Queueing Models and
Service Management
Vol. 8, No. 4, page 75-106, 2025

QMSM
©AU 2023

A Case Study on Optimizing Call Center Daily Staff Scheduling Using the Set Covering Model

Xiaodong Liu[1*], Yu Song[2], Minoru Kobayashi[2] and Hanlin Liu[1]

[1] Graduate School of Engineering
Fukuoka Institute of Technology, Japan

[2] Department of Information Management
Fukuoka Institute of Technology, Japan

(*Received August 2025; accepted November 2025*)

Abstract: Efficient staff scheduling in call centers improves operational efficiency, reduces costs, and ensures sufficient customer service. This study addresses the daily shift scheduling problem in a call center with multiple types of tasks. While the set covering model is widely used to derive optimal solutions for such problems, its application requires enumerating all possible combinations of timeslots and tasks. This process significantly increases the complexity of the model, leading to computational difficulties. Additionally, the call center considered in this study permits staff to perform two different types of tasks simultaneously within the same timeslot, which are referred to as pair tasks. This further exacerbates computational challenges, as it significantly increases the number of task pattern combinations, making the model even more difficult to solve.

To address the challenge of exponential growth in combinations, a heuristic algorithm is proposed that not only supports pair tasks within the same timeslot but also significantly reduces the number of task pattern combinations. The algorithm is first applied to the base model (Model 1) to obtain an initial near-optimal solution; then task patterns are refined to further improve solution quality. Subsequently, we augment Model 1 by incorporating real-world operational conditions, yielding Model 2 that better reflects practical requirements. Numerical experiments demonstrate the effectiveness and practicality of the proposed algorithm and models.

Keywords: Call center staff scheduling, daily shift, pair tasks, set covering problem.

1. Introduction

Staff scheduling is a critical component of business operations, directly influencing organizational efficiency, staff satisfaction, and overall productivity. Effective scheduling ensures optimal resource allocation, improves staff well-being, and reduces operational

* Corresponding author
Email: bd24102@bene.fit.ac.jp

costs. This problem arises across various sectors, including call centers, manufacturing, banking, and healthcare, underscoring its universal significance.

This study focuses on solving the daily shift scheduling problem in a call center with multiple types of tasks. Conducted as a case study, it investigates a real-world scheduling scenario with practical constraints. A key feature is the consideration of staff members who are permitted to perform two types of distinct tasks simultaneously within a given timeslot, provided they have skills in both tasks (hereafter, the term “tasks” is used to represent “types of tasks” for brevity). In the following discussion, we refer to this situation as "pair tasks".

In call center operations, pair tasks generally arise in two distinct forms. The first situation occurs when an agent performs multiple tasks within the same company. For instance, an agent may primarily handle inbound calls, but during the same timeslot, occasionally process customer emails during idle intervals. The second situation may arise in outsourced settings, where an agent may be assigned tasks for two different companies simultaneously, handling tasks for Company A while also attending tasks for Company B. These operational patterns motivate the pair task setting, defined as sharing a single timeslot between two tasks under a fixed effort split, which can improve agent utilization while maintaining service levels.

Extensive research has been conducted on staff scheduling problems, exploring various methodologies and applications. Ernst et al. [8] provided a comprehensive review of staff scheduling challenges, highlighting models and algorithms developed for specific application areas. Their work also introduced general approaches for addressing call center scheduling problems.

As research in this field has progressed, different aspects of staff scheduling have been explored. Broadly, these studies can be categorized into three stages based on their primary focus: staff service level prediction, multi-day scheduling, and daily shift scheduling.

The first stage [1][7][19] focuses on staff service level prediction, which estimates the required number of staff based on historical data, call volume trends, and so on. Accurate predictions of staff service levels are essential for ensuring operational efficiency, minimizing labor costs, and maintaining service quality in call centers. Moreover, Su et al. [20] proposed a queueing model for determining staffing levels in inpatient units with multi-type patients, representing a related line of research on capacity planning under heterogeneous service demand. However, these studies primarily address staffing level determination rather than the construction of daily shift schedules.

The second stage [4][6][18] focuses on research related to multi-day scheduling. The essence of multi-day scheduling lies in determining staff working and non-working days over a planning horizon while establishing shift schedules for each designated workday, without specifying individual task assignments. This stage seeks to optimize workforce allocation across multiple days to ensure adequate staffing levels, enhance operational efficiency, and support staff well-being. Well-structured multi-day scheduling contributes to workload balance, mitigates excessive overtime, and fosters long-term workforce stability.

