The School of Mathematics

THE UNIVERSITY
of EDINBURGH

A Heuristic Solution for MPSTDP-S

by

Wei Li

Dissertation Presented for the Degree of
MSc in Operational Research with Data Science

August 2023

Supervised by
Dr Joerg Kalcsics

Abstract

The primary objective of this study is to devise a heuristic algorithm specifically for the scheduling aspect of the Multi-Period Service Territory Design Problem (MPSTDP). Initially, we provide a foundational overview of the problem, its practical applications, and a survey of prior research in this domain. After making necessary modifications based on previous studies, we craft a model tailored for a specific problem context. Subsequently, we propose a novel heuristic algorithm to solve this model. We arrange customers into various groups, with considerations of both distance with one group and workload balancing among different groups. The heuristic solution can be bifurcated into initialization and improvement phases. Within the improvement phase, we separately assess the merits of both relaxing and adjusting solutions. Our findings indicate that compared to a direct problem solving approach, employing the heuristic algorithm, particularly when enhanced using the relaxing method—can substantially reduce computation time, narrowing the gap with the optimal value to less than 5%.

Acknowledgments

This dissertation represents more than just a project, it signifies the culmination of my year of learning and life at the University of Edinburgh. Throughout this year in an entirely new environment, I am profoundly grateful to the classmates, teachers and advisors who have assisted and supported me at every juncture.

I extend my deepest appreciation to my family, especially my parents, for their unwavering support and encouragement. My friends, both old and new, deserve my heartfelt thanks for being my anchor in this journey.

Specifically, for this dissertation, I would like to express my utmost gratitude to my supervisor, Dr. Joerg Kalcsics. His guidance, constructive feedback, and corrections throughout the research phase have been invaluable in shaping my work and thought process.

Own Work Declaration

I hereby declare that the work presented in this dissertation is entirely my own and I have not used any sources or resources other than those listed in the bibliography. Any passages or ideas from external sources have been properly cited and credited.

25th August 2023, Edinburgh

Signature: Wei Li

Contents

List of Tables

List of Figures

1 Introduction

In this paper, the acronym MPSTDP is used to denote the Multi-Period Service Territory Design Problem, which investigates the systematic visitation of targets within a specified territorial range over a period consisting of multiple time units. The objective of the problem is to optimize the metrics of interest (such as minimize cost, distance or maximize profit) while satisfying the visitation requirements. According to Bender et al.[2], the MPSTDP can be categorized into the partition and schedule components, representing territorial division and visitation planning, respectively. This study primarily addresses the schedule aspect, and thus, the problem can be abbreviated as MPSTDP-S. More specifically, in this study, for the formulation of such problems, we consider a specific time period within which we need to periodically visit customers at known locations. The visitation cycle varies for each customer, for instance, some clients may require a visit every three weeks, some every two weeks, and others on a weekly basis. Our objective is to determine which clients to visit each week throughout the entire period in order to minimize the overall cost.

Based on the model constructed by Bender et al.[2] in 2016, we made some adjustments to tailor it to a specific company's operations. This company primarily aims to visit 40 or 50 clients within a certain territorial range over a span of either 6 or 8 weeks, with visitations scheduled based on each customer's unique frequency requirements. The model is divided into daily and weekly parts, which, when combined, constitute the overall model. Subsequently, we proposed a new heuristic algorithm for solving this model, which represents the crux of our research. Since our available data mainly focuses on weekly descriptions, our algorithm predominantly analyzes on a weekly basis. However, due to the similarity between the weekly and daily models, our approach can be easily extended to daily scenarios. The heuristic method consists of initialization and improvement phases. The improvement phase is further split into relaxing and adjusting strategies. We tested various methods and iteration criteria, and by comparing results, analyzed the relative performances of the different approaches.

The structure of the paper is organized as follows:

- Section 1: A brief introduction to the problem and the research.
- Section 2: An exploration of the problem's practical applications and a review of previous studies.
- Section 3: An analysis of specific business scenarios, with detailed descriptions of each component of the model.
- Section 4: An exhaustive description of the heuristic approach, including the initialization and improvement phases. The improvement section is further subdivided into the relaxing and adjusting strategies. The criteria for terminating iterations within the heuristic algorithm are also elaborated upon.
- Section 5: Detailed presentation of results obtained from different method combinations and iteration criteria, followed by an analysis based on these outcomes.
- Section 6: A summary and conclusion of the entire paper.

2 Background

2.1 Problem Applications

The following example illustrates the real-world application of this problem: when shopping online, many platforms, such as Amazon, offer subscription services for essential household items like paper towels and laundry detergent. Customers can choose for periodic auto-replenishment, be it weekly, bi-weekly, or monthly. From the company's perspective, they are tasked with regularly deliveries based on customer preferences regarding frequency and schedule. During these delivery operations, the company's objective is to minimize the transportation route, thereby reducing costs and enhancing profitability.

This problem and cases which like the former instance find prevalent application in real-world scenarios. For instance, in urban transit planning, public bus and subway systems have to offer regular services to passengers. Throughout a day, they need to increase frequencies during peak hours to satisfy demand, while during off-peak hours, they might reduce frequencies to minimize operational costs. Furthermore, within a designated area, stations located in the city center might experience higher passenger flow, thereby justifying the allocation of more frequent service intervals. Conversely, for stations situated in peripheral regions distant from the city center, it would be appropriate to schedule services with lower frequencies. In modern agriculture, farming techniques deploy autonomous tractors and drones at scheduled intervals to monitor the health of fields, ensuring crops receive appropriate hydration and nutrients. The monitoring frequency can vary based on the type of crop.

Besides, within home healthcare services, doctors and nurses are required to periodically visit patients. The diverse symptoms, ages and availability time of patients will vary doctors' visitation patterns. By optimizing the visitation schedule, medical professionals can serve a broader patient base and simultaneously minimize travel-related expenses.

These examples can be analogously mapped onto our problem. In urban transportation planning, transit stations can be considered as customers. Stations in peak hours or those located in bustling city centers might be analogous to high-frequency visit customers, while off-peak stations or those situated farther from city centers with lesser foot traffic might be equated to low-frequency visit customers. In the modern agriculture domain, crops sensitive to climate variations can be likened to high-frequency visit customers. For home healthcare services, elderly patients or those with severe conditions can be treated as high-frequency visit clients.

Therefore, studying MPSTDP-S holds practical significance not only in the realm of customer visits but can also be generalized and applied across various domains of life and production. In chapter 3, we will delve deeper into this problem, model it in detail, and introduce heuristic algorithms for its resolution.

2.2 Related Work

The MPSTDP, standing for the Multi-Period Service Territory Design Problem, addresses the arrangement of regular services to customers within a specific geographical range over a given period. In everyday life, there are numerous scenarios requiring periodic revisits to customers. Apart from the instances mentioned in the previous section 2.1, many scholars have studied similar problems in the past. For instance, in eras when network communication was not as advanced, sales personnel had to periodically revisit customers[5]. Additionally, there were cases of heavy machinery and other products that necessitated regular maintenance by technicians[3].

To the best of our knowledge, the formal introduction and definition of this problem originate from Bender et al.[2], who also proposed a comprehensive mixed-integer programming model. Bender and his colleagues believed that the MPSTDP can be decomposed into two sub-problems, namely

MPSTDP-Partition (MPSTDP-P) and MPSTDP-Schedule (MPSTDP-S)[2]. The MPSTDP-P pertains to the partitioning of a specified area, aligning with the widely recognized districting problem [2]. This problem involves dividing a region into sub-regions, each managed by a different individual responsible for tasks in their respective territories[2]. Kalcsics provided a detailed description of this in 2015[6], which sheds light on the various application domains of districting and reviews the array of solutions used for these partitioning problems. It notes that a district, when geographically compact with minimal deviations from a roughly circular shape, is deemed optimally designed[6]. Typical units for such partitioning encompass customers, streets, or postal code areas, which find applications in fields such as politics and education[6]. On the other hand, MPSTDP-S focuses on scheduling: over the entire period in question, it determines how the roster is arranged, i.e., when customers in each sub-region are visited [2].

In practical solutions to such problems, heuristic algorithms are often the choice of many scholars. In 2009, Ríos-Mercado and his colleagues devised a reactive GRASP approach specifically for the territory design problem (TDP)[9]. GRASP, an acronym for "Greedy Randomized Adaptive Search Procedure", is a metaheuristic algorithm tailored for combinatorial optimization challenges. This procedure alternates between two principal phases in a loop: the construction phase and the local search phase. The method developed by Ríos-Mercado and his team integrates several features[9]. Notably, it exhibits reactivity by permitting the self-adjustment of the restricted candidate list quality parameter[9]. Moreover, it employs filtering to circumvent the execution of the local search phase when faced with unpromising solutions generated during the construction phase[9]. This approach was tested on several datasets[6], achieving commendable results. Subsequently, in 2016, they introduced an even more efficient GRASP variant augmented with a path relinking heuristic[8].

In 2013, Ríos-Mercado and colleagues addressed the commercial territory design problem (TDP)[10]. In their study, they introduced a mixed-integer programming model that holistically considers the number of clients, their demands, and the company's workload[10]. To solve this model, they proposed a solution procedure grounded on an iterative cut generation strategy embedded within a branch-and-bound framework[10]. This procedure is tailored to tackle large-scale instances by integrating multiple algorithmic strategies that effectively diminish the problem's size[10]. In 2015, specifically addressing the districting problem-solving, Lei and colleagues developed an adaptive large neighbourhood search metaheuristic for the Multiple Traveling Salesmen and Districting Problem with Multi-periods and Multi-depots[7]. This algorithm was evaluated on modified Solomon and Gehring & Homberger instances and yielded promising results[7].

After the formal definition of MPTSDP, in 2020, Bender et al. devised a two-stage method for parcel delivery scheduling, representing the MPSTDP-P and MPSTD-S stages respectively[1]. In the first stage, the approach districts the customers[1]. Building upon the model introduced in 2016, three more specific models were defined based on the level of detail of their input data, their expected adherence to service consistency, and the driver's contractual working hours[1]. The second stage schedules the specific visits for each day[1]. In contrast to traditional vehicle routing techniques, this approach allows for daily adaptations[1]. Moreover, extensions and elaborations on this topic have been made by other scholars. In 2020, building on the model and problem presented by Bender in 2016, Antonio Diglio and colleagues modeled the problem with stochastic demands and tested this model using real-world data[4].

In 2020, M. Gabriela Sandoval and the colleagues proposed an exact algorithm to address the districting problem with a p-center minimization function[11]. This algorithm employs a cut-generation strategy to ensure territory connectivity and demonstrates rapid convergence. The effectiveness of this algorithm was validated through tests on 140 datasets, each with 300 nodes[11]. In 2021, Lin Zhou and colleagues proposed a heuristic algorithm for a real-world territory design problem tailored for the distribution scheme of a major dairy company[12]. In their model, they adopted an approach similar to Bender's, where the objective function encompasses both districting costs (MPSTDP-P) and routing costs (MPSTDP-S)[12]. To address this challenge, they introduced a hybrid multi-population genetic algorithm, enhanced with various evolution and search techniques[12]. This algorithm demonstrated favorable outcomes when tested across numerous datasets[12].

3 Model of the MPSTDP-S

The Multi-Period Service Territory Design Problem (MPSTDP-S) and its associated linear programming model were first conceptualized by Bender et al.[2] in 2016. In this section, we provide a detailed exposition of the problem and derive a model influenced by Bender's original. The adaptation aligns more closely with the specific data and practical necessities and serves as the foundational basis for the heuristic solution explored within this project.

3.1 Problem Description

In this study, our research problem and data are sourced from a company (the name of which remains confidential). This company's product requires periodic visits to customers within specific period and spatial intervals, to inspect the product and conduct surveys on its usage. The location and visitation pattern for each customer vary. A subset of the data is presented in the table below.

Number of BasicUnits: 40									
Number of Weeks: 6									
Number of Days per Week: 3									
BasicUnits					BasicUnits				
Index	X	Y	Time	Visit Rhythm	Index	X	Y	Time	Visit Rhythm
1	3.6	7.1	21.5	2	11	4	5	24.8	3
2	3.4	9.1	23.6	1	12	4.1	1	15.5	1
3	1.1	6.9	17.2	1	13	9.7	6	20.9	3
4	9.6	0.8	17.3	2	14	8.9	6.1	14.1	1
5	6.8	8.7	12.9	2	15	5	7.6	10.6	3
6	2.7	8.6	24.1	3	16	6	9.8	22.3	2
7	1.3	0.8	20.6	2	17	3.8	9.1	29.5	2
8	4	1.9	29.4	3	18	6	4.5	17.8	3
9	8.5	7.6	14.7	1	19	8.2	2.1	23.7	3
10	9.1	3.7	18.5	1	20	9.2	0.1	17.5	2
...	...	...	...	...	...	...	...	...	...

Table 1: Part of a Data

Table 1 displays a subset of the dataset. From the first three rows, it can be observed that there are 40 BasicUnits, which represent the number of customers. Within one time period, there are 6 weeks, with 3 working days each week. The second part of the dataset provides detailed information for each customer (also known as 'BasicUnits', due to space constraints, the table only displays 20 out of the 40 customer data entries). The first column indicates the customer's Index, while the second and third columns respectively denote their X and Y coordinates. The third column, Times, represents the time spent on each single visit to the corresponding customer, excluding travel time. The fourth column specifies the visitation frequency. A value of 3 indicates visiting this customer every 3 weeks, 2 means visiting every 2 weeks, and 1 implies a weekly visit.

There are 60 datasets of like the upper one in total. Among these datasets, 30 contain data for 40 customers each, and the remaining 30 contain data for 50 customers each. The time period of some datasets is 6 weeks, while others' is 8 weeks. The locations and single visiting times for each customers are different. It's important to note that in each dataset, the visit frequency for customers is always a divisor of the corresponding total time period. For example, if a dataset has a time period of 8 weeks, then the visit frequencies for all customers in that set could only be once a week, once every 2 weeks, once every 4 weeks, or once every 8 weeks.

Based on former descriptions, the primary objective of this project is to determine which customers to visit each week within the time period (6 weeks or 8 weeks), while ensuring all customers are visited at their required frequencies. In practical, apart from the constraints specified in the dataset, there are additional constraints to be met. For instance, the workload disparity between weeks and between days within the time period should not be excessive. A significant imbalance in workload might adversely impact the stable operations of the company.

Furthermore, for any company dealing with such operations, their primary concern is to minimize operational costs and maximize profits. Hence, in an ideal situation, when formulating a model for this problem, the most intuitive value to measure travel costs would be the total distance traveled to visit the customers. However, optimizing the total distance is often only feasible in scenarios with smaller customer datasets. In actual visitation scenarios, companies might not be able to strictly follow the optimal routes calculated by the model. This is because customers might alter their available time slots for the day or even change the date of the visit, thereby disrupting the pre-planned sequence and schedule.

To solve this problem, Bender et al.[2], while initially defining the MPSTDP-S, proposed an ingenious approach. Compared to the traditional method of pre-determining the dates, sequence, and routes of visits (akin to the Travelling salesman problem), this strategy has greater flexibility, which is a important consideration for practical applications. The method involves determining a central point when confirming the clients to be visited each week. The distance from each customer to this central point is then calculated. The sum of these distances for the week is defined as the 'compactness' of that week's visits. The objective of the model is to minimize the overall compactness across all weeks, effectively balancing operational distance and flexibility in real-world scenarios.

