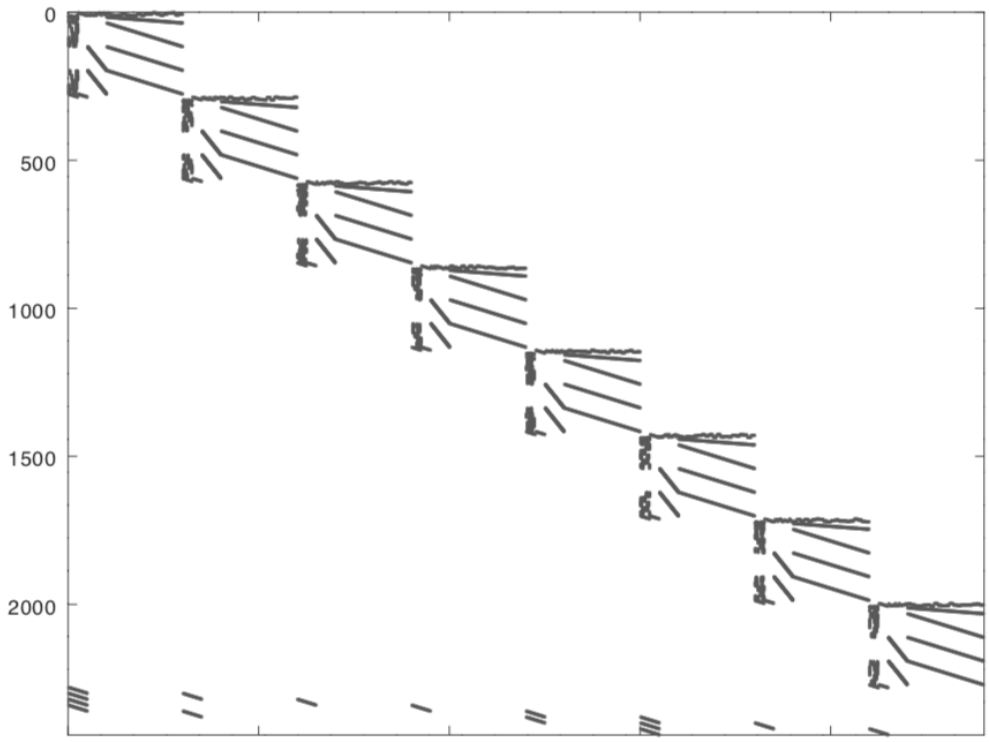

Figure 4.1: Sparsity pattern of the coefficient matrix

constraints in which variables from different scenario-tree nodes are involved. So we do the decomposition based on the scenario-tree nodes and only constraints (2.4.2i), (2.4.2j) are left in the master problem.

For each $n \in \mathcal{N}$, we define $\mathcal{X}_n = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) \mid \forall n \in \mathcal{N}\}$ where $\boldsymbol{x}_n = \{x_{n,j} \mid \forall j \in \mathcal{J}\}$ with $x_{n,j} \in \{0, 1\}$ and $\boldsymbol{y}_n = \{y_{n,j} \mid \forall j \in \mathcal{J}\}$ with $y_{n,j} \in \mathbb{Z}^+$. All $\boldsymbol{x}_n$'s and $\boldsymbol{y}_n$'s satisfy the constraints

$$\sum_{i \in \mathcal{I}_{n,j}} \theta_{n,i}(w_{n,i}\alpha_{n,i,j} + b_{n,i} \sum_{k \in \mathcal{J}_{n,i}} z_{n,i,j,k}) \leq C y_{n,j} \quad \forall j \in \mathcal{J} \tag{4.2.1a}$$

$$\sum_{k \in \mathcal{J}_{n,i}} x_{n,k} \geq 1 \quad \forall i \in \mathcal{I} \tag{4.2.1b}$$

$$y_{n,j} \leq M_j x_{n,j} \quad \forall j \in \mathcal{J} \tag{4.2.1c}$$

with $\alpha_{n,i,j} = \dfrac{e_{i,j}x_{n,j}}{\sum_{k\in\mathcal{J}_{n,i}} e_{i,k}x_{n,k}}$ and $z_{n,i,j,k} = \alpha_{n,i,j}x_{n,k}, \forall i \in \mathcal{I}, j \in \mathcal{J}$.

Since the variables are binary or integer, the set $\mathcal{X}_n$ has finite points. Thus we could write it as $\mathcal{X}_n = \{(\boldsymbol{x}_n^q, \boldsymbol{y}_n^q) \mid q = 1, ..., Q_n\}$. Any point $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ in $\mathcal{X}_n$ can be expressed as $\boldsymbol{x}_n = \sum_{q=1}^{Q_n} \lambda_n^q \boldsymbol{x}_n^q$ and $\boldsymbol{y}_n = \sum_{q=1}^{Q_n} \lambda_n^q \boldsymbol{y}_n^q$ for $\sum_{q=1}^{Q_n} \lambda_n^q = 1$ and $\lambda_n^q \in \{0,1\}, \forall q = 1, ..., Q_n$. Hence, the problem can be reformulated as

$$
\begin{aligned}
\min \quad & \sum_{n\in\mathcal{N}} \sum_{q=1}^{Q_n} \left(\sum_{j\in\mathcal{J}} (c_{n,j}^{(1)} x_{n,j}^q + c_{n,j}^{(2)} y_{n,j}^q) \right) \lambda_n^q - \Psi && && \text{(4.2.2a)} \\
\text{s.t.} \quad & \sum_{q=1}^{Q_{a(n)}} x_{a(n),j}^q \lambda_{a(n)}^q \le \sum_{q=1}^{Q_n} x_{n,j}^q \lambda_n^q && \forall n \in \mathcal{N} \quad \forall j \in \mathcal{J} && \text{(4.2.2b)} \\
\text{(MP)} \quad & \sum_{q=1}^{Q_{a(n)}} y_{a(n),j}^q \lambda_{a(n)}^q \le \sum_{q=1}^{Q_n} y_{n,j}^q \lambda_n^q && \forall n \in \mathcal{N} \quad \forall j \in \mathcal{J} && \text{(4.2.2c)} \\
& \sum_{q=1}^{Q_n} \lambda_n^q = 1 && \forall n \in \mathcal{N} && \text{(4.2.2d)} \\
& \lambda_n^q \in \{0,1\} && \forall n \in \mathcal{N} \quad \forall q = 1, ..., Q_n && \text{(4.2.2e)}
\end{aligned}
$$

where $c_{n,j}^{(1)} = \phi_n(c_{n,j}^s + \bar{c}_{n,j}^s) - \sum_{n'\in s(n)} \phi_{n'} c_{n',j}^s$, $c_{n,j}^{(2)} = \phi_n(c_{n,j}^p + \bar{c}_{n,j}^p) - \sum_{n'\in s(n)} \phi_{n'} c_{n',j}^p$ and $\Psi = \phi_1 \sum_{j\in\mathcal{J}} (c_{1,j}^s x_{0,j} + c_{1,j}^p y_{0,j})$.

Replace $\lambda_n^q \in \{0,1\}$ by $\lambda_n^q \ge 0$, we can get MP-LP, i.e., the linear relaxation of the master problem. In the branch-and-bound algorithm, we need to solve the MP-LP to get the optimal solution of the linear relaxation problem and hence lower bound of the branch.

However, the cardinality of $\mathcal{X}_n$ will be huge for any $n \in \mathcal{N}$. It is too time consuming to get all the points (also called columns) for $\mathcal{X}_n$. In fact, it is almost impossible to do so according to its complexity. So we use the column generation method to solve the linear relaxation of the master problem.

Similar to the column generation method, we use a subset of $\mathcal{X}_n$ instead of the entire set. By doing so, we get a restricted version of the master problem. The restricted master problem RMP is

$$\min \quad \sum_{n\in\mathcal{N}}\sum_{q\in\mathcal{Q}_n}\left(\sum_{j\in\mathcal{J}}(c^{(1)}_{n,j}x^q_{n,j}+c^{(2)}_{n,j}y^q_{n,j})\right)\lambda^q_n-\Psi \tag{4.2.3a}$$

$$\text{s.t.} \quad \sum_{q\in\mathcal{Q}_n}x^q_{n,j}\lambda^q_n-\sum_{q\in\mathcal{Q}_{a(n)}}x^q_{a(n),j}\lambda^q_{a(n)}\geq 0 \qquad \forall n\in\mathcal{N} \quad \forall j\in\mathcal{J} \tag{4.2.3b}$$

$$\text{(RMP)} \quad \sum_{q\in\mathcal{Q}_n}y^q_{n,j}\lambda^q_n-\sum_{q\in\mathcal{Q}_{a(n)}}y^q_{a(n),j}\lambda^q_{a(n)}\geq 0 \qquad \forall n\in\mathcal{N} \quad \forall j\in\mathcal{J} \tag{4.2.3c}$$

$$\sum_{q\in\mathcal{Q}_n}\lambda^q_n=1 \qquad \forall n\in\mathcal{N} \tag{4.2.3d}$$

$$\lambda^q_n\geq 0 \qquad \forall n\in\mathcal{N} \quad \forall q\in\mathcal{Q}_n \tag{4.2.3e}$$

where $\mathcal{Q}_n\subseteq\{1,...,Q_n\}$ for all $n\in\mathcal{N}$.

Let $\pi^{(1)}_{n,j}$ and $\pi^{(2)}_{n,j}$ be the dual variables associated with constraints (4.2.3b) and (4.2.3c) for each $j\in\mathcal{J}$ and $n\in\mathcal{N}$, respectively. Dual variable associated with constraint (4.2.3d) is μ_n for each $n\in\mathcal{N}$.