The third stage explores various approaches to daily shift scheduling. Daily shift scheduling involves determining staff’s specific working hours, break times, and task assignments within a single workday. It ensures efficient task allocation, minimizes understaffing, and improves service quality by optimizing intra-day workforce deployment. This study falls under this category.

Although studies such as [10][22][23] address daily scheduling problems, they focus on domains other than call centers, and thus differ from the objectives of this paper. Thompson [21] utilized simulated annealing to address staff scheduling problems. Mason et al. [16] adopted an integrated approach combining simulation, heuristics, and optimization. Fukunaga et al. [9] employed artificial intelligence search methods to generate schedules under various constraints, and Avramidis et al. [3] applied simulation algorithms to daily shift scheduling in call centers. However, the methods proposed in these studies are not directly applicable to the problem explored in this paper, as none of them considers the challenge of assigning pair tasks to staff members.

To the best of our knowledge, there is currently no method in the literature that addresses the assignment of pair tasks within the same timeslot. Therefore, the pair task mechanism introduced in this study represents a novel concept, and no direct benchmark methods are available for comparison.

This study adopts the set covering model to address the daily shift scheduling problem in the call center. Originally proposed by Dantzig [5], the set covering model is among the most widely used approaches for solving staff scheduling problems and employs a Binary Integer Programming (BIP) framework to derive optimal shift allocations. However, the application of a typical set covering model requires the enumeration of all possible task patterns. The term “task pattern” refers to a predefined combination of tasks to be performed in each timeslot (see Section 4 for details). The number of task patterns increases exponentially with the number of tasks (k) and timeslots (l), following a complexity of k^l. For instance, with 10 tasks and 5 timeslots, the number of possible task patterns amounts to 10^5. This combinatorial explosion renders the direct application of the model computationally challenging. The associated modeling and computational burdens make it practically infeasible to solve on standard computing resources, even without considering more complex pair task scenarios.

To mitigate this problem, some previous studies have proposed alternative approaches [11] [15]. For example, Lavoie et al. [11] reduced the generation of task patterns using column generation methods, effectively addressing the computational complexity of task pattern enumeration.

Additionally, there are approaches that do not rely on set covering models and thus do not require task pattern generation. For instance, Aykin [2] and Liu et al. [13][14] proposed an algorithm that constructs schedules without generating task patterns. Li et al. [12] addressed this problem by applying quantum computing approaches. However, none of the above methods considers scenarios in which staff members can perform pair tasks.

This study proposes a heuristic algorithm that can greatly reduce the number of task patterns and generate pair task patterns. The task patterns generated by the heuristic algorithm are typical patterns commonly used in this call center. To verify the effectiveness of this algorithm, solutions are first obtained by applying the generated task patterns to Model 1 (the basic set covering model) through numerical experiments. These solutions are then compared with the results of a linear model which provides a theoretical lower bound to evaluate the performance of the algorithm. Although we attempted to compare the proposed heuristic algorithm with other state-of-the-art approaches, existing methods are not directly applicable to the pair task setting, which further highlights the novelty of our approach. Then, to make the results closer to those of the linear model, the task patterns are further adjusted. Finally, to better reflect real-world operational requirements, Model 2 is developed, and its practicality is verified through numerical experiments.

This paper is organized as follows:

Section 2 explains the assumptions and problem. Section 3 introduces the Model 1 to solve the daily shift problem in the call center. In Section 4, we propose a heuristic algorithm to reduce the number of task patterns and demonstrate the effectiveness of this algorithm through numerical experiments. Section 5 focuses on improving task patterns. Section 6 introduces Model 2, which is designed to better reflect real-world conditions. Numerical experiments are then conducted to verify the feasibility of this model. Finally, Section 7 concludes the study and outlines potential directions for future research.

2. Assumptions and Problem Description

This section describes the assumptions and outlines the problem addressed in this study. The assumptions aim to simplify the modeling and computation process, while the problem description provides a detailed explanation of the daily shift scheduling challenge in the multi-task call center. Specifically, the study focuses on assigning tasks to staff based on their proficiencies, while incorporating constraints such as working hours, break times, and task demands.

2.1 Assumptions

To effectively address the daily shift scheduling problem in the multi-task call center, several assumptions (Table 1) are made to simplify models and ensure computational feasibility.

A critical component of the scheduling model is the definition of timeslots (TS). The task pattern consists of multiple consecutive timeslots, and task assignments are made within these intervals. In the call center, timeslots are in practice set at 15-minute intervals. However, to reduce computational complexity, each timeslot is defined as a one-hour interval in this study (Assumption 1).