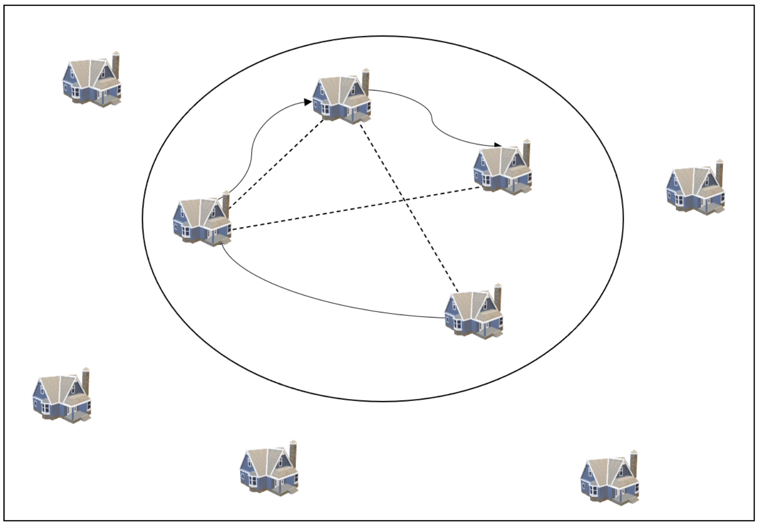


Figure 1: Example of Compactness

In Figure 1, we can more clearly see the advantages of using compactness as the value we need to optimize. The figure can be interpreted as a community represented by a dataset. Customers within the circle are considered to be allocated to the same time slot for visits. As can be observed, whether following the solid route or the dashed route, the difference in distance is not significant, and the overall travel distance appears to be optimal intuitively. Furthermore, this approach of selecting week center and minimizing the compactness for each week can be extended to the problem of determining which customers to visit on each day within the week.

Consequently, this problem can be concluded to reducing the company's costs while meeting specific constraints, which are:

- based on the customer visitation rhythms, determine which customers to visit each week, ensuring the satisfaction of customer visitation needs;
- balance the workload for each time unit (daily, weekly).

To consider about the flexibility in practical, our objective of reducing company costs has been transformed into optimizing the compactness for all weeks. Under this objective, in addition to the above two constraints, we also need to decide:

- a central point for each week to calculate the compactness.

In this project, we need to determine which customers to visit each week and which customers to visit on each working day of the week. During the modeling process, we will initially establish a whole model, incorporating all constraints concerning daily and weekly schedules. However, in practical computation, solving this model in one go would lead to prolonged computational times and a potential risk of infeasible due to excessive constraints. Consequently, following the comprehensive model, we will introduce a week-focused model. This will first ascertain the customers to visit on a weekly basis, then further subdivide visits for each day of that week. In subsequent section 3.2, we will approach this problem from the perspective of a linear programming model. We will progressively construct the model, elucidating its index, parameters, variables, objective function, and constraints. Section 3.3 will present a notation and provide an overview of the whole model.

3.2 Model Description and Definition

3.2.1 Indexes

In addressing this problem, it's evident that we require indices to represent all customers (B), all weeks (W), and all days (D). The index B is a range from 1-40 or 1-50, representing distinct customers within a dataset. The week index, W, encompasses a range of 1-6 or 1-8, signifying all weeks in the dataset's specific time period. The day index, D, covers a range from 1 up to the multiply value of the total weeks and the corresponding number of days in each week, representing every day within this period. For instance, if a dataset specifies the number of weeks as 6 and the number of days as 3, then D would span a range from 1-18.

Furthermore, to address the constraints of visitation frequency, Bender et al.[2] introduced the concept of 'week pattern'. Week pattern defines a set of week indices that need to be visited within a specific time period based on the visit rhythm. For instance, if the visitat rhythm for a customer is once every three weeks and the whole period for the corresponding dataset is 6 weeks, then the corresponding week pattern would be $\{[1,4],[2,5],[3,6]\}$. When determining which week to visit this customer, a choice is made from the three elements in this set. The table2 provides the week patterns for all visitation frequencies in a 6-week period (note that the number of weeks is divisible by the visit rhythm). From the given dataset, we can obtain the total number of weeks and all valid visitat rhythm. Using these, we can calculate all corresponding week patterns. The index P represents the sequence of all week patterns starting from 1. In this case, it spans from 1 to 12.

Number of Weeks: 6		
Week Rhythm	Week Pattern Index	Week Patterns
1	1	[1,2,3,4,5,6]
2	2	[1,3,5]
	3	[2,4,6]
3	4	[1,4]
	5	[2,5]
	6	[3,6]
6	7	[1]
	8	[2]
	9	[3]
	10	[4]
	11	[5]
	12	[6]

Table 2: An Example of Week Patterns

After determining all the week patterns, we can deduce the valid week pattern for each customer and record them under the index P_b. During the modeling in programming, P_b is a nested list with an outer length equivalent to the total number of customers, and an undefined inner length. Drawing from the above example, if a dataset contains five customers with visit rhythms of *[6,3,2,1,3]*, the corresponding P_b index would be: *[[7,8,9,10,11,12],[4,5,6],[2,3],[1],[4,5,6]]*. The introduction of the week pattern ensures that companies can schedule visits in accordance with the visit rhythm requirements from customers. By enumerating all week patterns during the preparatory phase before model-solving, we can substantially reduce the complexity of subsequent model construction and computation.

3.2.2 Parameters

In this model, the main parameters are as follows:

1. $c_{i,b}$: this parameter is a 40 × 40 or 50 × 50 matrix, determined by the number of customers, representing the distance from customer i to customer b. The formula is given as:

$$\sqrt{(x_i - x_b)^2 + (y_i - y_b)^2} \tag{3.1}$$

2. n_b: in practical, certain customers might require multiple visits during their designated weeks. n_b denotes the visit times required for a customer in a given week, and these values can be directly extracted from the given data.

3. t_b: the parameter t_b represents the time required for each visit to different customers. The values can be directly extracted from the data and this parameter is used to balance the workload across different time units (days and weeks).

4. ϕ_d: ϕ_d is used to identify the corresponding week for each given day. For instance, in a dataset encompassing 6 weeks with 3 days in each week, there are a total of 18 days spanning the entire period. Here, $\phi(1) = 1$, since the first day corresponds to the first week, and $\phi(18) = 6$ as the eighteenth day belongs to the sixth week.

5. ψ_p^w: the parameter ψ_p^w is a significant component of the model, denoting whether week w is contained in week pattern p. In the modeling phase, this is structured as a matrix formed by the total number of weeks multiplied by the quantity of all week patterns. As illustrated by the prior example, the matrix dimensions are 6 × 12.

$$\psi_p^w = \begin{cases} 1 & \text{week pattern } p \text{ contains week } w \\ 0 & \text{Otherwise} \end{cases}$$

6. μ_{week}: the parameter k represents the average workload for each week. This parameter is used to ensure a roughly equal balance of workload across different weeks. The computation method is:

$$\mu_{week} = \sum_{b \in B} (t_b \times n_b \div rhythm_b) \tag{3.2}$$

7. τ_{week}: in the practical scheduling process, it's certain that the workload for each week cannot be exactly the same. The parameter τ_{week} denotes the permitted percentage deviation from the average workload for each week. This value is assigned based on manager's judgement, and typically ranges between *0.3* to *0.4*.

8. $\mu_{day}[w]$: The parameter $\mu_{day}[w]$ represents the average daily workload. Based on our discussion earlier, during the problem-solving process, we will first determine which customers are visited each week. Consequently, the average daily workload varies across different weeks. This parameter is employed to ensure that workloads are approximately balanced across different days. The computation method is:

$$\mu_w^{day} = \sum_{b \in B'} (t_b \times n_b \div rhythm_b) \tag{3.3}$$

where $B^{'}$ is a subset of B, containing customers who arranged to the same week w among the time period.

9. τ_{day}: τ_{day} is analogous to τ_{week}, but it pertains to the allowable deviation from the average daily workload.

10. λ: when considering a holistic model that contain both weekly and daily aspects, we need to simultaneously optimize the compactness for each week and each day. However, the relative importance of these two components may have different priority in real-world operations. For instance, the company might place greater emphasis on the compactness of each week, with daily compactness being of lesser concern. This could be due to potential sudden changes in customer visiting days. The parameter λ is a value between 0 and 1, representing the importance of weekly compactness, while $(1-\lambda)$ is the importance of daily compactness. When we modeling the weekly and daily problems in separate models, this parameter might not be used.

3.2.3 Variables

In this paper, due to the introduction of compactness as a value to be optimized, for the weekly problem we need to determine which point to select as the center for calculating compactness. Concurrently, it's essential to decide which customers are allocated to this center in the given week. Specifically, we define the following two variables:

$$x_b^w = \begin{cases} 1 & \text{customer } b \text{ is the week center at week } w \\ 0 & \text{otherwise} \end{cases}$$

$$u_{ib}^w = \begin{cases} 1 & \text{customer } b \text{ is allocated to week center } i \text{ in week } w \\ 0 & \text{otherwise} \end{cases}$$

Similarly, it's necessary to determine the daily center and decide which customers are allocated to this specific day. Specifically:

$$y_b^d = \begin{cases} 1 & \text{customer } b \text{ is the day center at day } d \\ 0 & \text{otherwise} \end{cases}$$

$$v_{ib}^d = \begin{cases} 1 & \text{customer } b \text{ is allocated to day center } i \text{ in day } d \\ 0 & \text{otherwise} \end{cases}$$

Additionally, to satisfy each customer's visit rhythm, we need to determine which week pattern is allocated to each customer. The valid week pattern for each customer has already been pre-computed in the indexing section. Once the week pattern is determined, customers just need to be visited in the weeks included in the specified week pattern to satisfy their visit rhythm. Specifically:

$$g_{bp} = \begin{cases} 1 & \text{week pattern } p \text{ is assigned to customer } b \\ 0 & \text{otherwise} \end{cases}$$

Similarly, for the days, since our model does not have a requirement for the daily visit rhythm, we can define:

$$h_{bd}^w = \begin{cases} 1 & \text{visit customer } b \text{ in day } d \text{ and week } w \\ 0 & \text{otherwise} \end{cases}$$

3.2.4 Objective Function

Based on our analysis above, here, we aim to optimize the compactness. Compactness refers to the sum of distances from all points assigned to a center to the center itself over a specific period. For the week part, this can be expressed as:

$$\sum_{b\in B}\sum_{i\in B}\sum_{w\in W} n_b c_{ib} u_{ib}^w$$

For the day part, this can be expressed as:

$$\sum_{b\in B}\sum_{i\in B}\sum_{d\in \mathcal{D}} c_{ib}v_{ib}^{d}$$

When considering both components within the same model, the weight λ should be used to trade off the significance of these two parts. Therefore, the objective function can be expressed as:

$$\lambda\sum_{b\in B}\sum_{i\in B}\sum_{w\in W} n_b c_{ib} u_{ib}^{w} + (1-\lambda)\sum_{b\in B}\sum_{i\in B}\sum_{d\in \mathcal{D}} c_{ib}v_{ib}^{d}$$

Additionally, to measure the compactness of visits within the same period (weeks or days), we can also consider the sum of distances between all customers assigned to a particular center. The calculation can be defined as:

$$\lambda\sum_{b_1\in B}\sum_{i\in B}\sum_{b_2\in B}\sum_{w\in W} n_b c_{b_1b_2} u_{ib_1}^{w}u_{ib_2}^{w} + (1-\lambda)\sum_{b_1\in B}\sum_{i\in B}\sum_{b_2\in B}\sum_{d\in D} c_{b_1b_2}v_{ib_1}^{d}v_{ib_2}^{d} \tag{3.4}$$

This method is more prudent as it reduces the occasional occurrence of having customers assigned to the same center but situated far apart from other customers assigned to this center. However, given that the current dataset contains at most 50 customers, to decrease the computational complexity of the model, this paper primarily adopts the first approach to measure compactness.

3.2.5 Constraints

Based on our analysis and description of the problem before, to calculate the compactness for each week and to minimize the sum of all compactness values, we must assign a customer a center for each time unit and allocate certain other customers who need to be visited during this time period to this center.

Consequently, when formulating constraints, each week (3.9) and each day need a customer's location as center (3.15). If within a given week or day, there's a need to visit some customers, then that these customers must be allocated to a center for that week(3.7) and day(3.13). In the actual modeling process, within a constraint, we first ensure that each customer, who needs to be visited in a given week or day, is allocated to itself or another customer, where this customer itself or the other customer must serve as the center. Thus, we also require constraints (3.8 and 3.14) to ensure that the allocated customer is a center for that week or day.

In determining which customer will be visited at which week, it's imperative to adhere to their visit rhythm. As previously discussed, we have enumerated all valid week patterns for every customers. By simply selecting one pattern from each customer's valid week patterns, we can ensure that visits are scheduled in accordance with their visit rhythm. Consequently, when addressing the weekly problem, a week pattern needs to be allocated to each customer (3.6). As for the daily aspect, within each week a customer needs to be visited, they should be visited n_b times (3.12) during each visiting week. This constraint also bridges the weekly and daily components of the problem.

Additionally, constraints 3.10 to 3.11 serve to balance the workload between weeks, using the average workload as a baseline, and allowing deviations within the range of τ_{week}. The workload balance between days within the same week is achieved through constraints 3.16 and 3.17.

3.3 An Overview of the Model

This section comprehensively presents the model developed for MPSTDP-S. Section 3.3.1 summarizes all the variables, indices, and parameters used in the model, along with their explanations. Section 3.3.2 shows a complete model that includes both daily and weekly components. This full model consists of six variables and eighteen constraints. Meanwhile, section 3.3.3 presents the sub-model within the complete model dedicated to the weekly component. During the subsequent model solving process, the presence of numerous constraints and variables, combined with a dataset of 40-50 items, might make the solving process challenging and time-consuming. A more concerning situation is the model might become infeasible due to the multitude of constraints cutting the feasible region. Thus, we will first address the sub-model in section 3.3.3. Afterward, the variable results related only to the week *(x, u, g)* will be treated as parameters and fed into the full model, from which we can obtain the variable solutions related to the day *(y, v, h)*.

3.3.1 Notations

Table 3 is a table summarizing all notations used in the model.

Index Set	
B	Customers
W	Weeks in the planning horizon
D	Days in the planning horizon
P	All week patterns
P_b	Valid week patterns for customer $b \in B$
Parameters	
$c_{ib} \in R^+$	Distance from customer $i \in B$ to customer $b \in B$
$n_b \in N^+$	Number of visits of customer $b \in B$ per visiting week
$t_b^w \in R^+$	Time for serving customer $b \in B$ in week $w \in W$
$\phi(d) \in W$	Week of day $d \in D$
$\psi_p^w \in \{0,1\}$	Indicates whether week pattern $p \in P$ contains week $w \in W$ (1) or not (0)
$\mu^{week} \in R^+$	Average weekly service time
$\mu_w^{day} \in R^+$	Average daily service time
$\tau^{week} \in R^+$	Maximum allowable deviation of the actual from the average weekly service time
$\tau^{day} \in R^+$	Maximum allowable deviation of the actual from the average daily service time
$\lambda \in [0,1]$	Weight for weekly compactness
Variables Set	
$g_{bp} \in \{0,1\}$	if week pattern $p \in P_b$ is selected for customer $b \in B(1)$ or not (0)
$h_{bd}^w \in \{0,1\}$	if we visit customer $b \in B$ at week $w \in W$ and day $d \in D$ (1) or not (0)
$u_{ib}^w \in \{0,1\}$	if customer $b \in B$ is assigned to week center $i \in B$ in week $w \in W(1)$ or not (0)
$v_{ib}^d \in \{0,1\}$	if customer $b \in B$ is assigned to day center $i \in B$ on day $d \in D(1)$ or not (0)
$x_b^w \in \{0,1\}$	if customer $b \in B$ is selected as the center for week $w \in W$ (1) or not (0)
$y_b^d \in \{0,1\}$	if customer $b \in B$ is selected as the center for day $d \in D$ (1) or not (0)

Table 3: Notations

3.3.2 Complete Model

$$\min \quad \lambda \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} n_b c_{ib} u_{ib}^w + (1-\lambda)\sum_{b\in B}\sum_{i\in B}\sum_{d\in\mathcal{D}} c_{ib} v_{ib}^d \tag{3.5}$$

s.t.