For any scenario-tree node $n \in \mathcal{N}$, the corresponding pricing problem SP (n) is

$$\min \quad \sum_{j \in \mathcal{J}} \left[\left(c^{(1)}_{n,j} - \pi^{(1)}_{n,j} + \sum_{n' \in s(n)} \pi^{(1)}_{n',j} \right) x_{n,j} + \left(c^{(2)}_{n,j} - \pi^{(2)}_{n,j} + \sum_{n' \in s(n)} \pi^{(2)}_{n',j} \right) y_{n,j} \right] - \mu_n \tag{4.2.4a}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{I}_{n,j}} \theta_{n,i} (w_{n,i} \alpha_{n,i,j} + b_{n,i} \sum_{k \in \mathcal{J}_{n,i}} z_{n,i,j,k}) \leq C y_{n,j} \qquad \forall j \in \mathcal{J} \tag{4.2.4b}$$

$$\sum_{k \in \mathcal{J}_{n,i}} x_{n,k} \geq 1 \qquad \forall i \in \mathcal{I} \tag{4.2.4c}$$

$$\sum_{k \in \mathcal{J}_{n,i}} e_{i,k} z_{n,i,j,k} = e_{i,j} x_{n,j} \qquad \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i} \tag{4.2.4d}$$

$$(\text{SP}(n)) \quad z_{n,i,j,k} \leq x_{n,k} \qquad \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i}\ \forall k \in \mathcal{J}_{n,i} \tag{4.2.4e}$$

$$z_{n,i,j,k} \leq \alpha_{n,i,j} \qquad \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i}\ \forall k \in \mathcal{J}_{n,i} \tag{4.2.4f}$$

$$z_{n,i,j,k} \geq \alpha_{n,i,j} + x_{n,k} - 1 \qquad \forall i \in \mathcal{I}\ \forall j \in \mathcal{J}_{n,i}\ \forall k \in \mathcal{J}_{n,i} \tag{4.2.4g}$$

$$y_{n,j} \leq M_j x_{n,j} \qquad \forall j \in \mathcal{J} \tag{4.2.4h}$$

$$x_{n,j} \in \{0, 1\} \qquad \forall j \in \mathcal{J} \tag{4.2.4i}$$

$$y_{n,j} \in \mathbb{Z}^+ \qquad \forall j \in \mathcal{J} \tag{4.2.4j}$$

It should be noted that we keep the integrity of decision variables $x_{n,j}$ and $y_{n,j}$ in the SP(n) as seen in constraints $(4.2.4i)$ and $(4.2.4j)$. The integrity constraints in the pricing problem lead to integer columns in which $x_{n,j}$ is binary and $y_{n,j}$ is integer.

This will slow down the procedure of solving the pricing problem, which in turn slows down the procedure of solving the subproblem in the branch-and-bound tree. However, it dramatically increases the lower bound of the subproblem. The effect is so obvious that the algorithm can stop before the branch-and-bound tree grows big. More precisely, we can get a lower bound which is very close to the optimal objective value in most cases after solving the subproblem for the root node of the branch-and-bound tree. In general, it reduces the total solution time.

Now the branch-and-price procedure is clear. First, we solve the RMP to get its dual solution $\{(\pi^{(1)}_{n,j}, \pi^{(2)}_{n,j}, \mu_n) \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$. Then for each $n \in \mathcal{N}$, using the corresponding dual solution $\{(\pi^{(1)}_{n,j}, \pi^{(2)}_{n,j}, \mu_n) \mid \forall j \in \mathcal{J}\}$, we construct the SP($n$) whose objective is to minimize the reduced cost of any columns associated with variables for this specific n. After that, we solve all the SP(n)s and add all generated columns with negative reduced costs to the RMP. Repeat this procedure until all optimal objective values of the SP(n)s are non-negative. Now, we have included all basic columns in the RMP. It means that the optimal objective value of the RMP is the lower bound of this subproblem in the branch-and-bound tree. Since the most difficult part of the branch-and-bound algorithm is to get the lower bound for a subproblem in the branch-and-bound tree, we can then solve the EVCE model in the branch-and-bound framework. Algorithm 5 in Appendix A describes the branch-and-price algorithm we use.

4.3 Implementation of the Basic Branch-and-Price Algorithm

In this section, some implementation details of the branch-and-price algorithm for the EVCE model will be explained.

The CPLEX MIP solver is used in the branch-and-price algorithm implementation to solve the RMP and the pricing problems. The whole branch-and-price framework is built by combining C++ and CPLEX C++ API (application programming interfaces). In the numerical experiment part of this chapter, we will also solve the EVCE model directly by the CPLEX MIP solver and compare the computational results and the solution time with those of the branch-and-price algorithm.

4.3.1 Branching Rules

The decision variables of the MP are λ_n^qs for all $n \in \mathcal{N}$ and $q = 1, \cdots, Q_n$. A natural idea about the branching rule is to fix λ_n^q to 0 or 1 in two branches respectively for a specific n and q. However, this branching rule is not efficient since there are too many variables and only very few of them will take the value 1 in the optimal solution. A better branching rule for a program based on the Dantzig-Wolfe decomposition is to fix the original decision variables $x_{n,j}$ and $y_{n,j}$ for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$. This branching rule will be applied in our branch-and-price algorithm.

The branching procedure can be performed to fix $x_{n,j}$'s first and then to fix $y_{n,j}$'s when all the $x_{n,j}$'s are assigned to certain values. Moreover, we have a much simpler implementation. According to Theorem 3.1.2, when $x_{n,j}$'s are fixed, we can easily get the optimal integer solution for the original EVCE model by solving the linear

relaxation of EVCE and rounding up the resulted values for $y_{n,j}$'s. In other words, it is sufficient to branch only by fixing the values of $x_{n,j}$'s. This will greatly reduce the size of the branch-and-bound tree and also save the work when programming.

To fix the value of a specific $x_{n,j}$ during branching procedure under the Dantzig-Wolfe decomposition framework, we can simply add a constraint $\sum_{q\in\mathcal{Q}_n} x^q_{n,j}\lambda^q_n = 0$ (or 1) to the RMP. But we actually used another method when programming. We add the constraint $x_{n,j} = 0$ (or 1) to the pricing problem SP(n). In this way, any new column $(\boldsymbol{x}^q_n, \boldsymbol{x}^q_n)$ which is generated by solving the SP(n) will satisfy the constraint $x^q_{n,j} = 0$ (or 1). We also make sure that any initial columns we put in the column pool for this subproblem of the branch-and-bound tree will satisfy the constraint $x^q_{n,j} = 0$ (or 1). Then, $\sum_{q\in\mathcal{Q}_n} x^q_{n,j}\lambda^q_n = 0$ (or 1) is automatically satisfied because $x^q_{n,j} = 0$ (or 1) for all $q \in \mathcal{Q}_n$. Therefore, we can keep the RMP unchanged during the branching procedure. Thus the structure of the dual solution of the RMP will stay the same, and so does the structure of the objective functions of the pricing problems. This will make the code of the algorithm more concise and clear.

Due to the principle that stations and chargers can not disappear over time, an implementation detail should be noted in the branching procedure. Once $x_{n,j}$ is fixed to 0, it is reasonable to fix all $x_{n',j}, \forall n' \in \mathcal{P}_n$ to 0. Similarly, when $x_{n,j}$ is fixed to 1, all the corresponding decision variables $x_{n',j}$ at its successors nodes ($n' \in \mathcal{T}_n$) should be fixed to 1. Although the algorithm still works without doing so, branching in this way will make it convenient when we construct an initial solution for a subproblem, because it saves the work of checking if the stations and chargers disappear in the solution we construct.

4.3.2 Initialization

The initialization of the branch-and-price algorithm is trivial. It is all right to set the lower bound of each subproblem to 0 and to set the global upper bound to infinity. Since the column generation method is used to solve the linear relaxation of the subproblems in the branch-and-bound tree, the column pools are necessary to be initialized. An easy way to initialize the column pools $\mathcal{Q}_n$ is to construct a solution $(\boldsymbol{x}, \boldsymbol{y}) = \{(\boldsymbol{x}_n, \boldsymbol{y}_n) \mid n \in \mathcal{N}\}$ by assigning 1 to all unfixed $x_{n,j}$ and let $y_{n,j} = M_j \cdot x_{n,j}$ for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$. This column $(\boldsymbol{x}, \boldsymbol{y})$ automatically satisfy the coupling constraints due to the branching detail we mentioned in the previous subsection. If for any $n \in \mathcal{N}$, $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ is feasible to the pricing problem SP(n), the column pool $\mathcal{Q}_n$ is initialized by putting $(\boldsymbol{x}_n, \boldsymbol{y}_n)$ in it. Otherwise, this subproblem is infeasible and it should be "pruned" from the branch-and-bound tree.

4.3.3 Lower Bound of Subproblems

According to the theories of MILP we presented in Section 4.1, the solution of the RMP is optimal to the MP only when all the optimal objective values of the pricing problems are non-negative. Therefore, a lower bound of the subproblem we are solving can only obtained when the column generation procedure is totally completed. In practice, this is not good since for most of the instances of the EVCE model, the branch-and-price algorithm needs to solve very few subproblems, sometimes only the subproblem at the root node in the branch-and-bound tree, to get the optimal solution. It implies that most of the time there is no lower bound for us to estimate the progress of the work. Therefore, we need a new method to obtain a lower bound during the column generation process when some of the optimal objective values of

the pricing problems are still negative. The new lower bound construction method is also useful in the case where some optimal objective values of the pricing problems remain negative because of computing precision. Since the number of bits used to store a number will often cause some loss of accuracy, the optimal objective value of a pricing problem may appear as a small negative number even when its exact arithmetic value is 0 because of the rounding errors during the computation. The new lower bound construction method can help us in this case to estimate if the lower bound is good enough to stop the column generation process.