Break scheduling must comply with the provisions of the Labour Standards Act [17] of Japan, which mandates a minimum break of 45 minutes when the total working time exceeds 300 minutes. In practice, short breaks of approximately 15 minutes at regular timeslots are also commonly implemented. However, to simplify the modeling process, the following assumptions are made: Assumption 2 sets a break duration of one hour, and Assumption 3 restricts the break period to either TS12 (12:00–13:00) or TS13 (13:00–14:00).

To enhance flexibility and better reflect real-world scenarios, Model 2 (Section 6) relaxes Assumption 3. This adjustment provides greater adaptability, enabling more practical and realistic break arrangements to meet operational demands.

Table 1. Assumptions in This Study

No.	Assumption	Sections
1	Each timeslot is one hour	Section 3-6
2	The break duration is one timeslot	Section 3-6
3	Breaks at TS12 or TS13	Section 3-5

2.2 Problem description

This section provides a detailed description of the call center. In the center, staff members are required to handle various tasks, each with distinct skill requirements. Staff members possess varying levels of proficiency for different tasks, and the scheduling process aims to meet task demands as much as possible while minimizing understaffing when full coverage cannot be achieved.

In this problem, as described in Section 1, the outputs from the first stage (staff service level prediction, including staff proficiency and task demands) and the second stage (multi-day scheduling, including monthly shift and shift pattern) are given as input. Accordingly, this study focuses on optimizing task allocation within the given shift assignments, aiming to minimize understaffing.

Table 2 is an example of task demands. For instance, task 1 has no demand during TS8, whereas in all other timeslots, the demand is 2 person · hours.

Table 2. Example of Task Demands

Task	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
Task 1	0	2	2	2	2	2	2	2	2	2	2
Task 2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Task 3	1	1	1	1	1	1	1	1	1	1	0
Task 4	0	0	1	0	0	0	0	0	0	0	0

(Unit: person · hour)

Table 3 provides an example of task proficiency levels. For instance, Staff 1 has proficiency levels of 2, 1, 0, and 0.5 for the four tasks, respectively. A proficiency level of 1 indicates that the staff member is capable of handling 1 person · hour of task 2, which represents an average level of proficiency. A level of 2 for task 1 indicates proficiency at twice the average level, while a level of 0.5 for task 4 reflects half the average level. Proficiency level of 0 for task 3 means the staff member is not qualified to perform it.

Figure 1 shows the example of shift patterns. The color-coding highlights the working status: green cells denote idle timeslots, and orange cells indicate active assigned timeslots. For example, Shift ID 0 is completely idle, meaning it does not work throughout the day, while Shift ID 1 is active from TS8 to TS16.

Table 3. Example of Proficiency Levels

Staff ID	Task			
	Task 1	Task 2	Task 3	Task 4
1	2	1	0	0.5
2	2	0	1	0
3	0	0	1	1
4	2	0	0	0
5	1	0	1	0

Shift ID	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
0	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩
1	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩
2	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩	🟩	🟩
3	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩	🟩
4	🟩	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
5	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩
6	🟩	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩
7	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩
8	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩	🟩	🟩
9	🟩	🟩	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧
10	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩	🟩
11	🟩	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩	🟩

Figure 1. Example of Shift Pattern

Table 4 presents the example of a part of the monthly shift. For instance, Staff 1 is assigned Shift ID 1 on Day 1, which corresponds to the row labeled Shift ID 1 (active during TS8–TS16) in Figure 1. Similarly, Staff 2 is assigned Shift ID 8 on Day 1, corresponding to the row labeled Shift ID 8 (active during TS9–TS14).

Table 4. Example of a Part of the Monthly Shift

Staff ID	Timeslot						
	Day 1	Day 2	Day 3	Day 4	Day 5	...	Day31
1	1	2	2	0	0	...	2
2	8	0	11	11	0	...	8
3	8	8	8	0	0	...	8
4	8	8	1	0	0	...	2
5	0	1	9	1	0	...	9

The objective of this study is to generate daily shift schedules as illustrated in Table 5. In this table, each cell indicates the assigned task, break timeslot (denoted as "break"), or idle timeslot (marked as "0"). For example, staff 1 is assigned to work on TK1 (task 1) during TS9–TS15, takes a break during TS12, and has no assignment during TS16–TS18. Staff 4 is assigned to TK4 during TS8–TS14, except that the task in TS10 is replaced by TK3. The staff member takes a break during TS13 and has no assignment during TS15–TS18. Staff 5 is assigned to a pair task, working on TK1 and TK2 simultaneously during TS8–TS15.