$$\sum_{p \in P_b} g_{bp} = 1 \qquad b \in B \tag{3.6}$$

$$\sum_{i \in B} u_{ib}^w = \sum_{p \in P_b} \psi_p^w g_{bp} \qquad w \in W, b \in B \tag{3.7}$$

$$u_{ib}^w \leq x_i^w \qquad b, i \in B, w \in W \tag{3.8}$$

$$\sum_{b \in B} x_b^w = 1 \qquad w \in W \tag{3.9}$$

$$\sum_{b \in B, p \in P_b} t_b^w \psi_p^w g_{bp} u_{ib}^w \geq (1 - \tau^{week}) \mu^{week} \qquad w \in W, i \in B \tag{3.10}$$

$$\sum_{b \in B, p \in P_b} t_b^w \psi_p^w g_{bp} u_{ib}^w \leq (1 + \tau^{week}) \mu^{week} \qquad w \in W, i \in B \tag{3.11}$$

$$\sum_{d \in D} h_{bd}^w = n_b \times \sum_{p \in P_b} \psi_p^w g_{bp} \qquad b \in B, w \in W \tag{3.12}$$

$$\sum_{i \in B} v_{ib}^d = h_{bd}^{\phi(d)} \qquad b \in B, d \in D \tag{3.13}$$

$$v_{ib}^d \leq y_i^d \qquad b, i \in B, d \in D \tag{3.14}$$

$$\sum_{b \in B} y_b^d = 1 \qquad d \in D \tag{3.15}$$

$$\sum_{b \in B} t_b^{\phi(d)} h_{bd}^{\phi(d)} \geq (1 - \tau^{day}) \mu_{\phi[d]}^{day} \qquad d \in D \tag{3.16}$$

$$\sum_{b \in B} t_b^{\phi(d)} h_{bd}^{\phi(d)} \leq (1 + \tau^{day}) \mu_{\phi[d]}^{day} \qquad d \in D \tag{3.17}$$

$$g_{bp} \in \{1, 0\} \qquad b \in B, p \in P_b \tag{3.18}$$

$$h_{bd}^w \in \{1, 0\} \qquad b \in B, d \in D, w \in W \tag{3.19}$$

$$u_{ib}^w \geq 0 \qquad b, i \in B, w \in W \tag{3.20}$$

$$v_{ib}^d \geq 0 \qquad b, i \in B, d \in D \tag{3.21}$$

$$x_b^w \in \{0, 1\} \qquad b \in B, w \in W \tag{3.22}$$

$$y_b^d \in \{0, 1\} \qquad b \in B, d \in D \tag{3.23}$$

Constraint	Analysis
3.6	Every customer has a week pattern to cover their demands in certain frequencies
3.7	If week w is a visiting week for customer b, it will be arranged to a week center
3.8	Ensure that the 'center' i arranged to customer b in week w is a real week center
3.9	Centers' number equal to sales man's number every week
3.10,3.11	balance service time for each sales man and each weeks
3.12	visit customer b n_b times in its visiting week
3.13	if we visit customer b at day d (week $\phi(d)$), there is a corresponding day center
3.14	Ensure that the 'center' i arranged to customer b in week w is a real day center
3.15	Centers' number equal to sales man's number every wee
3.16,3.17	Balance service time for each days

Table 4: Explanation for Constraints

3.3.3 Sub-Model for Week Part

This section represents a sub-model of the complete model, where the objective function retains only the weekly compactness component. All constraints can be directly mapped to constraints 3.6 - 3.11.

$$\min \quad \sum_{b\in B}\sum_{i\in B}\sum_{w\in W} n_b c_{ib} u_{ib}^w$$

s.t.

$$\sum_{p\in P_b} g_{bp} = 1 \qquad b \in B$$

$$\sum_{i\in B} u_{ib}^w = \sum_{p\in P_b} \psi_p^w g_{bp} \qquad w \in W, b \in B$$

$$u_{ib}^w \leq x_i^w \qquad b, i \in B, w \in W$$

$$\sum_{b\in B} x_b^w = 1 \qquad w \in W$$

$$\sum_{b\in B, p\in P_b} t_b^w \psi_p^w g_{bp} u_{ib}^w \geq (1 - \tau^{week})\mu^{week} \qquad w \in W, i \in B$$

$$\sum_{b\in B, p\in P_b} t_b^w \psi_p^w g_{bp} u_{ib}^w \leq (1 + \tau^{week})\mu^{week} \qquad w \in W, i \in B$$

$$g_{bp} \in \{1, 0\} \qquad b \in B, p \in P_b$$

$$u_{ib}^w \geq 0 \qquad b, i \in B, w \in W$$

$$x_b^w \in \{0, 1\} \qquad b \in B, w \in W$$

4 Heuristic Solution for MPSTDP-S

In this section, we primarily describe a heuristic method to address the problem at hand. This encompasses a qualitative analysis of the issue, the initialization of the heuristic algorithm, two distinct improvement strategies (Relaxing Rearrangement and Adjusting Rearrangement), and the criteria for concluding the iterations within the heuristic approach. It's pivotal to recognize that the weekly problem is highly analogous to the daily problem.

The heuristic for the weekly issue can be seamlessly extrapolated to the daily scenario. Besides, in the context of this study, we have not obtained data pertaining to the day pattern. Consequently, beginning with this section, our methodological descriptions and result analyses will predominantly focus on the weekly problem.

4.1 Solution Description

In the previous section, we formulated a model for MPTSDP. To fit practical requirements, the model aims to optimize the sum of compactness across each week and day. While ensuring all clients adhere to their visit rhythm, the model balances the workload across different time units. The results derived from the model aim to offer as much flexibility as possible while minimizing travel costs. Within this framework, when considering both weeks and days simultaneously, a total of six variables need to be determined, with the smallest being of the second order (e.g.g) and the largest of the third order (e.g.u). During computation, the model encompasses 18 constraints: while the complexity of the majority of constraints is *O(n)*, ten are $O(n^2)$ and the remainder are $O(n^3)$.

In our dataset comprising 60 data points, there are a minimum of 40 clients over six weeks with three days per week, and at most 50 clients over eight weeks with five days per week. Therefore, to reduce the model's computational time, we plan to make the following two adjustments. First, we intend to divide the model's calculation into two separate stages. As previously discussed, we will start with a sub-model that only considers weekly related variables, constraints, and the objective function. In the second stage, the outcomes of the first step will be used as parameters to derive daily associated variable values. Secondly, we plan to propose a heuristic algorithm. Through this heuristic approach, we can attain either a complete or partial solution to the model. By iterative improvement, we aim to continuously narrow the gap between the objective function values derived from the heuristic and the optimal solution. Given the similarities between the weekly and daily models, we anticipate leveraging analogous heuristic algorithms for both. In the subsequent analyses, we will primarily focus on the weekly components of the model.

Because of the problem's intricate constraints and the multitude of variables to determine, to narrow the gap between the heuristic algorithm's objective value and the optimal value, we might contemplate fixing only a portion of the solution derived from the heuristic method rather than the complete solution, leaving the remaining portion for the solver to tackle. For the weekly component of our model, three variables require resolution. The variable g decided which weekly pattern each customer is assigned to, thus determining the weeks a customer is visited. The variable x decides the center for each week based on customer locations. Meanwhile, u determines which center is assigned to a customer during their visit weeks. Bender et al.[2] introduced a location-allocation approach. As its name suggests, this method encompasses two stages: location and allocation. The location step primarily located the weekly centers. Following the location of these weekly centers, the solver is then used during the allocation stage to determine the week pattern for each customer and their respective assigned weekly centers. Subsequently, based on the results of the allocation, the weekly centers are repositioned and passed back to the solver for resolution, recommencing the allocation step. This process iterates until an iteration termination condition is met.

Bender et al.[2]'s heuristic algorithm fixes the centers for each week and day. Then, using a solver, it allocates week patterns to each customer to achieve as optimal a compactness value as possible under the heuristic. In this paper, we introduce an new heuristic algorithm.

We aim to impose certain constraints on the week pattern for each customer, and then employ the solver to find the week center, maximizing the compactness value under the heuristic approach. Bender's method, by setting the centers in place, relies on the solver to determine the scheduling of customer visits each week, allowing for some computational space for solver. However, diverging from this approach, if we fix all the week patterns, it implies that the weekly customer visits are already scheduled, which will bring 2 problems. Firstly, after fixing the week patterns, when determining the weekly centers, the solver would only need to identify customers positioned relatively central within each week's arranged visits to act as the center. If we consider the values of variables determined by the heuristic method as new constraints, cutting the feasible region of the original problem, this action could lead to an overly restrictive cut, possibly excluding the optimal solution of the original problem from the feasible domain. Furthermore, the original problem contains additional constraints, such as balancing the workload. If all week patterns are strictly fixed, these factors can only be roughly balanced in the heuristic. If these factors also need precise measurement within the heuristic approach, the heuristic would become excessively intricate. If they are only roughly considered or ignored entirely, the heuristic might slice the feasible domain of the original problem, potentially the problem will be infeasible.

Thus, if we aim to design a heuristic algorithm starting from the week patterns, it's essential to ensure the heuristic results have a more flexible and relaxed impact on the original problem, rather than rigidly fixing all week patterns. Consequently, in the method proposed in this project, customers are bundled together into groups. Primarily, the grouping of customers is based on the distances between them, while also roughly balancing the workload among each group. In the heuristic approach introduced here, customers within the same group either visit or not visit in the same week. In doing so, the week pattern for the customer group is essentially allocated. For instance, if a customer has a visit rhythm of *2* over a total period of *6* weeks, then their valid week patterns would be *[[1,3,5], [2,4,6]]*. If as the heuristic result, this customer is grouped with others where all members are required to visit in the third week, the customer's week pattern effectively becomes *[1,3,5]*. The above explanation is about the how to divide groups in the heuristic method in the start. The specific initialization method for these groups will be detailed in section 4.2.

After reasonably constraining the week patterns of the customers, the solver allocates the week patterns for the remaining customers and determines the center for each week. Based on the feedback received from solver, we can make some adjustments to the initial groupings, such as changing the week order associated with the groups or removing customers that are far from the week center from their respective group. This step constitutes the improvement part of the heuristic method. During the group adjustment process, the heuristic results seek a better cutting or a certain degree of relaxation within the feasible region of the original problem. The entire iterative process can be seen as a continuous allocation-location procedure, which through a few iterations, makes the results closer to the optimal value of the original problem. We will discuss the improvement part's algorithm in detail in section 4.3.

4.2 Initialize Groups Arrangement

For a heuristic method, a good initialization can reduce the number of iterations and the time needed to reach the optimal value. In the algorithm proposed in this project, when arranging groups, we mainly consider the distance among members within a group and the workload between different groups. Here we adopt a method similar to K-means clustering for grouping. K-means is a clustering method widely used in unsupervised machine learning. The K-means algorithm first randomly generates k centroids, and assigns data points closest to a particular centroid to that cluster. It then re-calculates the centroid of the cluster using all the points in that cluster, and repeats this step until there are no unassigned points (points not assigned to any of the k clusters).

In the algorithm proposed in this paper, customers are divided into n groups, where n represents the total time periods for the given dataset, typically 6 or 8 weeks. Subsequently, customers are then gradually merged into different groups. The merging rule is somewhat likely to gravitational attraction, the likelihood of merging into a particular group is directly proportional to the total visit time of the customer and inversely proportional to the distance from the centroid to the customer. The formula for calculating the total visit time for each customer is:

$$\frac{Number\ of\ Weeks}{Visit\ rhythm} \times Single\ Visit\ Time$$

Firstly, select the n customers with the highest total visit times to serve as the initial centroids, and use their respective total visit times as the total visit times for each group. To roughly balance the total workload across the groups, when choosing the next group to be allocated, select the one with the current smallest total time. Compute the values of the total times of all unassigned customers to their distance from the centroid of this group. Add the customer with the highest value of this ratio to the group. After adding, update the centroid position and total visit time of the group. Repeat this step until no customers remain unassigned. For a detailed step-by-step breakdown of the algorithm, please refer to Algorithm 1 and 2. Algorithm 1 aims to the index of customer that should be allocated to the input group. Algorithm 2 constitutes the primary portion of the group initialization process.

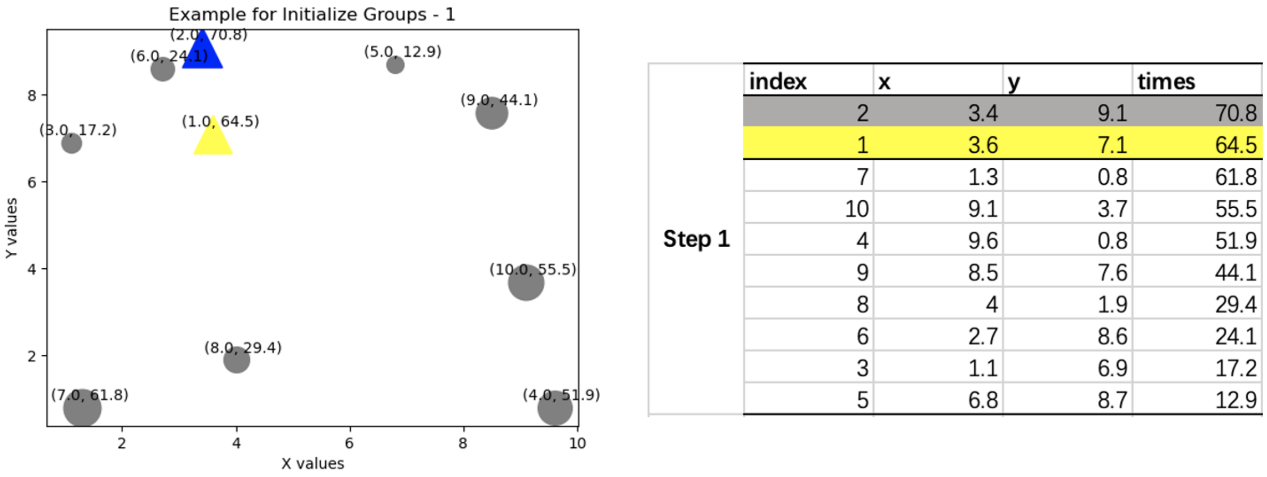
Algorithm 1: Next Customer to Groups

```
Input: Customers(X,Y,Total Times),Group Center,Rest
$evaluated \leftarrow +\infty$
foreach $i$ in Rest do
    $eva\_value \leftarrow Total\ Times[i] \div distance(i, Group\ Center)$
    if $eva\_value \leq evaluated$ then
        $evaluated \leftarrow eva\_value$
        $NC \leftarrow i$
Output: NC (Next Customer's Index)
```

Algorithm 2: Initialize Groups

```
Input: Customers(X,Y,Total Times),Number of Weeks/Days
$n \leftarrow$ Number of Weeks/Days
$Groups(customers, x, y, total\ times) \leftarrow$ an empty list with length n
$Select \leftarrow$ an empty list, $Rest \leftarrow Customers(X, Y, Total\ Times)$
$i \leftarrow 1$
for $i \leq n$ do
    $k \leftarrow$ a Customer Index (Customer in Rest with largest Total Times)
    $Groups[i][customers] \leftarrow Groups[i][customers].append(k)$
    $Groups[i][total\ times] \leftarrow$ sum of group i customers' total time
    $Groups[i][x, y] \leftarrow$ mean of group i customers' x,y
    $Select \leftarrow Select.append(k)$
    $Rest \leftarrow Rest.delete(k)$
    $i \leftarrow i + 1$
while $length(Rest) > 0$ do
    $g \leftarrow$ a Group's Index (Group with smallest Total Times for the corresponding group)
    $NC \leftarrow Next\ Customer\ to\ Groups(Customers, g, Rest)$
    $Groups[g][customers] \leftarrow Groups[i][customers].append(NC)$
    $Groups[g][total\ times] \leftarrow$ sum of group g customers' total time
    $Groups[g][x, y] \leftarrow$ mean of group g customers' x,y
    $Select \leftarrow Select.append(NC)$
    $Rest \leftarrow Rest.delete(NC)$
Output: Groups
```

The following figure illustrates an example of initial groupings. In this example, there are 2 weeks with a total of 10 customers. On the right side of Figure 2, one can see the indices of all customers, the x and y values of their locations, and the pre-computed total visit time for each customer. The table on the right side of Figure 2 arranges the data for all customers in descending order based on their total visit times. The left side of Figure 2 provides a schematic representation of all customers' locations and workloads. The larger the area of the point, the greater the total visit time. Triangular points represent customers that have been allocated to a group (with yellow and blue indicating different groups), while gray circular points represent unassigned customers. From Figure 2, it is evident that we have assigned the customers with the longest total times, namely the 2nd and 1st customers, to two distinct groups.