Theorem 4.3.1 *Let z^{MP-LP} denote the optimal objective value of the linear relaxation of the MP. In an iteration of the column generation, let z^{RMP} and ξ_n $(\forall n \in \mathcal{N})$ denote the optimal objective value of the RMP and the SP(n), respectively. Then*

$$z^{MP-LP} \geq z^{RMP} + \sum_{\xi_n<0} \xi_n.$$

Proof. Suppose that in an iteration of the column generation, the dual solution of the RMP is $(\boldsymbol{\pi}, \boldsymbol{\mu})$. Let us use the dual solution as multipliers to reformulate the objective function of the MP-LP. Let $r_{n,q}$ denote the reduced cost of $\lambda_{n,q}$ under the multipliers $(\boldsymbol{\pi}, \boldsymbol{\mu})$ for all $n \in \mathcal{N}$ and $q = 1, \cdots, Q_n$. The objective function of the MP-LP is reformulated by the multipliers as

$$\sum_{n\in\mathcal{N}} \sum_{q=1}^{Q_n} r_{n,q}\lambda_{n,q} + z^{RMP} = \sum_{\xi_n\geq 0} \sum_{q=1}^{Q_n} r_{n,q}\lambda_{n,q} + \sum_{\xi_n<0} \Big(\sum_{q\in\mathcal{Q}_n} r_{n,q}\lambda_{n,q} + \sum_{q\notin\mathcal{Q}_n} r_{n,q}\lambda_{n,q} \Big) + z^{RMP}$$

For any $n \in \mathcal{N}$, it is clear that $r_{n,q} \geq 0$ for $q \in \mathcal{Q}_n$ and $r_{n,q} \geq \xi_n$ for $q \notin \mathcal{Q}_n$. It is also true that for $q \notin \mathcal{Q}_n$,

$$r_{n,q} \geq 0, \text{if } \xi_n \geq 0, \quad \text{and} \quad r_{n,q} \geq \xi_n^-, \text{if } \xi_n < 0.$$

Let z^{MP-LP} denote the optimal objective value of the MP-LP. We can get a lower bound of z^{MP-LP} by setting $\lambda_{n,q} = 0$ if $r_{n,q} \geq 0$. Then

$$\begin{aligned} z^{MP-LP} &\geq \min\Big\{ \sum_{\xi_n<0} \sum_{q\notin\mathcal{Q}_n} r_{n,q}\lambda_{n,q} + z^{RMP} \mid \sum_{q\notin\mathcal{Q}_n} \lambda_{n,q} = 1, \lambda_{n,q} \geq 0, \forall \xi_n < 0 \Big\} \\ &\geq \min\Big\{ \sum_{\xi_n<0} \xi_n \cdot \sum_{q\notin\mathcal{Q}_n} \lambda_{n,q} \mid \sum_{q\notin\mathcal{Q}_n} \lambda_{n,q} = 1, \lambda_{n,q} \geq 0, \forall \xi_n < 0 \Big\} + z^{RMP} \\ &= z^{RMP} + \sum_{\xi_n<0} \xi_n \end{aligned}$$

□

Therefore, we can use $z^{RMP} + \sum_{\xi_n<0} \xi_n$ as a lower bound during the column generation iterations even when some of the reduced costs are negative. We can also stop the column generation procedure when this lower bound is good enough.

4.3.4 Feasible Solutions and Upper Bounds

Normally, the branch-and-price algorithm tests the integrity of the variables when a subproblem in the branch-and-bound tree is solved in its linear relaxation form and the branch-and-price algorithm updates the currently best solution and the upper bound if the current relaxed solution happens to be an integer and a better solution. But it is possible to capture feasible solutions in each iteration of the column generation progress. A corollary of Theorem 3.1.1 is presented.

Corollary 4.3.1 *In an iteration of the column generation, let* $\{\lambda_n^q \mid \forall n \in \mathcal{N}, \forall q \in \mathcal{Q}_n\}$ *denote the optimal solution of the RMP and the column pool is* $\{(\boldsymbol{x}_n^q, \boldsymbol{y}_n^q) \mid \forall n \in \mathcal{N}, \forall q \in \mathcal{Q}_n\}$. *Let* $x_{n,j} = \sum_{q \in \mathcal{Q}_n} \lambda_n^q \cdot x_{n,j}^q$ *and* $y_{n,j} = \sum_{q \in \mathcal{Q}_n} \lambda_n^q \cdot y_{n,j}^q$, $\forall n \in \mathcal{N}, \forall j \in \mathcal{J}$. *If all* $x_{n,j}$*'s are binary, then* $\{(x_{n,j}, \lceil y_{n,j} \rceil) \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is a feasible solution to the EVCE.*

During the iterations of the column generation, the currently best solution and the upper bound of the problem can both be updated if a better solution is captured as shown in Corollary 4.3.1.

4.4 Improvement of the Branch-and-Price Algorithm

Although the basic branch-and-price algorithm performs better than the CPLEX MIP solver, we have tried some ways to improve the algorithm by further reducing the solution time in order to solve larger scale instances.

4.4.1 Initialization

It is not necessary to set initial feasible solutions for the branch-and-price algorithm, since it will capture feasible solutions during the column generation process. But this process is very inefficient because it can only get a feasible solution in rare cases when all the integer variables accidentally obtain values without fractional components. If no good solution is captured, it will lead to an extremely high upper bound. In this case, the algorithm has to continue even if the lower bound is good enough. This is a

waste of time and this waste can be avoided if an initial feasible solution is assigned to each subproblem in the branch-and-bound tree.

The heuristic algorithm in Chapter 3 suggests a better way to initialize the feasible solution, the upper bound and the initial columns in the column pool. It is convenient to take the heuristic solution directly as the initial feasible solution. Thus, the initial upper bound can be the objective value of the heuristic solution. Moreover, several columns are generated during the procedure of the heuristic algorithm (Algorithm 3), and some of these columns can be in the initial column pool. In practice, the heuristic algorithm will consume very little time, and it offers a good initial solution and several useful initial columns which greatly improve the efficiency of the branch-and-price algorithm for some instances.

In Figure 4.2, the bold lines show the lower and upper bounds of the branch-and-price with the initial heuristic solution and related initial columns while the slim lines show the lower and upper bounds of the basic branch-and-price algorithm over time. It is clear that the initial heuristic solution is very useful. The basic branch-and-price does not stop when its lower bound is good enough because its upper bound is too far away from the optimal objective value. Moreover, the initial columns also help the algorithm by speeding up the convergence.

4.4.2 Column Generation

The most time consuming part of the branch-and-price algorithm is the column generation process. One of the reasons is that it takes time to get columns by solving the pricing problems since they are mixed-integer programs.

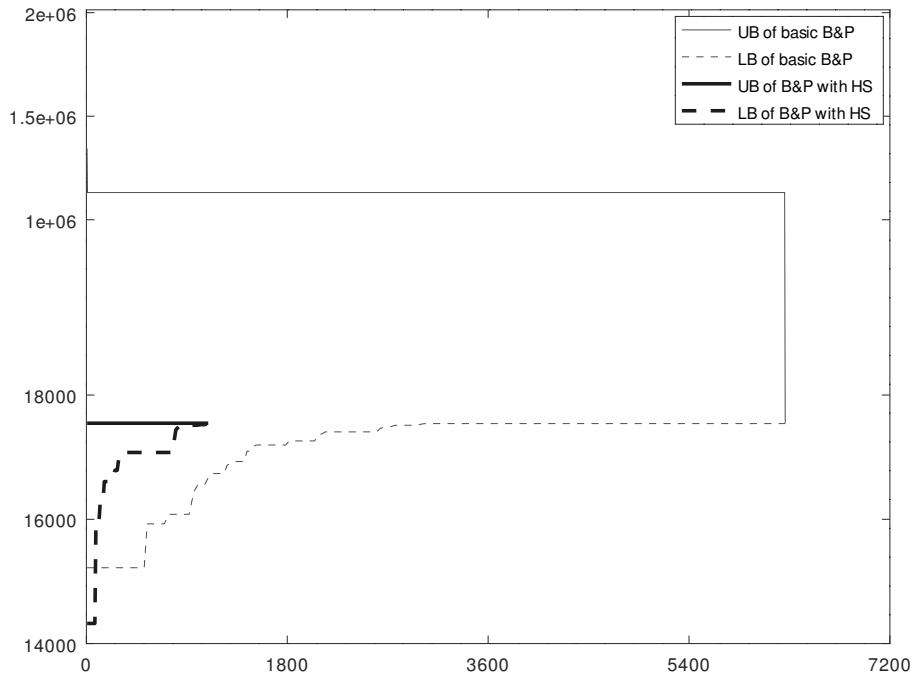

Figure 4.2: Improvement of the branch-and-price algorithm with the heuristic solution

Heuristic Columns

A heuristic method which is similar to Algorithm 4 is used to help create columns with negative reduced costs during column generation procedure.

For the pricing problem SP(n) of a subproblem in the branch-and-bound tree, $\forall n \in \mathcal{N}$, let $coeX_{n,j}$ and $coeY_{n,j}$ be the coefficient of $x_{n,j}$ and $y_{n,j}$ in the objective function of SP(n), respectively. Thus, $coeX_{n,j} = c^{(1)}_{n,j} - \pi^{(1)}_{n,j} + \sum_{n' \in s(n)} \pi^{(1)}_{n',j}$ and $coeY_{n,j} = c^{(2)}_{n,j} - \pi^{(2)}_{n,j} + \sum_{n' \in s(n)} \pi^{(2)}_{n',j}$. The objective function of SP(n) is expressed as $\sum_{j \in \mathcal{J}} coeX_{n,j} x_{n,j} + \sum_{j \in \mathcal{J}} coeY_{n,j} y_{n,j} - \mu_n$.

Then, Algorithm 4 can be used to help create columns for SP(n) with very few

changes. The input pricing problem is changed from SP(1) to SP(n). At step 14, choose candidate location j_0 with the lowest $coeX_{n,j} + \bar{y} \cdot coeY_{n,j}$, where $\bar{y}$ is a constant which is better to be close to the average number of chargers a charging station will have. After getting each $\boldsymbol{x}_n^q$, let $\boldsymbol{y}_n^q = g(\boldsymbol{x}_n^q)$. Lastly, calculate the objective value of $(\boldsymbol{x}_n^q, \boldsymbol{y}_n^q)$ as $\sum_{j \in \mathcal{J}} coeX_{n,j} x_{n,j}^q + \sum_{j \in \mathcal{J}} coeY_{n,j} y_{n,j}^q - \mu_n$. If the objective value is negative, a qualified column $(\boldsymbol{x}_n^q, \boldsymbol{y}_n^q)$ is found. See Algorithm 6 in Appendix A for more details.

Using this heuristic method instead of solving the pricing problem to generate columns can save a lot of time in practice. But it is best to limit the number of the heuristic columns in the column pools in case they take up too much memory and slow down the algorithm.

Approximation Columns

In Section 3.1, an approximation algorithm (Algorithm 1) is presented for EVCE. The idea is very simple. To decrease the difficulty of solving this mixed-integer program, it removes the integer constraints for the variable $y_{n,j}$ and then round up the result for $y_{n,j}$ for all $n \in \mathcal{N}$ and $j \in \mathcal{J}$. This idea will be used in column generation when solving the pricing problems.

Corollary 4.4.1 *For any pricing problem SP(n), let SP-LP(n) denote its relaxation form by removing the integer constraints of variable $y_{n,j}$ for all $j \in \mathcal{J}$. If $(\boldsymbol{x}, \boldsymbol{y})$ is a feasible solution of SP-LP(n), then $(\boldsymbol{x}, \lceil \boldsymbol{y} \rceil)$ is feasible to SP(n).*

The proof is omitted here since it is almost the same with the proof of Theorem 3.1.1. Therefore, $(\boldsymbol{x}, \lceil \boldsymbol{y} \rceil)$ can be used as a column generated by our approximation method. Calculate its objective value $\sum_{j \in \mathcal{J}} coeX_{n,j} x_{n,j} + \sum_{j \in \mathcal{J}} coeY_{n,j} \lceil y_{n,j} \rceil - \mu_n$.

If the objective value is negative, then this column is put into the column pool successfully. Otherwise, we can only get a column or prove optimality by solving SP(n) directly.

Solving Pricing Problems Periodically

The heuristic method can generate columns very quickly, so it is our first choice to generate columns with negative reduced costs. However, if the columns generated at the beginning of the heuristic method have non-negative costs, the heuristic method should be terminated in case it spends too much time searching a qualified column in vain. Therefore, we set a threshold which is 0 or a negative number for the reduced costs of the heuristic columns. If the first column the heuristic method generates has a reduced cost less than the threshold, the heuristic method will be the only method we use to generate columns in this iteration. Otherwise, we skip the heuristic method and use the approximation method instead which has a better chance to get a column with the minimal reduced cost. If the approximation column is not qualified either, the pricing problem has to be solved to optimality.

In the beginning of the column generation, the two methods may continue to work without activating the optimization procedure, but we need to periodically evaluate the lower bound. At the end of the column generation, the two methods have few chances to get a qualified column, so it may be time wasting to try them in every iteration before activating the optimization procedure. Therefore, we design a procedure to control the frequency of using the heuristic method and the approximation method in iterations.

Let n_{iter} denote the number of an iteration for the column generation process and

n_{period} denote a constant integer which is set in advance as the number of iterations in a period. The pricing gap δ_p is

$$\delta_p = \min\left\{\frac{z^{RMP} - LB}{z^{RMP}}, 1\right\}$$

where z^{RMP} is the objective value of the RMP and LB is the lower bound of this subproblem currently.

Note that $n_{mod} = (n_{iter} \bmod n_{period})$ and it is the remainder when we divide n_{iter} by n_{period}. So n_{mod} will be an integer from 0 to $n_{period} - 1$ with n_{iter} increasing. The procedure is as follows,

1. if $n_{mod} < n_{period} \cdot \delta_p - 1$, try methods in the order of the heuristic method, the approximation method and then solving the pricing problems to optimality until a qualified column is obtained or the optimality is proved;

2. else, skip the heuristic and the approximation method and solve the pricing problems directly to get a column or prove the optimality.

This procedure performs well in practice. Here is an example for how it works. Set n_{period} to 10, so there are 10 iterations in each period.

1. When the pricing gap $\delta_p > 90\%$ at the beginning of the column generation, the heuristic and approximation methods are applied when the remainder of n_{iter} divided by 10 is from 0 to 8, and they are skipped when the remainder is 9.

2. When the pricing gap δ_p is around 50%, the two methods are applied when the remainder is from 0 to 3 and are skipped when the remainder is from 4 to 9.

3. When the pricing gap $\delta_p < 10\%$ at the end of the column generation, the two methods are skipped in every iteration.

The difference of the solution procedure is presented in Figure 4.3 for the branch-and-price algorithm with (the bold line) and without (the slim line) the heuristic columns and the approximation columns.

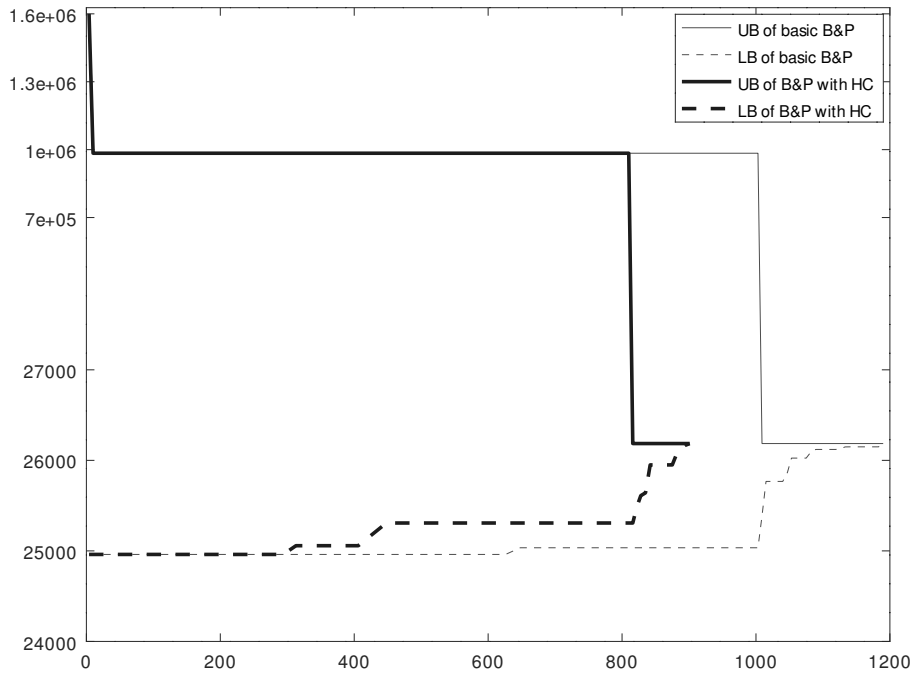

Figure 4.3: Improvement of the branch-and-price algorithm with the heuristic columns and the approximation columns

4.4.3 Column Management

Looking at the process of the branch-and-price algorithm, we find that the time of each iteration is getting longer and longer during the column generation procedure.

The main reason is that there are more and more redundant columns in the column pool and they are slowing down the column generation and convergence. Therefore, column management is necessary and the whole algorithm will benefit from deleting redundant columns periodically. But it is very important to decide which column is most likely to be redundant, because it would be time wasting if a column is deleted and generated repeatedly.

The main work of column generation is to find out all the columns corresponding to the basic variables in the MP and to make sure that any column which is not in the column pool has a non-negative reduced cost. So during the column generation process, a column is very likely to be unnecessary to stay in the column pool if it has not been in the basis of the RMP for many iterations. The idea of managing the columns is to delete these columns periodically. Deleting the redundant columns will help the algorithm to speed up, especially for the improved branch-and-price algorithm with the heuristic columns and the approximation columns, since these columns are very likely to be redundant.

We solve the same instance in the previous subsection and do column management when using the heuristic columns and the approximation columns. The solution process is presented in Figure 4.4. The bold line is for the solution process with column management which is abbreviated as CM in the figure. The other two lines are the same as the lines in Figure 4.3.

4.5 Numerical Experiments

We solve the same instances generated in Section 3.3. Table 4.1 shows the comparison between the CPLEX MIP solver and the improved branch-and-price algorithm. For

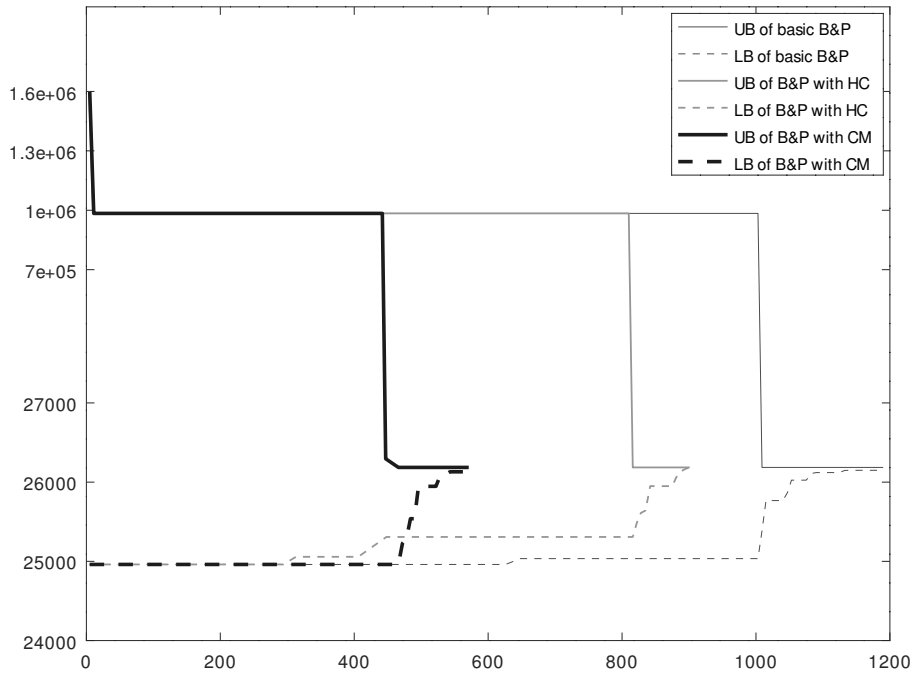

Figure 4.4: Improvement of the branch-and-price algorithm with column management

some instances, both of them will get the optimal solution within 7200 seconds, thus their objective values are the same and the gaps are all 0%. For these instances, we omit the objective values and gaps and put an asterisk in the table to make it easy to read and compare.