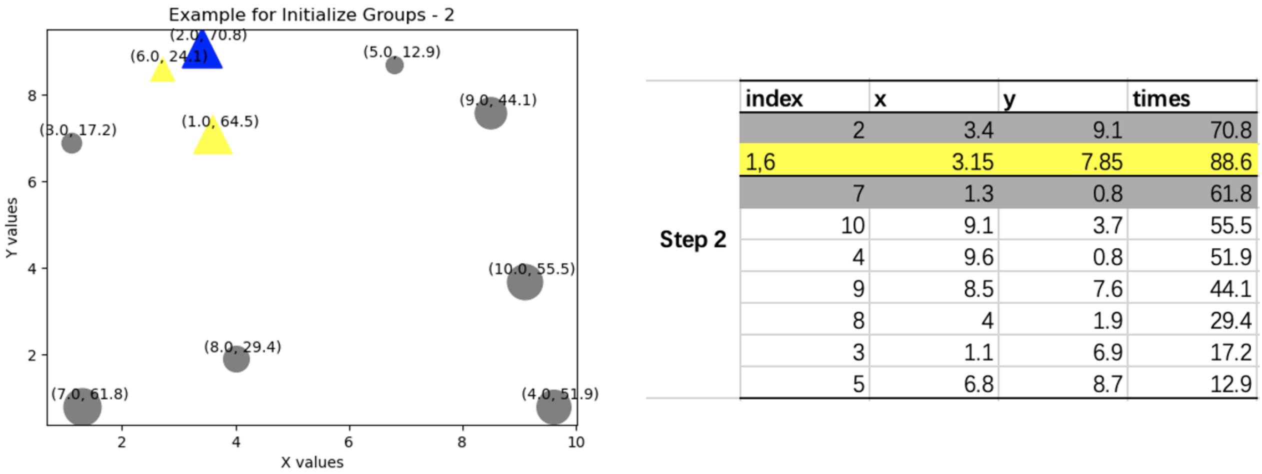


Step 1

index	x	y	times
2	3.4	9.1	70.8
1	3.6	7.1	64.5
7	1.3	0.8	61.8
10	9.1	3.7	55.5
4	9.6	0.8	51.9
9	8.5	7.6	44.1
8	4	1.9	29.4
6	2.7	8.6	24.1
3	1.1	6.9	17.2
5	6.8	8.7	12.9

Figure 2: Initialize Groups Example (Step 1)

Since the total visit time for the yellow group *(64.5)* is relatively smaller, our next customer to be grouped will be assigned to the yellow group. Among all customers, the sixth customer's total visit time divided by the distance to the yellow group (which currently includes customer no.1) is the largest. Hence, it is allocated to the yellow group. The group's center point is then updated *(x: (3.6+2.7)/2 = 3.15, y: (7.1+8.6)/2 = 7.85)* along with its total visit time *(64.5+24.1 = 88.6)*, detailed in Figure 3.



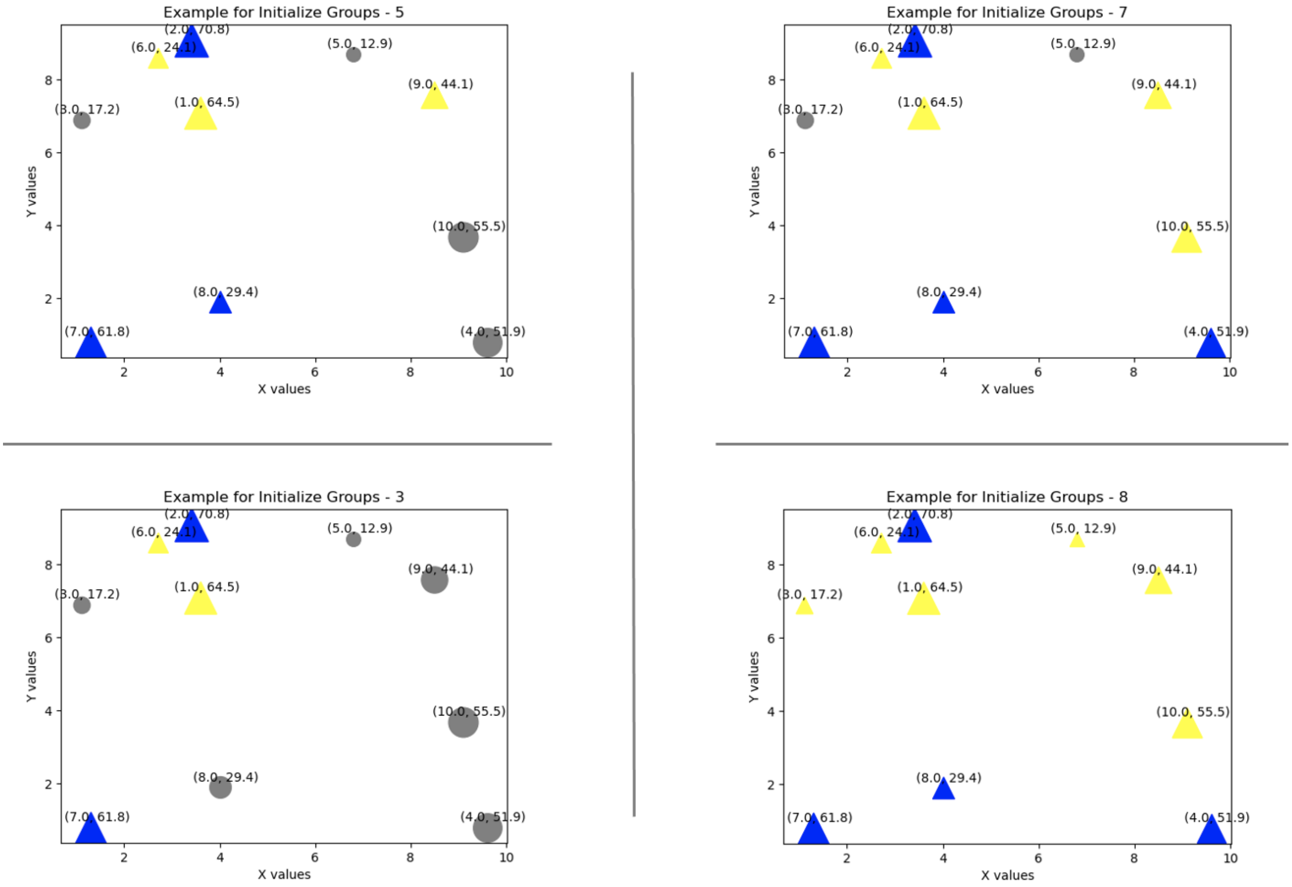
Step 2

index	x	y	times
2	3.4	9.1	70.8
1,6	3.15	7.85	88.6
7	1.3	0.8	61.8
10	9.1	3.7	55.5
4	9.6	0.8	51.9
9	8.5	7.6	44.1
8	4	1.9	29.4
3	1.1	6.9	17.2
5	6.8	8.7	12.9

Figure 3: Initialize Groups Example (Step 3,5,7,9)

The four sections in the Figure 4 respectively represent the 3rd, 5th, 6th, and 9th steps of the grouping process. We can observe that, in the end, the yellow group comprises customers *[1,3,5,6,9,10]* with a total visit time of 218.3, while the blue group includes customers *[2,4,7,8]* with a total visit time of

213.9.



Example for Initialize Groups - 5

Example for Initialize Groups - 7

Example for Initialize Groups - 3

Example for Initialize Groups - 8

Figure 4: Initialize Groups Example (Step 2)

As the limited size of this data sample and considering that this is an initial grouping, we cannot achieve perfectly grouping. However, as observed from the fourth figure in Figure 4, the geographical locations of the customers in the yellow group are relatively more clustered, and the total work time difference between the two groups is minimal.

4.3 Rearrangement Groups

After initializing the groups, it is essential to augment our original model with an additional constraint: members of the same group must either be concurrently visited or not visited during the specified week, to apply our heuristic algorithm. Consequently, the following parameters need to be added to the original model:

- G_w: A list of length w, where w represents the total number of weeks included in the dataset

In this list, each element is a list of variable length. Every sub-list contains the indices of all customers that form the group corresponding to a given week, which are derived from the results of the previous section 4.2.

And for the constraints of the original model, we need to add:

$$u^w_{i,b_1} = u^w_{i,b_2} \qquad w \in W, b_1 \in G_w, b_2 \in G_w, i \in B \tag{4.1}$$

In the additional constraints, the optimal value under the heuristic solution can be derived, as well as the identified center and the weekly visitation schedule. These results will be used in the subsequent optimization of the groupings. In this section, we will mainly focus on two aspects:

1. How to optimize the initial group (or the group used in the previous iteration) based on the centers to ensure that the solution of the subsequent iteration is closer to the initial solution of the primal problem.

2. How to determine the termination criteria for these two iterative steps.

4.3.1 Relaxing-Rearrangement

After obtaining the centers for each week, we can remove customers from the corresponding weekly group if they are located relatively far from the center. After exclusion, the modified groups are reintroduced to the model to solve for the new centers and weekly customer allocations. In the heuristic approach presented in this paper, the formula used to determine whether a customer should be removed from the group is as follows:

$$evaluate = mean(distance(center_w, Groups_{w,b})) + std(distance(center_w, Groups_{w,b})) \tag{4.2}$$

$$distane(center_w, Groups_{w,b}) \geq evaluate \tag{4.3}$$

Here, we first compute the distance between all customers within a group and the center for the given week. Any customer whose distance exceeds the average distance plus one times the standard value will be removed from that group.

Every adjustment to the groups essentially modifies the constraints of the original problem based on the heuristic solution. Each time we remove members far from the center, we are essentially performing a gradual relaxation of the problem's feasible region. This ensures that the optimal value in the next iteration is less than or equal to the optimal value of the current iteration. The specific algorithm is presented as Algorithm 3.

Algorithm 3: Relaxing Rearrangement Groups

```
Input: $Groups_{w,b}$, $Centers_w$
$G\_Compactness_{w,b} \leftarrow$ an list to record distances from $center_w$ to $Groups_{w,b}$
$Rest \leftarrow$ an list to record customers deleted from groups
foreach $w$ $in$ $W$ do
    foreach $binGroups_w$ do
        $G\_Compactness_{w,b} \leftarrow G\_Compactness_{w,b}.append(distance(centers_w, Groups_{w,b})$
foreach $w$ $inW$ do
    $evaluate \leftarrow mean(G\_Compactness_{w,b}) + std(G\_Compactness_{w,b}))$ $b \in Groups_w$
    foreach $b$ $in$ $Groups_w$ do
        if $G\_Compactness_{w,b} \geq evaluate$ then
            $Rest \leftarrow Rest.append(b)$
            $Groups_w \leftarrow Groups_{w,b}$
Output: $Groups_{w,b}$, $Rest$
```

The following is a detailed illustration of the method. The data for this example is sourced from the dataset ’Data_40_6_4_4.txt’. The initial groupings, as well as the centers resulting from these groupings based on the group initialization and heuristic method, are presented.

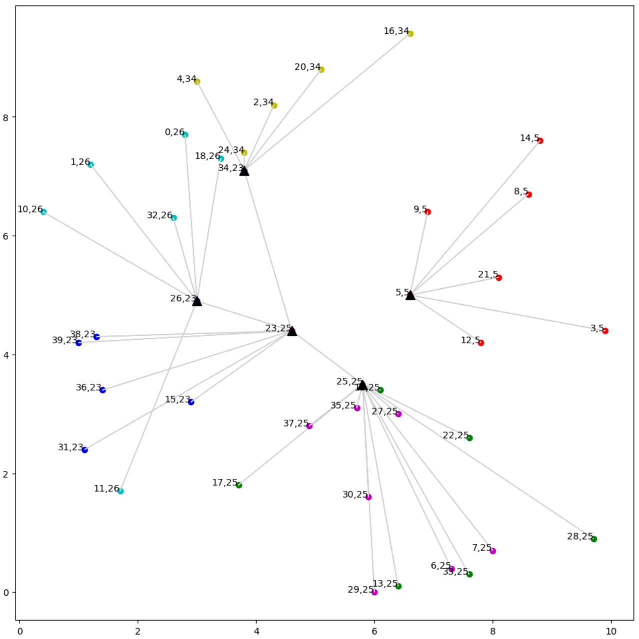


Figure 5: Relaxing Rearrangement Method Example 1

In Figure 5, each data point represents a distinct customer. The first number in the label signifies the customer's index, while the second denotes the associated center index. Different colors indicate separate groups, and a gray line connects each customer to their respective center. The allocation of customers and their respective weekly centers for each group can be more clearly and intuitively visualized in the six subfigures below.

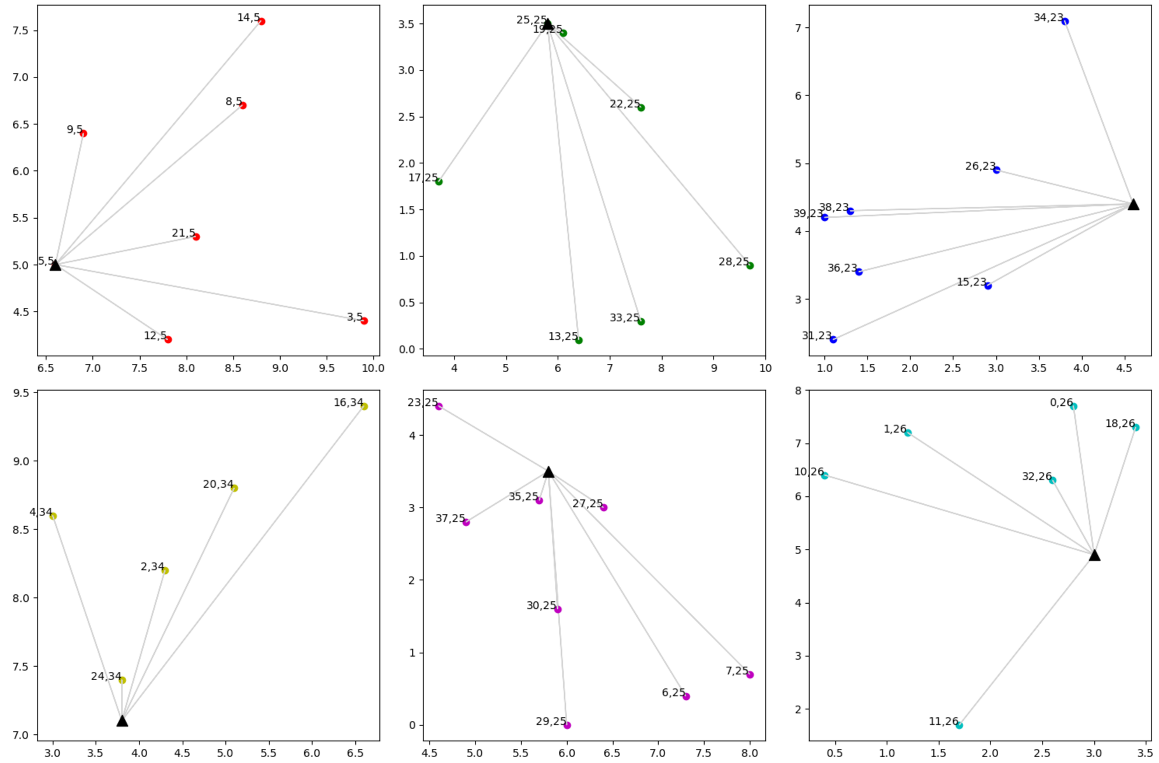


Figure 6: Relaxing Rearrangement Method Example 2

In the Relaxing-Rearrangement approach, we need to remove members from each group who are significantly distant from the center (specifically, those whose distance exceeds the group's average distance to the center by more than two standard deviations). In this example, customers who are deleted from groups are *[14, 3, 28, 31, 16, 7, 29, 6, 11]*. Following this removal, the arrangement of each group is depicted in the Figure 7.

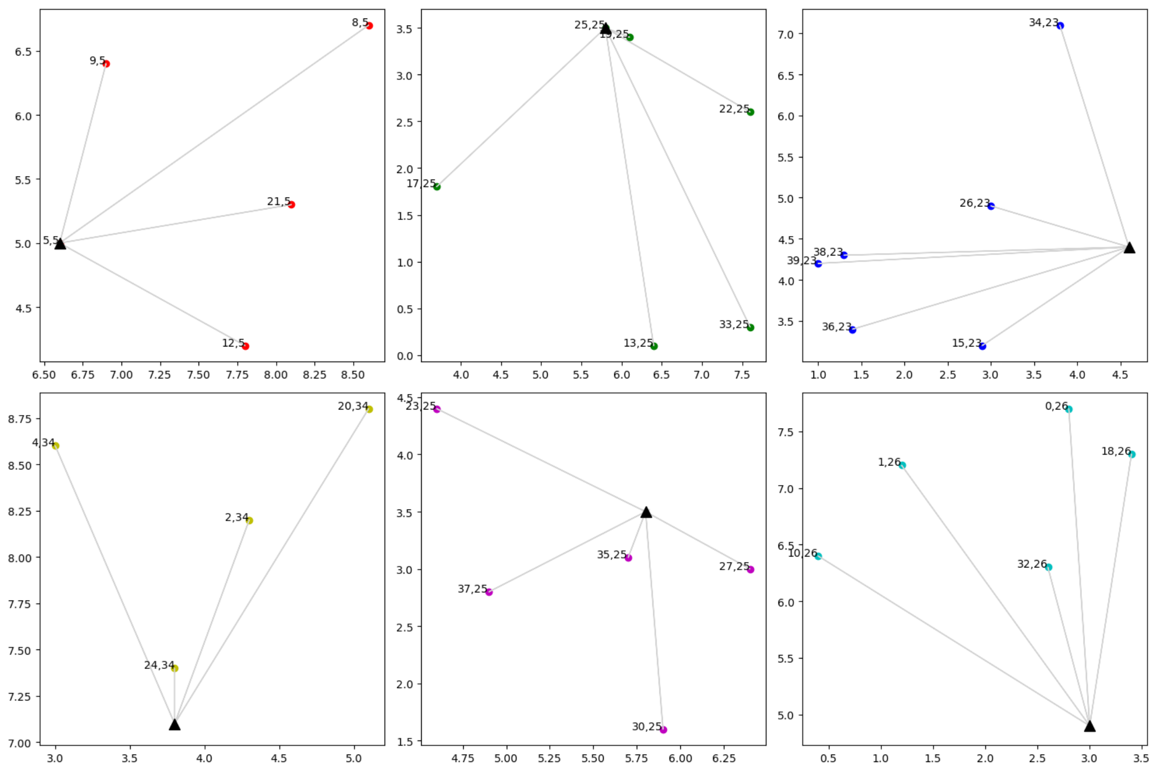


Figure 7: Relaxing Rearrangement Method Example 3

The specific allocations are detailed in the Table 5

Weeks	Centers	Group (before relaxing)	Groups (after relaxing)
1	5	[9, 5, 12, 21, 8, 14, 3]	[9, 5, 12, 21, 8]
2	25	[33, 13, 28, 22, 25, 19, 17]	[33, 13, 22, 25, 19, 17]
3	23	[31, 38, 36, 39, 34, 26, 15]	[38, 36, 39, 34, 26, 15]
4	34	[16, 20, 2, 24, 4]	[20, 2, 24, 4]
5	25	[7, 29, 6, 30, 27, 35, 37, 23]	[30, 27, 35, 37, 23]
6	26	[10, 1, 32, 18, 0, 11]	[10, 1, 32, 18, 0]

Table 5: First Iteration Week Groups and Centers

Direct removal is a straightforward approach and is guaranteed to be effective. However, one critical consideration when using this method is to control the looping conditions. If the number of iterations is too large, there's a risk that too many customers will be excluded, leaving them unassigned to any group. In such cases, the use of the heuristic algorithm becomes counterproductive. Moreover, more time might be expended compared to directly solving the problem without the heuristic, especially if iterations lead to extensive removals across all groups. Nevertheless, this method is effective. Results show that, within a limited number of iterations, the gap between the outcome of this method and the optimal value of the original problem can be reduced to a desirable amount. Furthermore, this approach tends to consume less time compared to the non-heuristic strategy. Specific details and comparisons will be presented in the results section 5.

4.3.2 Adjusting-Rearrangement

Similar to the process described in the previous subsection, we provide the initial groups to the solver. The solver then determines the centers for each week and the distribution of visit times for each customer. In doing so, we may observe certain customers in some weekly groups that are significantly distanced from the center of that week. Contrary to the previous subsection, in this rearranging method, "adjusting" refers to reallocating these distant customers to weeks where the geographical center is closer and where the workload can be balanced within a certain range, rather than directly removing them and leaving them unassigned to any group.

It's worth to note that when rearranging customers to groups, we need to consider the implications for their week patterns. The group represents a fixed visiting week for the customers within it, which indirectly determines the customer's week pattern. For instance, if a customer originally belongs to the first group with a week pattern of *[1,3,5]* and *[2,4,6]*, being reassigned to the third or fifth group would not change the end result. Thus, when reallocating, the week pattern should also be taken into account. However, there is an exception: if a customer's week pattern is 1, which means we need to visit this customer every week, then their assignment to any group won't impact their week pattern.

From the above analysis, it's clear that the adjusting method is more complex than the relaxation method mentioned in the previous subsection. In practical application, we will follow these steps:

1. Initially select the farthest group from the center in each week(or any customer whose distance exceeds the average distance plus one times the standard value).

2. If the customer's visit rhythm is not 1, choose their alternative groups. These are all the weeks that include week patterns outside of the original group's week pattern.

3. Reassign the customers removed from the original group to one of the alternative groups.

The specific algorithm is presented below. Based on our previous description, the Adjusting-Rearrangement process primarily consists of three algorithms. Algorithm 4 aims to identify and remove members that are relatively distant from the center of their respective groups. This step bears a resemblance to the Relaxing-Rearrangement method mentioned in section 4.3.1. However, here we need to keep track of the removed members and their original group affiliations.

Algorithm 4: Adjusting Rearrangement Groups - Select

```
Input: $Groups_{w,b}$, $Centers_w$
$G\_Compactness_{w,b} \leftarrow$ an list to record distances from $center_w$ to $Groups_{w,b}$
$Rest \leftarrow$ an list to record customers deleted from groups
$Original\_Group \leftarrow$ an list to record deleted customer's original group index
foreach $w$ in $W$ do
    for $b \in Groups_w$ do
        $G\_Compactness_{w,b} \leftarrow G\_Compactness_{w,b}.append(distance(centers_w, Groups_{w,b})$
foreach $w$ in $W$ do
    $evaluate \leftarrow mean(G\_Compactness_{w,b}) + std(G\_Compactness_{w,b}))\ b \in Groups_w$
    foreach $b$ in $Groups_w$ do
        if $G\_Compactness_{w,b} \geq evaluate$ then
            $Rest \leftarrow Rest.append(b)$
            $Groups_w \leftarrow Groups_{w,b}$
            $Original\_Group \leftarrow Original\_Group.append(w)$
Output: $Groups_{w,b}$, $Rest$, $Original\_Group$
```

Algorithm 5 aims to determine the valid groups to which the removed members can be allocated. The weeks represented by these viable groups should not overlap with any of the weeks in the week pattern of their original group.

Algorithm 5: Adjusting Rearrangement Groups - Valid Groups

```
Input: $Rest$, $Original\_Groups$, $psi$, $P_b$
$ValidGroups \leftarrow$ empty list
foreach $b$ in $Rest$ do
    $SubValidGroups \leftarrow$ empty list
    foreach $p$ in $P_b[b]$ do
        if $\psi[p, Original\_Groups[b]] = 0$ then
            foreach $p1$ in range of $\psi[p]$ do
                if $\psi[p, p1] > 0$ then
                    Append $p1$ to $SubValidGroups$
    Append $SubValidGroups$ to $ValidGroups$
Output: $ValidGroups$
```

Algorithm 6 aims to reallocate customers, who have not been assigned to any group, by drawing from the viable groups. This process likes the initialization phase, taking both workload and distance into consideration. It's important to note that some customers have a visitation frequency of once per week, they require visits every week. Assigning these customers to a particular group does not impact the outcome of the problem, so they are not reallocated in this phase.

Algorithm 6: Adjusting Rearrangement Groups - Rearrangement

```
Input: $Rest$, $Groups$, $centers$, $Parameters$, $BasicUnits$, $ValidGroups$, $c$
$N \leftarrow Parameters['NumberofWeeks']$
$Times \leftarrow (N/BasicUnits[:,4]) \times BasicUnits[:,3]$
$sorted\_indices \leftarrow$ argsort in descending order of $Times[Rest]$
$Rest2 \leftarrow Rest$
foreach $i$ in $sorted\_indices$ do
    if $ValidGroups[i]$ is empty then
        continue
    $Eva\_Time\_Dis \leftarrow$ empty list
    foreach $j$ in $ValidGroups[i]$ do
        $sumption \leftarrow \sum Times[Groups[j]]$
        Append $sumption/c[centers[j], Rest[i]]$ to $Eva\_Time\_Dis$
    $Add \leftarrow$ index of minimum in $Eva\_Time\_Dis$
    $min\_group \leftarrow ValidGroups[i][Add]$
    Append $Rest[i]$ to $Groups[min\_group]$
    Remove $Rest[i]$ from $Rest2$
Output: $Groups$, $Rest2$
```

The following example provides a detailed demonstration of this method. The data used for this example is consistent with that employed in the preceding subsection. Recall that Table 5 displays the initial groupings and, upon using the solver to determine the weekly centers, lists the indices of customers who remain ungrouped.

Weeks	Centers	Group(before relaxing)	Group(after relaxing)	Delete Members
1	5	[9, 5, 12, 21, 8, 14, 3]	[9, 5, 12, 21, 8]	[14,3]
2	25	[33, 13, 28, 22, 25, 19, 17]	[33, 13, 22, 25, 19, 17]	[28]
3	23	[31, 38, 36, 39, 34, 26, 15]	[38, 36, 39, 34, 26, 15]	[31]
4	34	[16, 20, 2, 24, 4]	[20, 2, 24, 4]	[16]
5	25	[7, 29, 6, 30, 27, 35, 37, 23]	[30, 27, 35, 37, 23]	[7,29,6]
6	26	[10, 1, 32, 18, 0, 11]	[10, 1, 32, 18, 0]	[11]

Table 6: Removed Customers

The Week Patterns corresponding to these customers are displayed in Table 7.

Weeks	**Delete Members**	**Week Patterns**
1	14	[1,2]
1	3	[1,2]
2	28	[0]
3	31	[0]
4	16	[0]
5	7	[0]
5	29	[0]
5	6	[3,4,5]
6	11	[3,4,5]

Table 7: Removed Customers Week Patterns

For this dataset, all week patterns show in Table 8.

Pattern Index	**Week 1**	**Week 2**	**Week 3**	**Week 4**	**Week 5**	**Week 6**
0	1	1	1	1	1	1
1	1	0	1	0	1	0
2	0	1	0	1	0	1
3	1	0	0	1	0	0
4	0	1	0	0	1	0
5	0	0	1	0	0	1

Table 8: Example Data's All Week Patterns

For instance, consider Customer 14. Their available Week Patterns are 1 and 2. Originally belonging to Group 1, this implies that they would either be visited or not visited during the first week, along with other members of their group. Consequently, during the rearrangement, it's imperative to avoid assigning them to any group associated with a week pattern containing the first week. As Week Pattern 1 encompasses the first week, during reallocation, Customer 14 should not be allocated to weeks 1, 3, and 5 which are covered by Week Pattern 1. Instead, he can only be placed within weeks 2, 4, and 6 as defined by Week Pattern 2. The valid weeks for the reallocation of all customers in this example can be found in Table 9.

Customer	**Valid Weeks**
14,3	2,4,6
28,31,16,7,29	∅
6,11	1,4,3,5

Table 9: Rest Customers Valid Weeks

Note that customers 28, 31, 16, 7, and 29 all have a visit rhythm of 1, which results in their valid weeks being an empty set. Subsequently, these ungrouped customers are allocated based on a method akin to that in 'initialize group', which balances both workload and distance considerations. To be specific, customers number 14, 3, and 6 are assigned to the group corresponding to the fourth week, while customer number 11 is allocated to the group for the fifth week. The schematic representation of the reorganized groups is depicted in Figure 8.

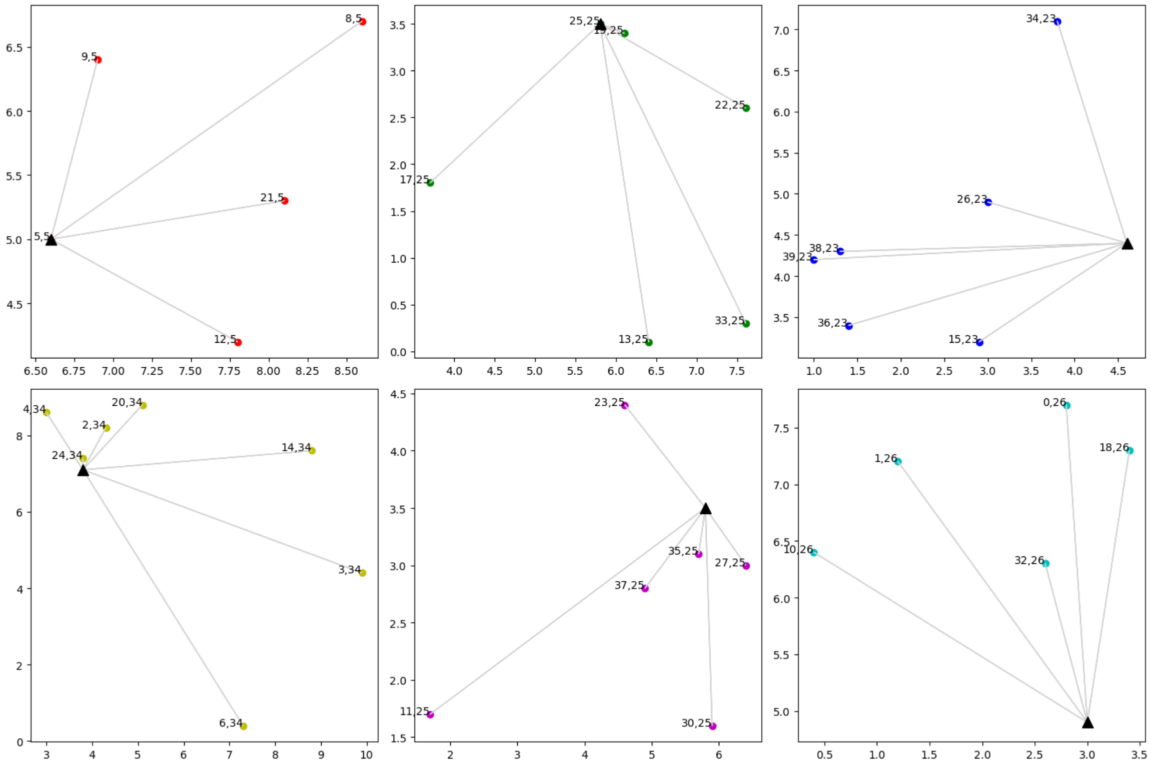


Figure 8: Adjusting Rearrangement Method Example 1 Time Result

This method essentially adjusts the feasible region of the original problem based on the constraints represented by the heuristic method's results. The advantage of this approach is that it prevents a situation we'd rather avoid, as seen in the relaxation method of the previous section 4.3.1, where all customers are removed from the group, which is equivalent to directly solving the original problem. However, at the same time, we cannot guarantee that each adjustment is effective. The results from the next iteration might be worse than those from the current cycle.

4.3.3 Termination Criteria for Iteration

In the iterative process of updating groups and identifying centers (allocation-location), it's imperative to establish a termination criterion to avoid infinite loops. This termination criterion is closely related to the final solution's deviation from the optimal value of the original problem. In this project, we will primarily employ the following methods:

1. By ensuring the gap with the optimal value of the original problem is less than a certain threshold. The calculation method is as follows:

$$\frac{\text{Heuristic Value - Optimal Value}}{\text{Optimal Value}} \times 100\% \tag{4.4}$$

   In principle, during actual problem-solving, we should not adopt this method. The reason we employ heuristic algorithms is to obtain a result that approximates the optimal solution of the original problem while reducing the computation time. Thus, when solving the problem, we do not know the original problem's optimal value, making it impossible to use the gap between the two as a termination criterion. However, during the research phase of the problem, to ascertain the number of iterations needed to reduce the gap to a specific value under ideal conditions, we might choose to incur a higher computational cost to determine the true optimal value of the original problem.

2. When the percentage reduction in the optimization value for the current iteration, compared to the previous iteration, falls below a certain threshold(5%,10%), the iteration process is termi-

nated. The calculation method is as follows:

$$\frac{\text{Heuristic Value (last iteration) - Heuristic Value (this iteration)}}{\text{Heuristic Value (last iteration)}} \times 100\% \tag{4.5}$$

This is a useful method, however, one drawback is the potential risk of getting trapped in a local optimum during the optimization process.