For most instances, the branch-and-price algorithm will obtain the optimal solution in a relatively short time. Another property of the branch-and-price algorithm which is not shown in the table is that it will give a good feasible solution and a good lower bound at the very beginning. This is important for the instances that do not need to solve to optimality. Figure 4.5 and Figure 4.6 show the upper and lower bounds of the branch-and-price algorithm over time compared with those of the

CPLEX MIP solver for Instances 11 and 12, respectively. These two instances are not solved to optimality in two hours for both the branch-and-price algorithm and the CPLEX MIP solver, but we can see that the branch-and-price algorithm obtains a feasible solution and a lower bound at the very beginning which are even better than the final solution and lower bound of the CPLEX MIP solver after two hours of work.

Table 4.1: Efficiency of the branch-and-price algorithm

Inst.	Inst. Size		CPLEX MIP solver		B&P	
#	$\|\mathcal{I}\|$	$\|\mathcal{J}\|$	$t(s)$	$z(GAP)$	$t_1(s)$	$z_1(GAP_1)$
1	10	10	15	*	7	*
2	10	10	42	*	81	*
3	10	15	2,579	*	15	*
4	10	15	468	*	31	*
5	10	20	$> 7,200$	16,611(5.6%)	155	16,611(0.0%)
6	10	20	$> 7,200$	17,546(6.6%)	2,003	17,546(0.0%)
7	15	15	192	*	16	*
8	15	15	530	*	27	*
9	15	20	3,924	*	285	*
10	15	20	$> 7,200$	27,048(6.7%)	2,659	26,894(0.0%)
11	15	30	$> 7,200$	27,332(35.2%)	$> 7,200$	25,683(13.3%)
12	15	30	$> 7,200$	26,125(26.2%)	$> 7,200$	26,025(4.8%)

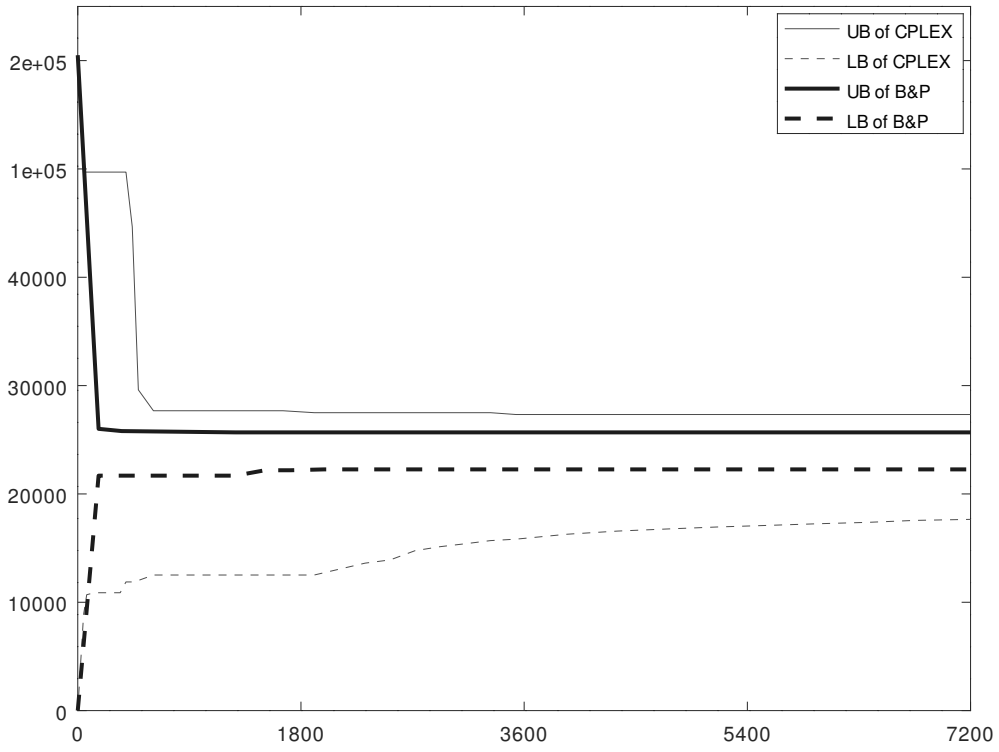

Figure 4.5: Solution procedure of Instance 11 by the branch-and-price algorithm and the CPLEX MIP solver

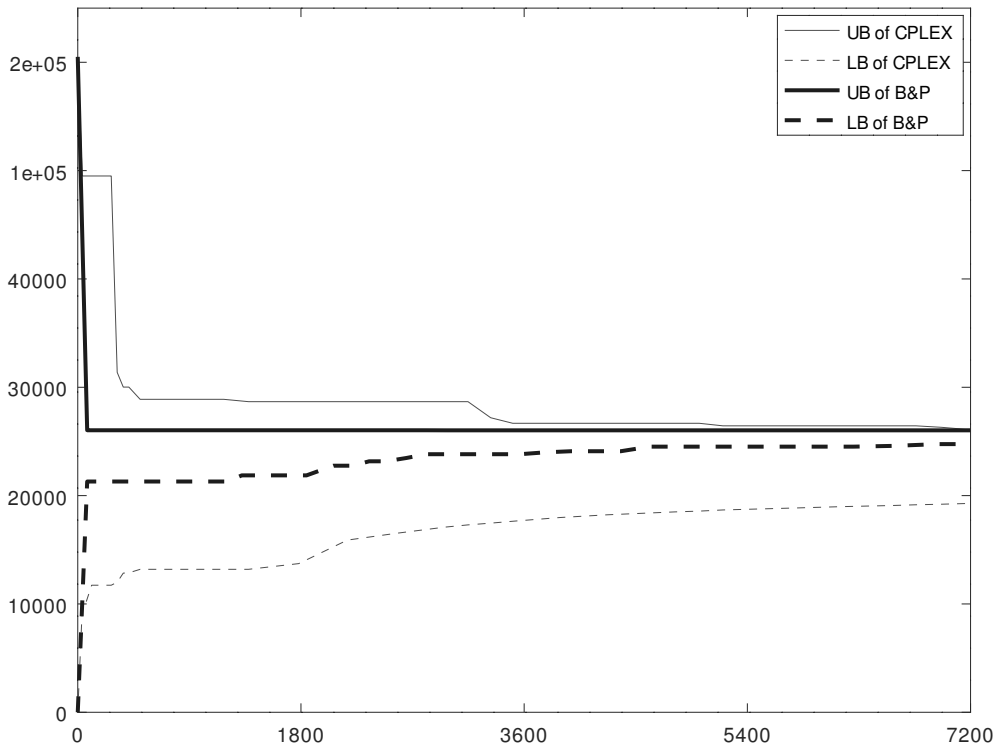

Figure 4.6: Solution procedure of Instance 12 by the branch-and-price algorithm and the CPLEX MIP solver

Chapter 5

Case Study: Electric Vehicle Charging Network Design for Oakville, Canada

We will demonstrate the model by applying it to the town of Oakville, which is located in southern Ontario between Toronto and Hamilton, Canada. The study period is explained by a scenario tree with three stages which represent the year of 2019, 2021 and 2023 respectively (each period lasts for two years). The scenario tree is shown in Figure 5.1.

The area of study covers about 139 km^2. It is divided into 57 zones in the Transportation Tomorrow Survey (TTS) (Data Management Group, 2016a). The estimates of charging demands for the TTS zones are based on the EV charging events of Greater Toronto and Hamilton Area from a survey conducted by the McMaster Institute for Transportation and Logistics (MITL) (MITL, 2018). The MITL's survey collects informations related to EV public charging, which means home charging and work

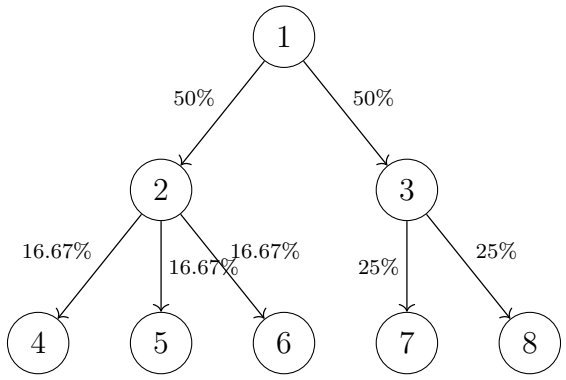

Figure 5.1: The scenario tree used in case study of Oakville

charging events are excluded. The EV owners in the survey report the number of public charging events they conduct during the resent week. These reported public charging events are linked to the EV owners' home where the survey is conducted. We use the origin-destination matrix for the Greater Toronto and Hamilton Area from the TTS in 2016 to allocate the charging events to all the TTS zones (Data Management Group, 2016b). Only discretionary trips 25 km and over during a typical weekday were used in the origin-destination matrix to get the public charging events linked to destinations, since EV owners are less likely to charge on a journey close to home. The estimate of the basic charging demand for a TTS zone in Oakville is obtained by multiplying a random number from a lognormal distribution to the number of charging events lined to this zone as destination.

We use 70 sites for the charging stations including 57 candidate locations and 13 existed EV charging station sites. The candidate locations are chosen from points near main intersections and parking lots. The informations about the existed EV charging stations, including the locations and the number of charging ports, are collected from the website of ChargeHub which is recommended by the Ministry of Transportation Ontario Government in 2019 (Ministry of Transportation Ontario Government, 2019).

We assume that all the newly installed charging stations use AC level 2 chargers. An estimate of the costs of installing the charging stations and the chargers is based on the EV supply equipment unit cost and the installation cost which we get from the report about costs associated with non-residential electric vehicle supply equipment (Smith and Castellano, 2015), and an additional fee for rental of the location itself which we get from the real estate website which is owned and operated by the Canadian Real Estate Association (CREA, 2019). An estimate of the costs of operating the charging stations and the chargers during the following two-year period is based on the management costs and the electricity consumption charges (Cities, 2012).

The charging demands in 2021 and 2023 are estimated from the data in 2019 by multiplying a random number from a lognormal distribution with standard deviation 0.5 and mean 1 and 1.5 respectively. The cost data is discounted at a random rate between 0% to 20% for each stage of the scenario tree.

We change only the coverage radius $R_{n,i}$ for every stage in the scenario tree to get several instances. After solving all the instances, we summary the resulted different EV charging networks for Oakville in Table 5.1. In the table, we report the expected numbers of stations and chargers in each stage of the scenario tree and the total costs under different coverage pattern. The gap is reported together with the total cost when neither the CPLEX MIP solver nor the branch-and-price algorithm can solve the instance to optimality in 48 hours. From the table, we can see that a smaller coverage radius will results in a significant increase on the number of charging stations, but the number of chargers is relatively less affected.

In Figure 5.2, we show the result when the coverage radius is set to 1.2 km for node 7 in the scenario tree of Figure 5.1. The round symbols are the locations of the

Table 5.1: Results of EVCE model for Oakville with different coverage radiuses

Radius (km)			Expected # of Stations			Expected # of Chargers			Total Cost ($)
2019	2021	2023	2019	2021	2023	2019	2021	2023	
1.2	1.2	1.2	40.0	40.0	40.6	84.0	123.0	159.7	1,262,104
1.5	1.5	1.5	31.0	31.5	32.7	80.0	115.5	152.0	1,177,066 (9%)
2.0	2.0	2.0	25.0	26.0	26.4	80.0	116.5	149.1	1,159,769 (20%)
2.0	1.5	1.2	23.0	36.0	41.3	76.0	122.0	164.3	1,258,796 (3%)

existed charging stations and the square symbols are the candidate locations which are selected by the model for the scenario node 7. The number inside the round or square symbol shows how many chargers the model suggests to install at each charging station.

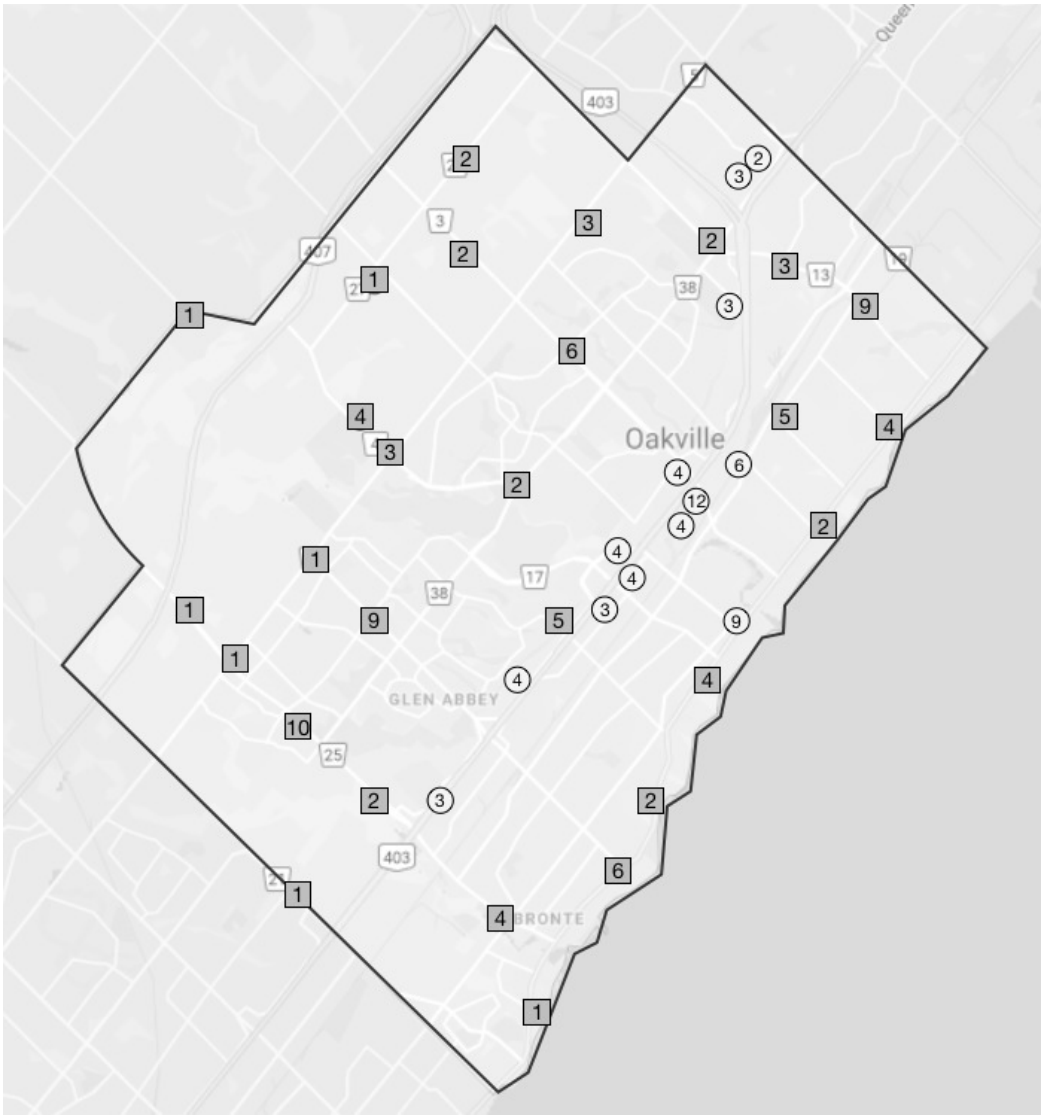

Figure 5.2: An example of the design of EV charging network for Oakville in 2023

Chapter 6

Conclusion

We proposed a facility location and capacity expansion model for the design of the EV charging network. The model can help determine not only the locations of the charging stations but also how many chargers are needed in the charging stations. A scenario tree is used in the model to deal with the uncertainty of the data and to make plans of designing the EV charging network over a discrete and finite time horizon. We use the logit choice model to estimate the customer choice behaviours and assume that distance is the only factor drivers will consider when choosing a charging station.

To solve this multi-stage stochastic mixed-integer program, we designed an approximation algorithm and a heuristic algorithm. We also investigated the branch-and-price algorithm based on the Dantzig-Wolfe decomposition and applied it to the model. The numerical experiments show that all these three algorithms have great advantages over the CPLEX MIP solver.

In future research, other heuristic algorithms could be designed or other methods such as the parallel computing could be included in the existed algorithms to make

the larger scale instances solvable. At the model level, we may improve the model by considering more factors when involving customer choice behaviours. For example, a driver's choice may also be affected by the probability that the charging station does not have available chargers.

Appendix A

Algorithms

Algorithm 4 A recursion method to get solutions for SP(1)

Input: p, Q

Output: $\{\boldsymbol{x}_1^q \mid 1 \leq q \leq Q\}$

1: $\mathcal{J}^{fix0} := \{j \in \mathcal{J} \mid x_{1,j}$ is fixed to 0 in p at node 1 $\}$

2: $\mathcal{J}^{fix1} := \{j \in \mathcal{J} \mid x_{1,j}$ is fixed to 1 in p at node 1 $\} \bigcup \{j \in \mathcal{J} \mid x_{0,j} = 1\}$

3: $\mathcal{I}^{covered} := \{i \in \mathcal{I} \mid \exists j$ s.t. $j \in \mathcal{J}^{fix1}$ and $d_{i,j} \leq R_{n,i}\}$

4: $\mathcal{I}^{now} := \mathcal{I} \setminus \mathcal{I}^{covered}$

5: $\mathcal{J}^{now} := \mathcal{J} \setminus (\mathcal{J}^{fix0} \bigcup \mathcal{J}^{fix1})$

6: $q := 0$

7: $RECURSION(\mathcal{J}^{fix1}, \mathcal{I}^{now}, \mathcal{J}^{now}, q)$

8: **return** $\{\boldsymbol{x}_1^q \mid 1 \leq q \leq Q\}$ ▷ End of the main algorithm

9: **procedure** RECURSION($\mathcal{J}^{fix1}, \mathcal{I}^{now}, \mathcal{J}^{now}, q$) ▷ The recursion precedure

10: $\mathcal{J}^{now} := \mathcal{J}^{now} \setminus \{j \mid d_{i,j} > R_{n,i}, \forall i \in \mathcal{I}^{now}\}$

11: **if** $\mathcal{J}^{now}$ is empty **then**

return
else
Choose a j_0 in $\mathcal{J}^{now}$ ▷ Choose randomly or under a specific rule
$\mathcal{J}^{now} := \mathcal{J}^{now} \setminus \{j_0\}$
$\mathcal{J}_{in}^{fix1} := \mathcal{J}^{fix1} \bigcup \{j_0\}$ ▷ Let j_0 in
$\mathcal{I}_{in}^{now} := \mathcal{I}_{in}^{now} \setminus \{i \mid d_{i,j_0} \leq R_{n,i}\}$
if $\mathcal{I}_{in}^{now}$ is empty **then** ▷ A feasible solution is found
$q := q + 1$
$\boldsymbol{x}_1^q := \{x_{1,j}^q = 1, \forall j \in \mathcal{J}^{fix1}$ and $x_{1,j}^q = 0, \forall j \notin \mathcal{J}^{fix1}\}$
if q ¿= Q **then return**
end if
else
$RECURSION(\mathcal{J}_{in}^{fix1}, \mathcal{I}_{in}^{now}, \mathcal{J}^{now}), q$
end if
$RECURSION(\mathcal{J}^{fix1}, \mathcal{I}^{now}, \mathcal{J}^{now}), q$ ▷ Let j_0 out
end if
end procedure