3. In the allocation-location process, we constrain the number of iterations. Once the cumulative count of iterations reaches specified benchmarks (e.g., 5, 10, 15, 20), the loop is terminated.

4. For Relaxing Rearrangement solution, the decision to continue the iteration can be determined by controlling the number of unassigned customers. As previously mentioned, this solution has a drawback in that it might loop until all groups no longer contain any customers. To effectively mitigate this scenario, one approach to control the loop is to halt the process when the number of unassigned customers reaches a certain percentage of the total customer base (e.g., 25%, 50%).

4.3.4 Methods Combination

In practical, we can combine 2 rearrangement methods with different iterative approaches. The fundamental combinations include:

- Relaxing-rearrangement + Iteration 1
- Relaxing-rearrangement + Iteration 2
- Relaxing-rearrangement + Iteration 3
- Relaxing-rearrangement + Iteration 4
- Adjusting-rearrangement + Iteration 1
- Adjusting-rearrangement + Iteration 2
- Adjusting-rearrangement + Iteration 3

The upper combinations will be shown in detail in Section 5.1.

In addition to the aforementioned combinations, we can also explore other combination strategies. For instance, we might initially employ the relaxing rearrangement method for improvement. However, if the number of customers removed exceeds 50% of the total (Iteration 4), we could then deploy the adjusting rearrangement method until the optimization value diminishes by a certain threshold, such as 1% (Iteration 2).

5 Computational Experiments

5.1 Computational Framework and Tools

All the programming and modeling in this paper were executed using **Python**. During the data processing phase, we employed the **os** library for batch file processing, the **Numpy** library for data analysis and preprocessing, and the **timeit** package to record the running time of the model solution. We also utilized the **matplotlib.pyplot** package to visualize the data and the progression of results, facilitating a more intuitive analysis. Additionally, we integrated the **xpress** package, which handles the model's construction and solver components. **Xpress** is a software extensively used for solving operations research problems. While the core principles of the **xpress** library in **Python** are congruent with the **Xpress** software, there may be minor discrepancies in coding syntax.

5.2 Result Analysis for Relaxing-Rearrangement Approach

5.2.1 Relaxing Method with Iteration 1

In this section, we present datasets spanning different weeks and varying numbers of customers. These datasets were subjected to tests using the heuristic solution method. Specifically, we employed the first iteration criterion described in the previous Section 4.3.3, which entails terminating the iteration once the gap between the heuristic result and the optimal value converges to a certain threshold. We examined the required number of iterations and the associated time for convergence thresholds of 2.5%, 5.0%, 7.5%, and 10.0%.

Figure 9 depicts the results for datasets across different weeks (either 6 or 8) and varying customer numbers (either 40 or 50). The customers are differentiated by distinct colors. The figure contrasts the total number of iterations required (represented by the bar chart on the left y-axis) when using the heuristic method of relaxing-rearrangement to achieve results within 2.5% of the optimal value, against the multiplicative factor of time taken in comparison to directly solving for the optimal value (illustrated by the line graph on the right y-axis).

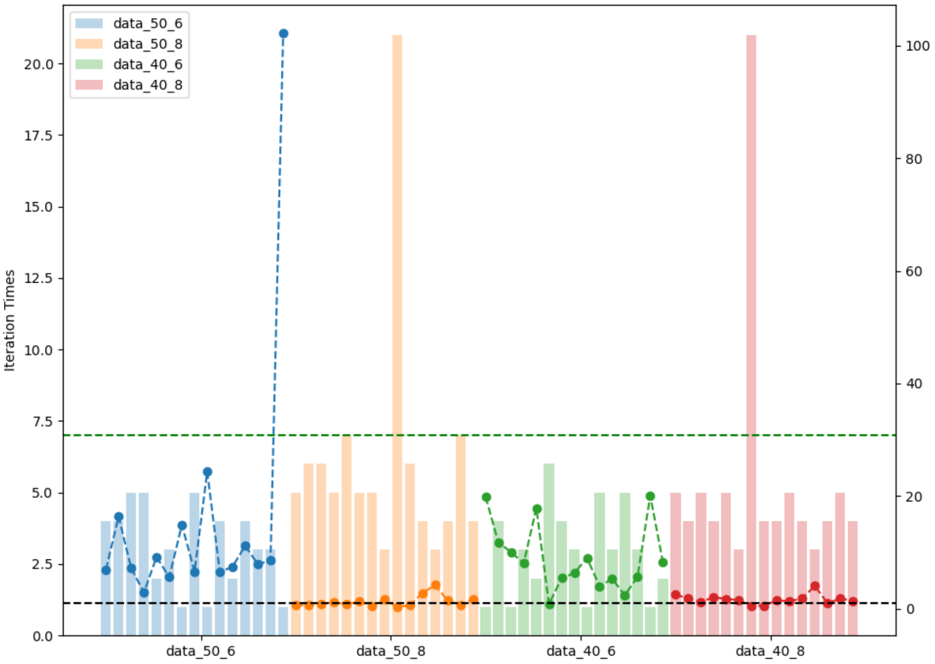


Figure 9: Iteration Criteria: Gap with Optimal Value ($\leq 2.5\%$)

In Figure 9, it is evident that there is a particular dataset for which the multiplicative factor of the reduction time is exceptionally high. Additionally, there are two datasets where the required number of iterations for the customers is unusually numerous. Excluding these outlier datasets, the other data can be observed more clearly in Figure 10. It is apparent that the time savings achieved through the heuristic algorithm has a relatively minor correlation with the number of customers but exhibits a more pronounced relationship with the number of weeks. Most datasets necessitate fewer than 7 iterations. The optimization time is notably longer for 6 weeks, while it's shorter for 8 weeks. Intriguingly, some datasets demand more time than that required for direct optimization.

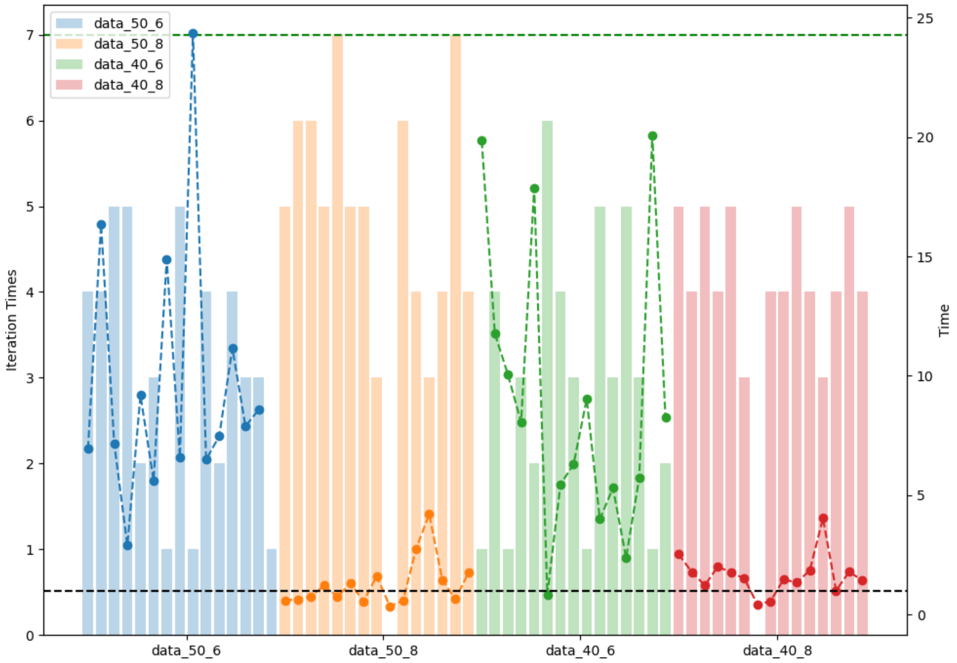


Figure 10: Iteration Criteria: Gap with Optimal Value ($\leq 2.5\%$, without outliers)

Figure 11 presents the data obtained after eliminating the outliers when testing for a gap reduction to 5%. It can be observed that the time consumed by the heuristic approach is predominantly less than that of the direct optimization. Additionally, the number of iterations is generally fewer than 4.

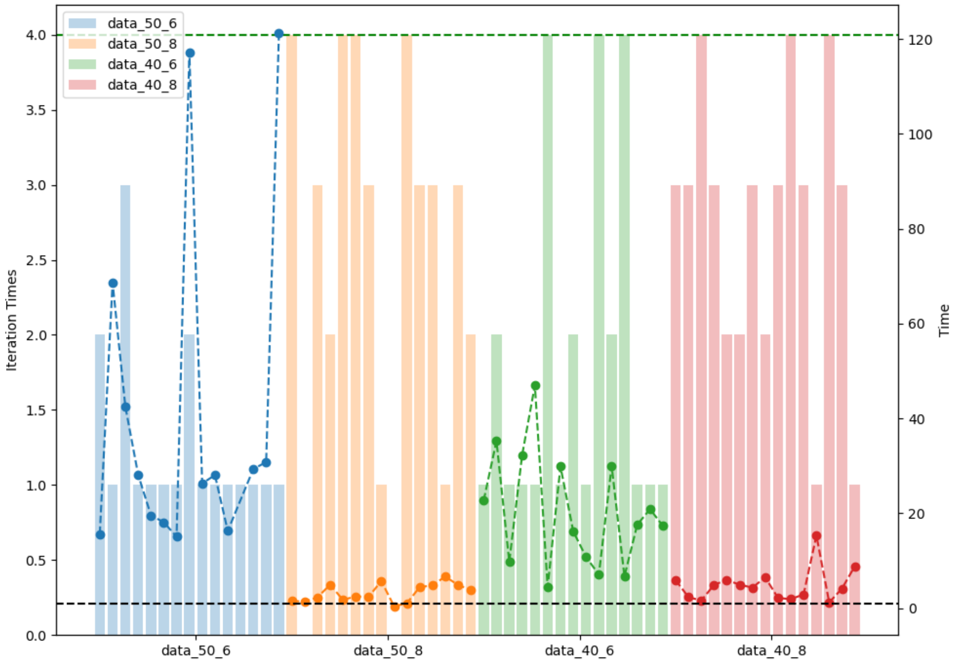


Figure 11: Iteration Criteria: Gap with Optimal Value ($\leq 5.0\%$, without outliers)

Figures 12 and 13 display the data for gap convergence to 7.5% (with outliers removed) and 10%, respectively. It can be observed that for most datasets, achieving a convergence of up to 7.5% requires just 3 iterations, while a convergence to 10% typically requires only 2 iterations. In both scenarios, the time required is consistently less than that for directly determining the optimal value of the problem. Compared to converge to 2.5% and 5%, the discrepancy in optimization time between the 6-week and 8-week decreasing progressively when converge to 7.5% and 10%.

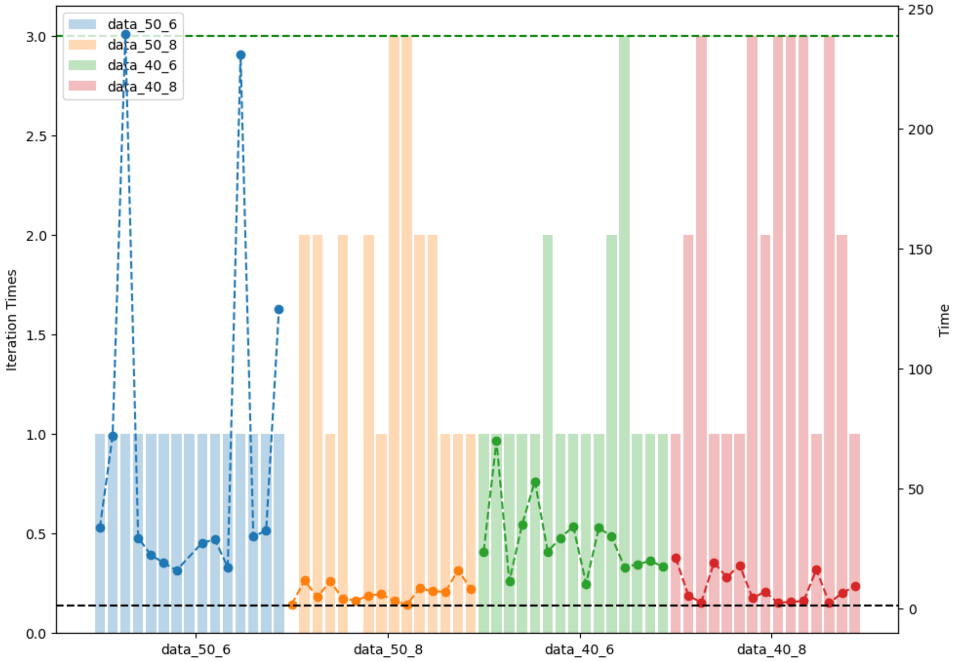


Figure 12: Iteration Criteria: Gap with Optimal Value (≤ 7.5%, without outliers)

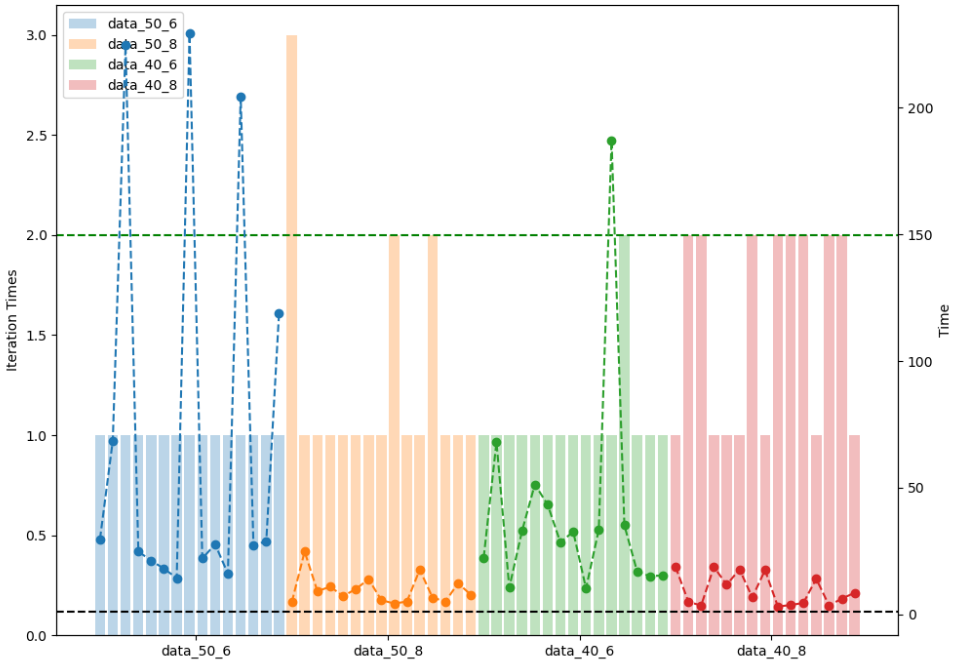


Figure 13: Iteration Criteria: Gap with Optimal Value (≤ 10.0%)

5.2.2 Relaxing Method with Iteration 2

Figures 14 and 15 respectively display the results for various datasets run under the criteria of iteration 2. Iteration 2 denotes the stopping criterion for the loop, where in the iteration ceases once the optimal value in the current loop is less than a predetermined percentage compared to the previous loop. Observably, when the threshold is set at 2.5%, 17 datasets have results within 10% of the optimal value, while the remaining 43 are within 5%. Among these datasets, one required 6 iterations, four needed 4 iterations, and the rest required 2-3 iterations.

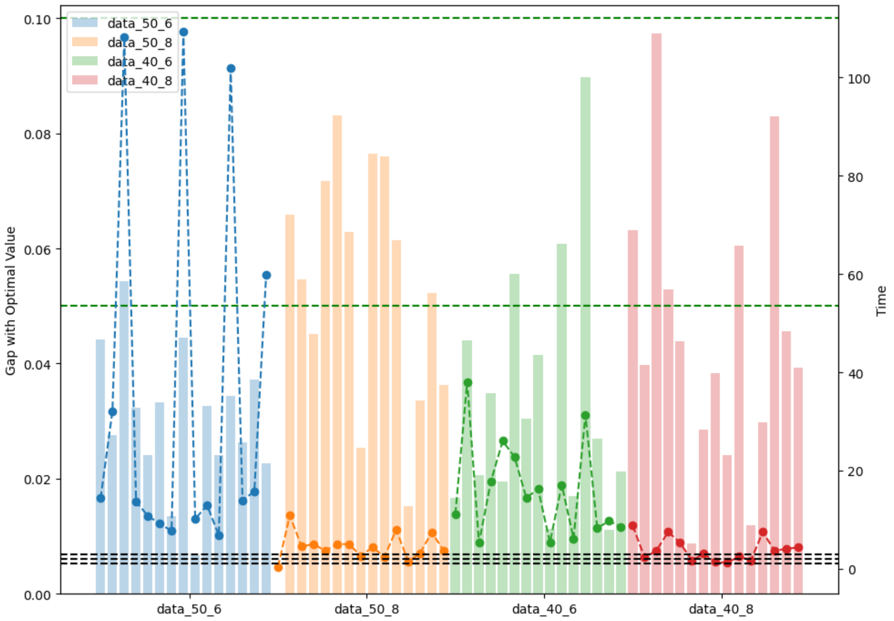


Figure 14: Iteration Criteria: Gap with Last Iteration Value (≤ 2.5%)

When the threshold is set at 7.5%, 23 datasets yield results that are within 10% of the optimal value, while the remaining 37 fall within 5%. For these datasets, the number of iterations consistently ranged from 2-3 times, and the time taken was slightly less than when the criterion was set at 2.5%.

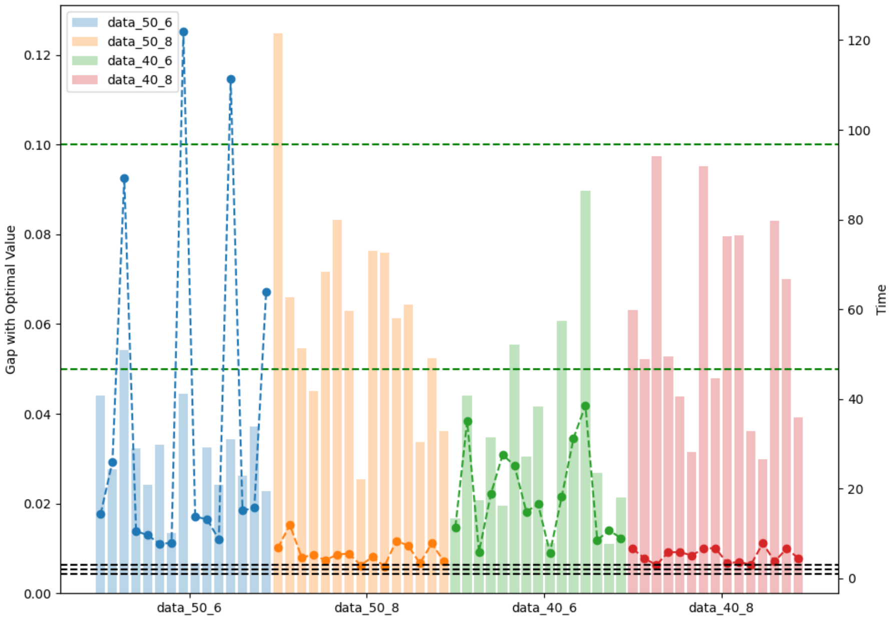


Figure 15: Iteration Criteria: Gap with Last Iteration Value (≤ 7.5%)

5.2.3 Relaxing Method with Iteration 3

Figures 16 and 17 present the scenarios under Relaxing Iteration with 3 and 5 cycles, respectively. As observed in Figure 16, when iterating 3 times, only 2 datasets have a gap of 6% or less with the optimal value. The rest are all within a 5% gap. Moreover, the computation time for almost all datasets is more efficient than the time required to directly determine the optimal solution.

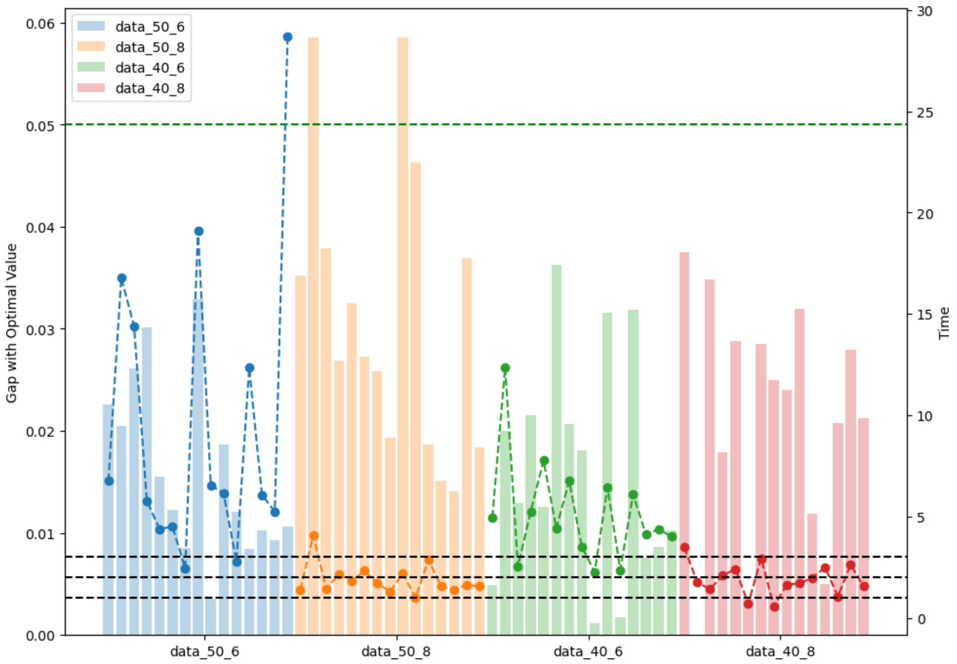


Figure 16: Iteration Criteria: Iteration Times (≤ 3)

In Figure 17, it can be observed that as the number of iterations increases, the gap between the results and the optimal value significantly narrows, with almost all being under 3%. However, concurrently, the required computational time tends to increase linearly. There are even some datasets for which the time taken exceeds that of directly determining the optimal solution.

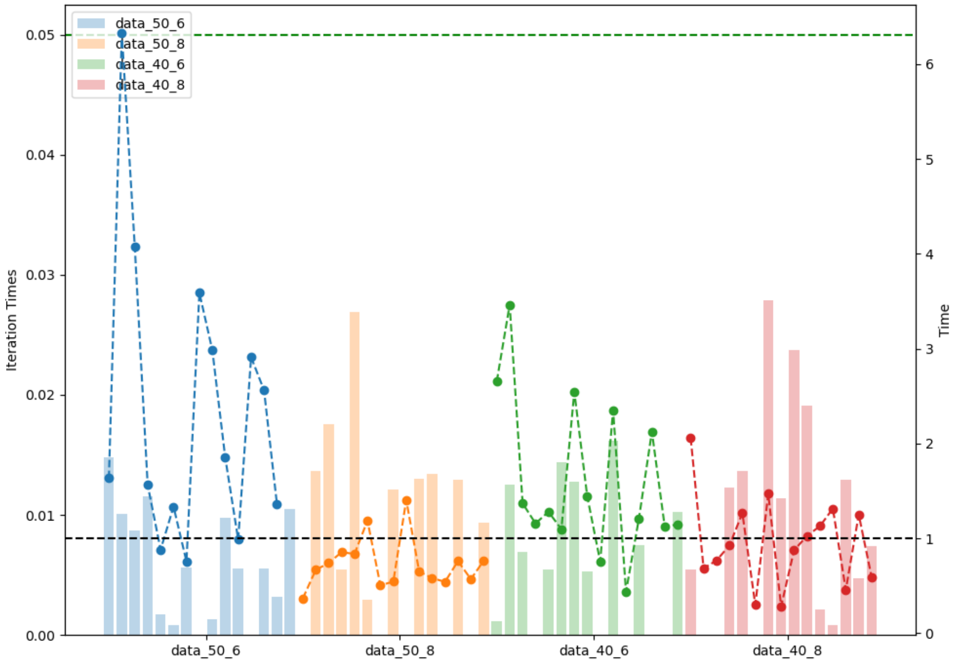


Figure 17: Iteration Criteria: Iteration Times (≤ 5)

5.2.4 Relaxing Method with Iteration 4

Figures 18 and 19 respectively represent the results obtained when employing the relaxing method in conjunction with iteration 4, halting the cycle when the ungrouped customers constitute 10% and 30%

of the total customer base. When the proportion of ungrouped customers is 10%, the gap between the results and the optimal value consistently remains below 10%. Meanwhile, when the proportion is 30%, the gap typically stays below 5%, with a few exceptions hovering around 6%. Under both these conditions, the computational time required is more favorable than that needed for directly calculating the optimal value.

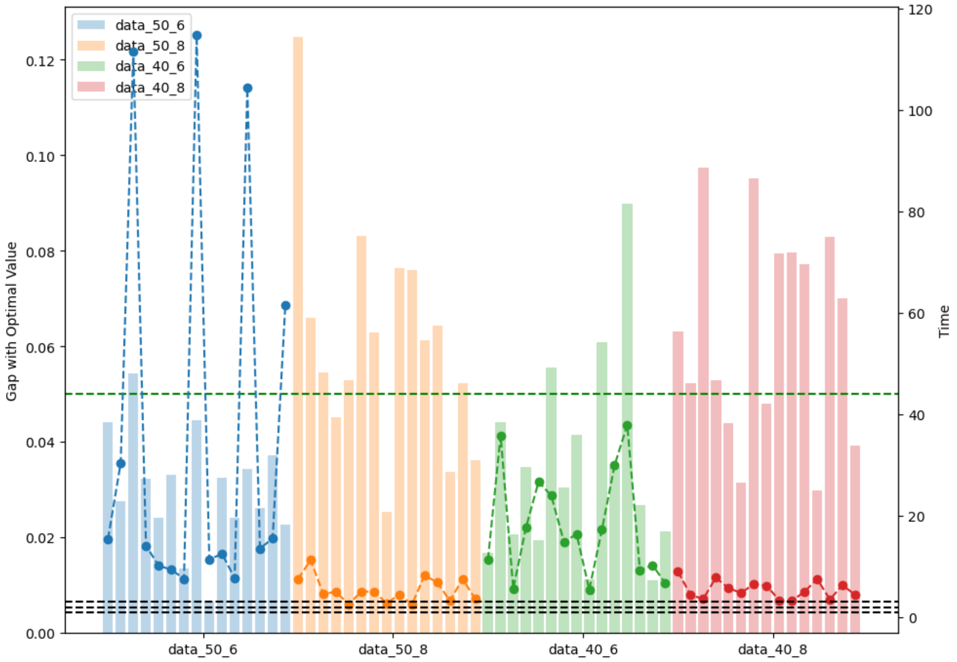


Figure 18: Iteration Criteria: Un-arranging Customers Percentage ($\leq 10\%$)

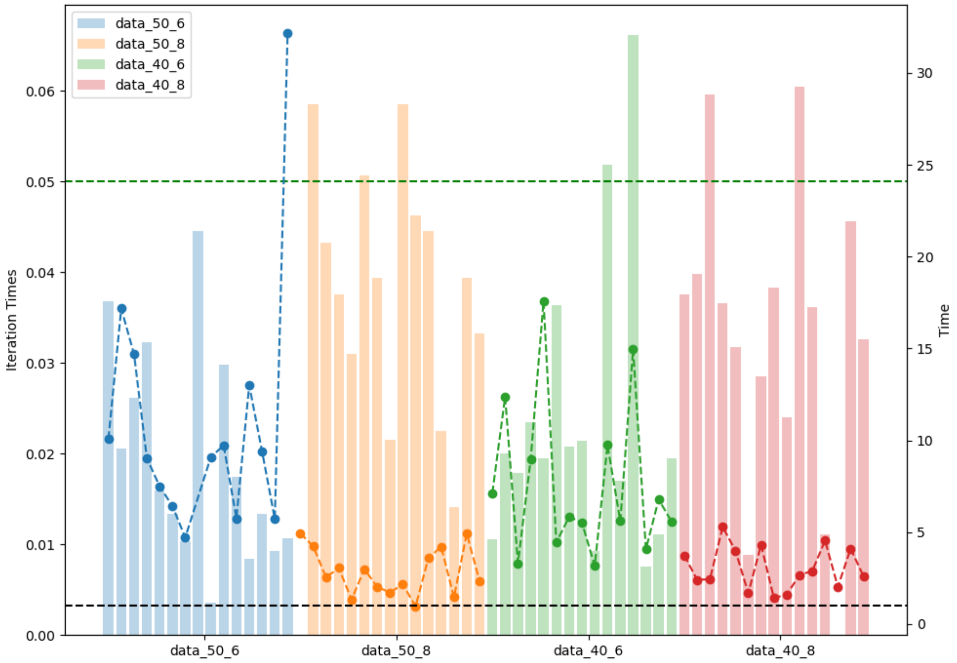


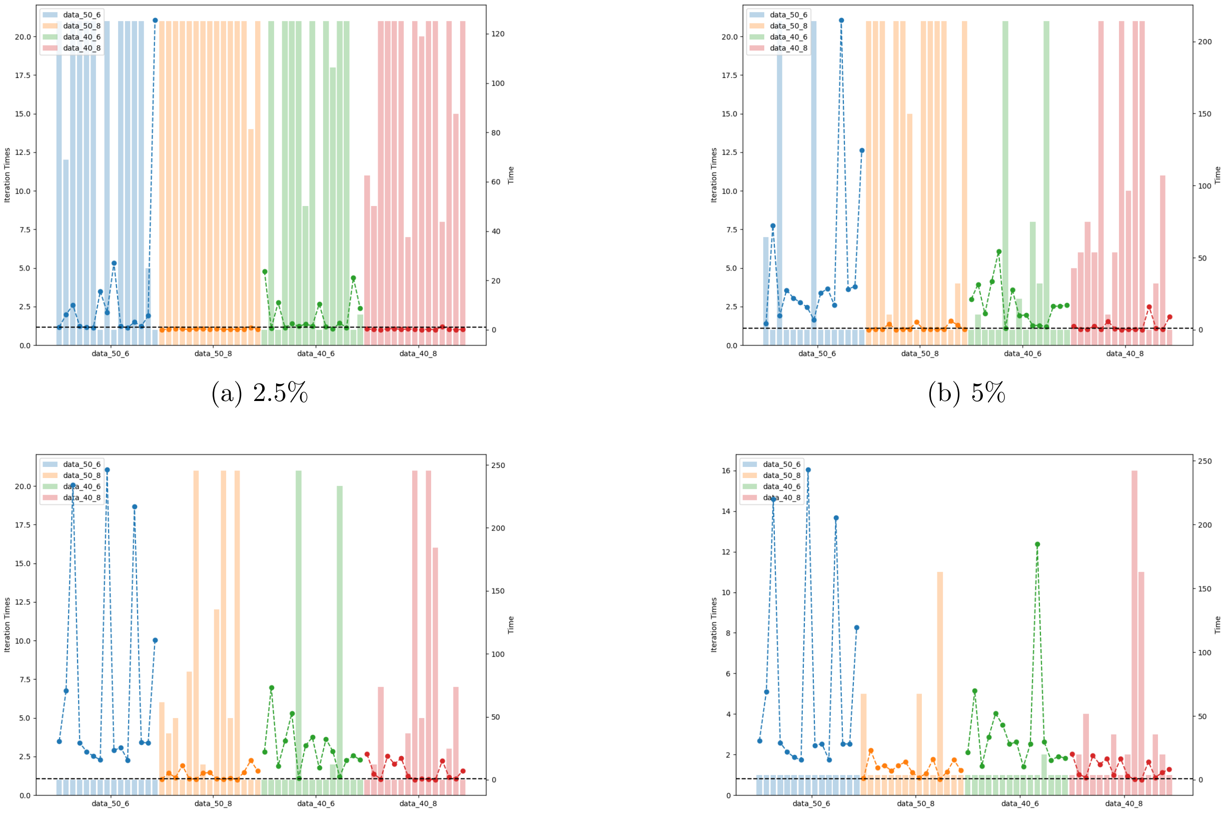
Figure 19: Iteration Criteria: Un-arranging Customers Percentage ($\leq 30\%$)

5.3 Result Analysis for Adjusting-Rearrangement Approach

5.3.1 Adjusting Method with Iteration 1

Figure 20 presents the results of combining the Adjusting-Rearrangement method with iteration 1. The vertical axis on the left represents the number of iterations required to achieve the corresponding gap, depicted using a bar chart. The vertical axis on the right indicates the factor by which the computational time, using the proposed method, is reduced compared to the time required for directly solving the original problem, illustrated with a line graph.