Algorithm 5 Branch-and-Price for EVCE

Input: p^{EVCE}, tolerance ϵ

Output: Optimal solution $(\boldsymbol{x}^*, \boldsymbol{y}^*)$

$p^{MP} \leftarrow p^{EVCE}$ ▷ Reformulate the problem into the (MP)
$(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow$initial feasible solution ▷ Initialize best feasible solution

3: $B^{up} \leftarrow obj(\boldsymbol{x}^*, \boldsymbol{y}^*)$ ▷ Set initial upper bound

4: $\mathcal{P} \leftarrow \{p^{MP}\}$ ▷ Initialize subproblem set

5: **while** $\mathcal{P} \neq \varnothing$ **do**

6: choose a p_ℓ from $\mathcal{P}$, $\mathcal{P} \leftarrow \mathcal{P} \backslash \{p_\ell\}$

7: initialize column pool $\mathcal{Q}^\ell = \bigcup_{n \in \mathcal{N}} \mathcal{Q}_n^\ell$

8: $p_\ell^{RMP} \leftarrow DW(p_\ell)$ with $\mathcal{Q}^\ell$ ▷ Dantzig-Wolfe decompostion

9: **if** p_ℓ^{RMP} is infeasible **then**

10: **goto** 5

11: **else**

12: $[(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{y}}), (\boldsymbol{\pi}, \boldsymbol{\mu})] \leftarrow solve(p_\ell^{RMP})$

13: ▷ optimal solution $(\widehat{\boldsymbol{x}}, \widehat{\boldsymbol{y}})$, dual solution $(\boldsymbol{\pi}, \boldsymbol{\mu})$

14: **if** $\widehat{\boldsymbol{x}}$ are binary **then** ▷ A better feasible solution is found

15: $(\boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow (\widehat{\boldsymbol{x}}, \lceil \widehat{\boldsymbol{y}} \rceil)$ ▷ Update the best feasible solution

16: $B^{up} \leftarrow obj(\boldsymbol{x}^*, \boldsymbol{y}^*)$ ▷ Update the upper bound

17: **end if**

18: $\{p_\ell^{SP(n)} \mid \forall n \in \mathcal{N}\} \leftarrow$ pricing problem at node n with $(\boldsymbol{\pi}, \boldsymbol{\mu})$

19: $(\widetilde{\boldsymbol{x}}_n, \widetilde{\boldsymbol{y}}_n) = solve(p_\ell^{SP(n)})$ for all $n \in \mathcal{N}$

20: **if** $\exists n \in \mathcal{N}$ s.t. $obj(\widetilde{\boldsymbol{x}}_n, \widetilde{\boldsymbol{y}}_n) < 0$ **then** ▷ New columns are found

21: $\mathcal{Q}_n^\ell = \mathcal{Q}_n^\ell \bigcup \{(\widetilde{\boldsymbol{x}}_n, \widetilde{\boldsymbol{y}}_n)\}$

22: **goto** 8

23: **else**

24: $B_{low} = obj(\widetilde{\boldsymbol{x}}_n, \widetilde{\boldsymbol{y}}_n)$

25: **if** $B_{low} > B^{up} - \epsilon$ **then** ▷ This branch is not promising

26: **goto** 5

else

  $\{p_\ell^k \mid k = 1, ..., K_\ell\} \leftarrow branch(p_\ell)$

  $\mathcal{P} \leftarrow \mathcal{P} \bigcup \{p_\ell^k \mid k = 1, ..., K_\ell\}$

  goto 5

end if

end if

end if

end while

return $(\boldsymbol{x}^*, \boldsymbol{y}^*)$

Algorithm 6 A recursion method to get columns for SP(n)

Input: p, SP(n),

Output: $\{(\boldsymbol{x}_n^q, \boldsymbol{y}_n^q) \mid 1 \leq q \leq Q\}$

$\mathcal{J}^{fix0} := \{j \in \mathcal{J} \mid x_{1,j}$ is fixed to 0 in p at node 1 $\}$

$\mathcal{J}^{fix1} := \{j \in \mathcal{J} \mid x_{1,j}$ is fixed to 1 in p at node 1 $\}$

$\mathcal{I}^{covered} := \{i \in \mathcal{I} \mid \exists j$ s.t. $j \in \mathcal{J}^{fix1}$ and $d_{i,j} \leq R_{n,i}\}$

$\mathcal{I}^{now} := \mathcal{I} \setminus \mathcal{I}^{covered}$

$\mathcal{J}^{now} := \mathcal{J} \setminus (\mathcal{J}^{fix0} \bigcup \mathcal{J}^{fix1})$

$Q := 0$

$RECURSION(\mathcal{J}^{fix1}, \mathcal{I}^{now}, \mathcal{J}^{now}, Q)$

return $\{\boldsymbol{x}_1^q \mid 1 \leq q \leq Q\}$ ▷ End of the main algorithm

procedure RECURSION($\mathcal{J}^{fix1}, \mathcal{I}^{now}, \mathcal{J}^{now}, Q$) ▷ The recursion precedure

  $\mathcal{J}^{now} := \mathcal{J}^{now} \setminus \{j \mid d_{i,j} > R_{n,i}, \forall i \in \mathcal{I}^{now}\}$

  if $\mathcal{J}^{now}$ is empty **then**

return

else

Choose a j_0 in $\mathcal{J}^{now}$ ▷ Choose randomly or under a specific rule

$\mathcal{J}^{now} := \mathcal{J}^{now} \setminus \{j_0\}$

$\mathcal{J}_{in}^{fix1} := \mathcal{J}^{fix1} \bigcup \{j_0\}$ ▷ Let j_0 in

$\mathcal{I}_{in}^{now} := \mathcal{I}_{in}^{now} \setminus \{i \mid d_{i,j_0} \leq R_{n,i}\}$

if $\mathcal{I}_{in}^{now}$ is empty **then** ▷ A feasible solution is found

$Q := Q + 1$

$\boldsymbol{x}_1^Q := \{x_{1,j}^Q = 1, \forall j \in \mathcal{J}^{fix1} \text{ and } x_{1,j}^Q = 0, \forall j \notin \mathcal{J}^{fix1}\}$

else

$RECURSION(\mathcal{J}_{in}^{fix1}, \mathcal{I}_{in}^{now}, \mathcal{J}^{now})$

end if

$RECURSION(\mathcal{J}^{fix1}, \mathcal{I}^{now}, \mathcal{J}^{now})$ ▷ Let j_0 out

end if

end procedure

Appendix B

Proof of Theorem 3.1.2

Theorem 3.1.2 *If* $\boldsymbol{y_L^*} = \{y_{n,j}^* \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is the optimal solution of REVCE(*$\bar{\boldsymbol{x}}$*), then* $\boldsymbol{y_I^*} = \{\lceil y_{n,j}^* \rceil \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$ *is an optimal solution of EVCE(*$\bar{\boldsymbol{x}}$*).*

Proof. The proposition will be proved by contradiction. It is obvious that $\boldsymbol{y_I^*}$ is feasible to EVCE($\bar{\boldsymbol{x}}$) by Theorem 3.1.1. If $\boldsymbol{y_I^*}$ is not optimal to EVCE($\bar{\boldsymbol{x}}$), we denote an optimal solution of program (3.1.2) as $\boldsymbol{y_I'} = \{y_{n,j}' \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\}$. So $f_{\bar{x}}(\boldsymbol{y_I'}) < f_{\bar{x}}(\boldsymbol{y_I^*})$. Then we can prove this theorem by creating a new solution $\boldsymbol{y_L'}$ to (3.1.3) with an objective value strictly smaller than the objective value of $\boldsymbol{y_L^*}$, which contradicts to the fact that $\boldsymbol{y_L^*}$ is optimal to REVCE($\bar{\boldsymbol{x}}$).

The new solution is

$$\boldsymbol{y_L'} = \{y_{n,j}^* + \epsilon(y_{n,j}' - \lceil y_{n,j}^* \rceil) \mid \forall n \in \mathcal{N}, \forall j \in \mathcal{J}\},$$

where ϵ is a small positive number in interval $(0, 1)$ which satisfies

$$\epsilon < \min \left\{ \frac{y_{n,j}^* - L_{n,j}^{\bar{x}}}{\lceil y_{n,j}^* \rceil - y_{n,j}'} \;\middle|\; \frac{y_{n,j}^* - L_{n,j}^{\bar{x}}}{\lceil y_{n,j}^* \rceil - y_{n,j}'} > 0, \forall n \in \mathcal{N}, \forall j \in \mathcal{J} \right\}$$

and

$$\epsilon < \min\left\{ \epsilon_{n,j} \;\middle|\; \epsilon_{n,j} = \frac{y^*_{n,j} - y^*_{a(n),j}}{\lceil y^*_{n,j} \rceil - \lceil y^*_{a(n),j} \rceil - (y'_{n,j} - y'_{a(n),j})}, \epsilon_{n,j} > 0, \forall n \in \mathcal{N}, \forall j \in \mathcal{J} \right\}$$

Obviously,

$$\begin{aligned} f_{\bar{x}}(\boldsymbol{y'_L}) - f_{\bar{x}}(\boldsymbol{y^*_L}) &= \left(\sum_{n\in\mathcal{N}} \sum_{j\in\mathcal{J}} c^{\bar{x}}_{n,j} \cdot \left[y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil) \right] + \Psi_{\bar{x}} \right) - \left(\sum_{n\in\mathcal{N}} \sum_{j\in\mathcal{J}} c^{\bar{x}}_{n,j} y^*_{n,j} + \Psi_{\bar{x}} \right) \\ &= \epsilon \left[\left(\sum_{n\in\mathcal{N}} \sum_{j\in\mathcal{J}} c^{\bar{x}}_{n,j} y'_{n,j} + \Psi_{\bar{x}} \right) - \left(\sum_{n\in\mathcal{N}} \sum_{j\in\mathcal{J}} c^{\bar{x}}_{n,j} \lceil y^*_{n,j} \rceil + \Psi_{\bar{x}} \right) \right] \\ &= \epsilon \left(f_{\bar{x}}(\boldsymbol{y'_I}) - f_{\bar{x}}(\boldsymbol{y^*_I}) \right) < 0. \end{aligned}$$

It remains to show that $\boldsymbol{y'_L}$ is feasible to REVCE($\bar{\boldsymbol{x}}$).