As observed, to achieve a gap of less than 2.5% between the heuristic solution and the optimal value, a significant number of iterations are required for most customers. If the aim is to maintain the gap under 5%, about one-third of the customers need multiple attempts, while the remaining customers necessitate fewer tries. As the gap threshold increases to 7.5% and 10%, the number of customers requiring multiple iterations gradually diminishes.



(a) 2.5%

(b) 5%

(c) 7.5%

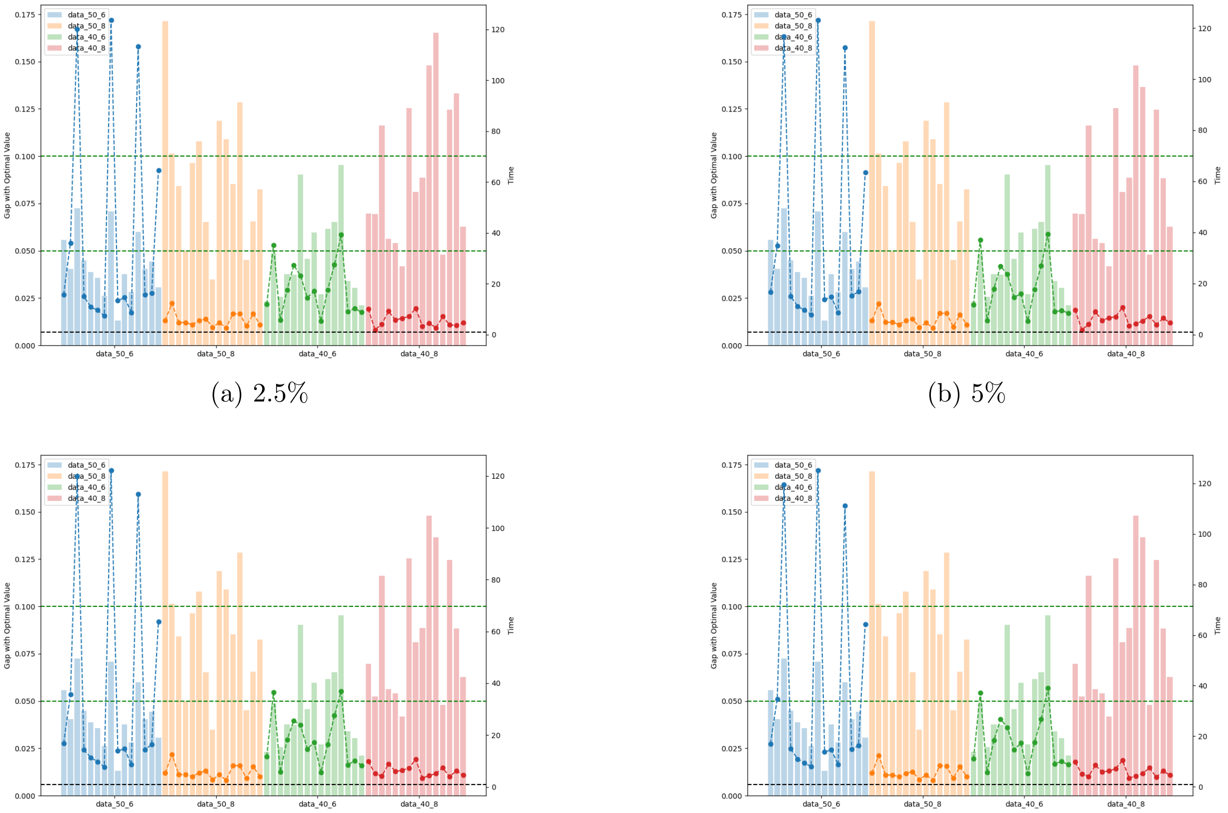
(d) 10%

Figure 20: Adjusting with Iteration 1

5.3.2 Adjusting Method with Iteration 2

Figure 21 displays the results obtained when combining the Adjusting method with iteration 2. The vertical axis on the left depicts the gap between the results of this method and the optimal value of the original problem, represented using a bar chart. On the right vertical axis, a line graph illustrates the factor by which the computational time, with the proposed method, is reduced compared to the time needed for directly solving the original problem.

It can be observed that irrespective of the optimization values from one iteration to the next (whether 2.5%, 5%, 7.5%, or 10%), the gap between the heuristic results and the optimal value shows minimal variation in terms of computational time.



(a) 2.5%

(b) 5%

(c) 7.5%

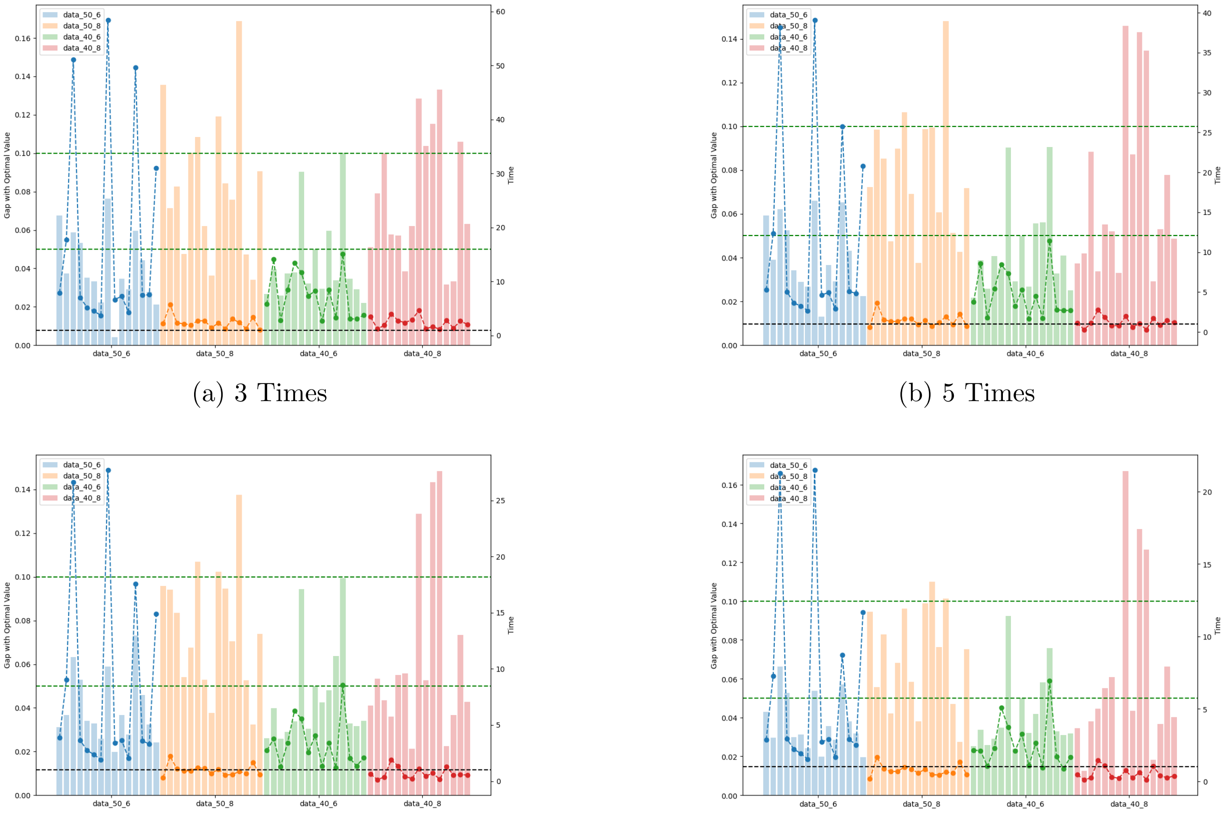
(d) 10%

Figure 21: Adjusting with Iteration 2

5.3.3 Adjusting Method with Iteration 3

Figure 22 displays the results obtained when combining the Adjusting method with iteration 2. The vertical axis on the left depicts the gap between the results of this method and the optimal value of the original problem, represented using a bar chart. On the right vertical axis, a line graph illustrates the factor by which the computational time, with the proposed method, is reduced compared to the time needed for directly solving the original problem.

From Figure 22, it can be observed that as the number of iterations increases, the gap between the heuristic algorithm's results and the optimal value of the original problem gradually decreases. However, this method seems not able to reduce the gap to below 5% for all datasets. Concurrently, the computational time required increases with the additional iterations.



(a) 3 Times

(b) 5 Times

(c) 7 Times

(d) 9 Times

Figure 22: Adjusting with Iteration 3

5.4 Analysis

From the analysis of the above results, it is evident that Relaxing-Rearrangement method performs well in most scenarios. This performance is good shows both in its ability to control the gap relative to the original optimal value and in reducing the computation time. When combined with iteration 1, the results indicate that, in general, only four iterations are needed to achieve a gap of less than 5% from the optimal value, with consistent improvements in time. Under this iterative condition, the computation time for most cases is 5 to 40 times faster than directly solving for the optimal solution, as illustrated in Figure 23.

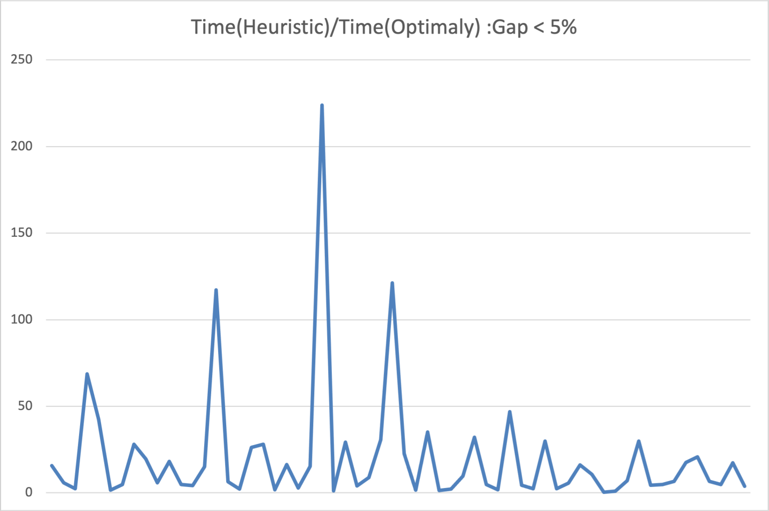


Figure 23: Timing Times Comparing with Optimaly Solving Problem(Gap 5%,relaxing method)

From the results of relaxing method with iteration 3, it is evident that by limiting the number of cycles to three, we can essentially ensure that the gap remains below 5%. In the findings from iteration 4, when the proportion of un-grouped entities is controlled at a 30%, it becomes feasible to guarantee that the gap is less than 5%.

Compared to the Relaxing method, the Adjusting method demonstrates inferior performance. As observed from the results under iteration 1, depending on the data structure during the iterative process, to reduce the gap between the result and the optimal value to a certain threshold, one either achieves this in the first iteration or multiple attempts still fall short. Moreover, in iteration 2, utilizing the convergence value of the gap from two consecutive iterations as the termination criterion appears to exert minimal influence on the final outcome.

This suggests that the Adjusting method's modification of the original problem's constraints is not as effective as the direct relaxation approach offered by the Relaxing method. We conjecture that, during the actual solution process, when dealing with datasets of a scale comparable to this project, one might consider solely employing the relaxing method while effectively controlling the number of iterations. However, when faced with larger and more complex datasets, the adjusting method could potentially be integrated with the relaxing approach. For instance, in scenarios where the relaxing method results in an excessive number of unassigned customers, one could deploy the adjusting method for reallocation.

6 Conclusion

The Multi-Period Service Territory Design Problem, abbreviated as MPSTDP, could be primarily divided into two distinct categories: MPSTDP-Partition and MPSTDP-Schedule. In this study, we have adapted the model proposed by Bender et al.[2] for MPSTDP-S, tailoring it to better suit the operations of a company which routinely services 40-50 clients over a span of 6-8 weeks. This model seeks a balance between the cost implications of routing schedules and their flexibility, choosing to use compactness as the primary metric to be optimized (minimized) within the objective function. To accommodate specific servicing intervals for clients, such as visits every 3 weeks or every 5 weeks, a week pattern is employed to determine the exact week of service for each client. The model consists of both weekly and daily components, both of which bear significant resemblance. Based on the data at our disposal, this study predominantly focuses on the weekly component.

To address the solution of this model, we have devised a novel heuristic algorithm, which comprises both initialization and improvement phases. During the initialization phase, efforts were made to equitably distribute the workload across different groups while minimizing the distance amongst clients within each group. In the improvement phase, we introduced two strategies. The first, termed as ”relaxing-rearrangement”, entails a relaxation of the constraints added to the original problem by the heuristic during each iteration. The second strategy, ”adjusting-rearrangement”, modifies the constraints added by the heuristic without the direct relaxation characteristic of the former method. By integrating iterative standards from sections 3-4, and setting four thresholds under each standard for investigation, we observed that, given our dataset (50-40 clients, spanning 6-8 weeks), the relaxing approach outperforms the adjusting method. The relaxing technique not only reduces the time required to solve the problem compared to non-heuristic methods but also ensures that the gap between the final result and the true optimal value remains below 5%. On the other hand, the adjusting method yielded sub-optimal final results, however, its computation time during individual iterations was relatively short.

In future research, we propose exploring the following three directions. First,Consider larger datasets and combine both improvement methods to investigate their cumulative effect on the final results. Second, Given sufficient data, we can also extend our heuristic algorithm to study the ”Day” component of the problem. Third,We can further incorporate the Partition aspect of the MPSTDP problem, as in practical scenarios, a particular region is typically managed by multiple salesmen over a certain time period.

References

[1] M. Bender, J. Kalcsics, and A. Meyer. Districting for parcel delivery services–a two-stage solution approach and a real-world case study. *Omega*, 96:102283, 2020.

[2] M. Bender, A. Meyer, J. Kalcsics, and S. Nickel. The multi-period service territory design problem–an introduction, a model and a heuristic approach. *Transportation Research Part E: Logistics and Transportation Review*, 96:135–157, 2016.

[3] F. Blakeley, B. Argüello, B. Cao, W. Hall, and J. Knolmajer. Optimizing periodic maintenance operations for schindler elevator corporation. *Interfaces*, 33(1):67–79, 2003.

[4] A. Diglio, S. Nickel, and F. Saldanha-da Gama. Towards a stochastic programming modeling framework for districting. *Annals of Operations Research*, 292(1):249–285, 2020.

[5] B. Fleischmann and J. N. Paraschis. Solving a large scale districting problem: a case report. *Computers & Operations Research*, 15(6):521–533, 1988.

[6] J. Kalcsics. *Districting Problems*, pages 595–622. Springer International Publishing, Cham, 2015.

[7] H. Lei, G. Laporte, Y. Liu, and T. Zhang. Dynamic design of sales territories. *Computers & Operations Research*, 56:84–92, 2015.

[8] R. Z. Rios-Mercado and H. J. Escalante. Grasp with path relinking for commercial districting. *Expert Systems with Applications*, 44:102–113, 2016.

[9] R. Z. Ríos-Mercado and E. Fernández. A reactive grasp for a commercial territory design problem with multiple balancing requirements. *Computers & Operations Research*, 36(3):755–776, 2009.

[10] R. Z. Ríos-Mercado and J. F. López-Pérez. Commercial territory design planning with realignment and disjoint assignment requirements. *Omega*, 41(3):525–535, 2013.

[11] M. G. Sandoval, J. A. Díaz, and R. Z. Ríos-Mercado. An improved exact algorithm for a territory design problem with p-center-based dispersion minimization. *Expert Systems with Applications*, 146:113150, 2020.

[12] L. Zhou, L. Zhen, R. Baldacci, M. Boschetti, Y. Dai, and A. Lim. A heuristic algorithm for solving a large-scale real-world territory design problem. *Omega*, 103:102442, 2021.