For any $n \in \mathcal{N}$, $j \in \mathcal{J}$, in the case of $y'_{n,j} - \lceil y^*_{n,j} \rceil \geq 0$, we have

$$y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil) \leq y^*_{n,j} + 1 \cdot (y'_{n,j} - \lceil y^*_{n,j} \rceil) = y'_{n,j} + (y^*_{n,j} - \lceil y^*_{n,j} \rceil) \leq y'_{n,j} \leq U^{\bar{x}}_{n,j},$$

and

$$y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil) \geq y^*_{n,j} \geq L^{\bar{x}}_{n,j}.$$

In the other case of $y'_{n,j} - \lceil y^*_{n,j} \rceil < 0$, $y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil) \leq y^*_{n,j} \leq U^{\bar{x}}_{n,j}$. Since both $y'_{n,j}$ and $\lceil y^*_{n,j} \rceil$ are integers, from $y'_{n,j} - \lceil y^*_{n,j} \rceil < 0$, we get $y^*_{n,j} > \lceil y^*_{n,j} \rceil - 1 \geq y'_{n,j} \geq L^{\bar{x}}_{n,j}$. Therefore, $\dfrac{y^*_{n,j} - L^{\bar{x}}_{n,j}}{\lceil y^*_{n,j} \rceil - y'_{n,j}} > 0$ and thus $\epsilon < \dfrac{y^*_{n,j} - L^{\bar{x}}_{n,j}}{\lceil y^*_{n,j} \rceil - y'_{n,j}}$. So

$$y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil) \geq y^*_{n,j} + \frac{y^*_{n,j} - L^{\bar{x}}_{n,j}}{\lceil y^*_{n,j} \rceil - y'_{n,j}} (y'_{n,j} - \lceil y^*_{n,j} \rceil) = L^{\bar{x}}_{n,j}.$$

Now we can claim that the solution $\boldsymbol{y'_L}$ satisfies constraint (3.1.3b).

For all $n \in \mathcal{N}$ and $j \in \mathcal{J}$, since $\boldsymbol{y_L^*}$ is feasible to (3.1.3), we know $y^*_{n,j} \geq y^*_{a(n),j}$. First let us consider the case in which $y^*_{n,j} = y^*_{a(n),j}$. Obviously, $\lceil y^*_{n,j} \rceil = \lceil y^*_{a(n),j} \rceil$ and

$$\left(y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil)\right) - \left(y^*_{a(n),j} + \epsilon(y'_{a(n),j} - \lceil y^*_{a(n),j} \rceil)\right) = \epsilon(y'_{n,j} - y'_{a(n),j}) \geq 0.$$

Now look at the other case where $y^*_{n,j} > y^*_{a(n),j}$. If $\lceil y^*_{n,j} \rceil - \lceil y^*_{a(n),j} \rceil - (y'_{n,j} - y'_{a(n),j}) \leq 0$,

$$\begin{aligned}
&\left(y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil)\right) - \left(y^*_{a(n),j} + \epsilon(y'_{a(n),j} - \lceil y^*_{a(n),j} \rceil)\right) \\
&= \left(y^*_{n,j} - y^*_{a(n),j}\right) - \epsilon\left(\lceil y^*_{n,j} \rceil - \lceil y^*_{a(n),j} \rceil - (y'_{n,j} - y'_{a(n),j})\right) > 0.
\end{aligned}$$

If $\lceil y^*_{n,j} \rceil - \lceil y^*_{a(n),j} \rceil - (y'_{n,j} - y'_{a(n),j}) > 0$, then $\dfrac{y^*_{n,j} - y^*_{a(n),j}}{\lceil y^*_{n,j} \rceil - \lceil y^*_{a(n),j} \rceil - (y'_{n,j} - y'_{a(n),j})} > 0$ and therefore $\epsilon < \dfrac{y^*_{n,j} - y^*_{a(n),j}}{\lceil y^*_{n,j} \rceil - \lceil y^*_{a(n),j} \rceil - (y'_{n,j} - y'_{a(n),j})}$. Now we have

$$\begin{aligned}
&\left(y^*_{n,j} + \epsilon(y'_{n,j} - \lceil y^*_{n,j} \rceil)\right) - \left(y^*_{a(n),j} + \epsilon(y'_{a(n),j} - \lceil y^*_{a(n),j} \rceil)\right) \\
&= \left(y^*_{n,j} - y^*_{a(n),j}\right) - \epsilon\left(\lceil y^*_{n,j} \rceil - \lceil y^*_{a(n),j} \rceil - (y'_{n,j} - y'_{a(n),j})\right) > 0,
\end{aligned}$$

which means the solution $\boldsymbol{y'_L}$ satisfies constraint (3.1.3c).

It is proved that $\boldsymbol{y'_L}$ is feasible to the linear program (3.1.3) and its objective value is strictly smaller than the objective value of $\boldsymbol{y_L^*}$, which is a contradiction. □

Bibliography

Ahmed, S. and Sahinidis, N. V. (2003). An approximation scheme for stochastic integer programs arising in capacity expansion. *Operations Research*, **51**(3), 461–471.

Aros-Vera, F., Marianov, V., and Mitchell, J. E. (2013). p-hub approach for the optimal park-and-ride facility location problem. *European Journal of Operational Research*, **226**(2), 277–285.

Barnhart, C., Johnson, E. L., Nemhauser, G. L., Savelsbergh, M. W., and Vance, P. H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations research*, **46**(3), 316–329.

Benati, S. and Hansen, P. (2002). The maximum capture problem with random utilities: Problem formulation and algorithms. *European Journal of Operational Research*, **143**(3), 518–530.

Bradley, S. P., Hax, A. C., and Magnanti, T. L. (1977). *Applied Mathematical Programming*. Addison-Wesley.

Church, R. and ReVelle, C. (1974). The maximal covering location problem. *Papers in regional science*, **32**(1), 101–118.

Cities, C. (2012). Plug-in electric vehicle handbook for public charging station hosts. *US Department of Energy Publication No. DOE/GO-102012-3275*.

CREA (2019). The real estate website in canada. `https://www.realtor.ca/`.

Data Management Group (2016a). Transportation tomorrow survey, university of toronto. `http://dmg.utoronto.ca/transportation-tomorrow-survey/tts-introduction`.

Data Management Group (2016b). Transportation tomorrow survey, university of toronto. `http://dmg.utoronto.ca/transportation-tomorrow-survey/origin-destination-matrices`. (Accessed July 10, 2019).

Frade, I., Ribeiro, A., Gonçalves, G., and Antunes, A. P. (2011). Optimal location of charging stations for electric vehicles in a neighborhood in lisbon, portugal. *Transportation Research Record*, **2252**(1), 91–98.

Haase, K. and Müller, S. (2014). A comparison of linear reformulations for multinomial logit choice probabilities in facility location models. *European Journal of Operational Research*, **232**(3), 689–691.

Hakimi, S. L. (1964). Optimum locations of switching centers and the absolute centers and medians of a graph. *Operations research*, **12**(3), 450–459.

He, S. Y., Kuo, Y.-H., and Wu, D. (2016). Incorporating institutional and spatial factors in the selection of the optimal locations of public electric vehicle charging facilities: A case study of beijing, china. *Transportation Research Part C: Emerging Technologies*, **67**, 131–148.

Hoffman, S. D. and Duncan, G. J. (1988). Multinomial and conditional logit discrete-choice models in demography. *Demography*, **25**(3), 415–427.

Huang, K., Kanaroglou, P., and Zhang, X. (2016). The design of electric vehicle charging network. *Transportation Research Part D: Transport and Environment*, **49**, 1–17.

Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the sixteenth annual ACM symposium on Theory of computing*, pages 302–311.

Khachiyan, L. G. (1979). A polynomial algorithm in linear programming. In *Doklady Academii Nauk SSSR*, volume 244, pages 1093–1096.

Kojima, M., Mizuno, S., and Yoshise, A. (1989). A primal-dual interior point algorithm for linear programming. In *Progress in mathematical programming*, pages 29–47. Springer.

Manne, A. S. (1961). Capacity expansion and probabilistic growth. *Econometrica: Journal of the Econometric Society*, pages 632–649.

Ministry of Transportation Ontario Government (2019). Caa's electric vehicle charging map. `https://www.caa.ca/maintenance/ev-map.html`.

MITL (2018). Electric mobility - national consumer stated preference sruvey. `http://mitl.mcmaster.ca/portfolio`. (Accessed July 20, 2019).

Murty, K. G. (1983). *Linear Programming*. Springer.

Renegar, J. (1988). A polynomial-time algorithm, based on newton's method, for linear programming. *Mathematical Programming*, **40**(1-3), 59–93.

Schrijver, A. (1998). *Theory of Linear and Integer Programming*. John Wiley & Sons.

Serra, D., Eiselt, H. A., Laporte, G., and ReVelle, C. S. (1999). Market capture models under various customer-choice rules. *Environment and Planning B: Planning and Design*, **26**(5), 741–750.

Smith, M. and Castellano, J. (2015). Costs associated with non-residential electric vehicle supply equipment: Factors to consider in the implementation of electric vehicle charging stations. Technical report.

Tong, D. and Murray, A. T. (2009). Maximising coverage of spatial demand for service. *Papers in Regional Science*, **88**(1), 85–97.

Tu, W., Li, Q., Fang, Z., Shaw, S.-l., Zhou, B., and Chang, X. (2016). Optimizing the locations of electric taxi charging stations: A spatial–temporal demand coverage approach. *Transportation Research Part C: Emerging Technologies*, **65**, 172–189.

Vanderbei, R. J., Meketon, M. S., and Freedman, B. A. (1986). A modification of karmarkar's linear programming algorithm. *Algorithmica*, **1**(1-4), 395–407.