Table 5. Example of Daily Staff Scheduling

Staff ID	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
1	0	TK1	TK1	TK1	break	TK1	TK1	TK1	0	0	0
2	0	0	TK1	TK1	TK1	break	TK1	TK1	TK1	0	0
3	0	0	TK3	TK3	break	TK3	TK3	TK3	0	0	0
4	TK4	TK4	TK3	TK4	TK4	break	TK4	0	0	0	0
5	TK1 TK2	TK1 TK2	TK1 TK2	TK1 TK2	break	TK1 TK2	TK1 TK2	TK1 TK2	0	0	0

3. Model 1 Formulation

The problem is modeled using the set covering approach [5] and formulated as an MIP model described below.

Symbols

I: number of staff, $i = 1, 2, 3, \ldots, I$
J: number of task patterns, $j = 1, 2, 3, \ldots, J$
K: number of tasks, $k = 1, 2, 3, \ldots, K$
L: number of timeslots, $l = 8, 9, \ldots, L + 7$
b_{kl} : demand of task k at timeslot l (as shown in Table 2)
p_{ik} : proficiency of staff i for task k (as shown in Table 3)
s_i: shift ID of staff i (as shown in Table 4)

$$a_{ls} = \begin{cases} 1, & \text{if timeslot } l \text{ is in the working period of shift } s (orange\ cells\ in \text{ Figure 1}) \\ 0, & \text{otherwise} \quad (\text{green } cells\ in \text{ Figure 1}) \end{cases}$$

$$e_{jkl} = \begin{cases} 1, & \text{when task } k \text{ is assigned alone (not pair) in timeslot } l \text{ of pattern } j \\ 0.5, & \text{when pattern } j \text{ assigns task } k \text{ as part of a pair task in timeslot } l \\ 0, & \text{otherwise} \end{cases}$$

w_k : penalty per timeslot for understaffing of task *k*

The e in e_{jkl} represents effort. When $e_{jkl} = 1$, task *k* in task pattern *j* during timeslots *l* is allocated the full effort alone; when $e_{jkl} = 0.5$, task *k* can only be assigned half of the effort.

Decision Variables

$$x_{ij} = \begin{cases} 1, & \text{assigning pattern } j \text{ to staff } i \\ 0, & \text{otherwise} \end{cases}$$

z_{kl}: amount of understaffing for task *k* at timeslot *l*

Model 1

Model 1 is the basic model for the daily shift problem.

Objective Function

$$\text{minimize } \sum_{k=1}^{K} \sum_{l=8}^{L+7} w_k z_{kl} \tag{1}$$

Subject to

$$\sum_{j=1}^{J} x_{ij} = 1 \quad (\forall i \in \{1,2,\dots,I\}, \text{with } s_i > 0) \tag{2}$$

$$\sum_{i=1}^{I} \sum_{j=1}^{J} x_{ij}\, a_{ls_i} p_{ik} e_{jkl} + z_{kl} \geq b_{kl} \quad (\forall k \in \{1,2,\dots,K\}, \forall l \in \{8,9,\dots,L+7\}) \tag{3}$$

$$x_{ij} = \{0,1\} \quad (\forall i \in \{1,2,\dots,I\}, \forall j \in \{1,2,\dots,J\}) \tag{4}$$

$$z_{kl} \geq 0 \quad (\forall k \in \{1,2,\dots,K\}, \forall l \in \{8,9,\dots,L+7\}) \tag{5}$$

Equation (1) minimizes the total understaffing across all timeslots and tasks. Equation (2) ensures that each staff member is assigned exactly one task pattern while working. Equation (3) defines the relationship between processing capacity and demand. This equation links staff processing capacity at a specific timeslot and task pattern to the corresponding demand. The first term on the left-hand side represents the staff's processing capability for task *k* at timeslot *l*, which is determined by the staff's proficiency (p_{ik}) and the effort (e_{jkl}). The second term z_{kl} denotes understaffing. When the processing capacity meets the demand, $z_{kl} = 0$; if the processing capacity is insufficient, understaffing occurs, causing $z_{kl} > 0$.

For example, assume that $x_{ij} = 1$ and $a_{ls_i} = 1$. Suppose in timeslot *l*, the demands for task 1 and task 2 are $b_{1l} = 2$ and $b_{2l} = 0.5$, respectively. Consider staff *i* with proficiencies $p_{i1} = 2$ for task 1 and $p_{i2} = 1$ for task 2. If this staff is assigned only to task 1, at this point $e_{j1l} = 1$ and $e_{j2l} = 0$, the demand for task 1 is fully satisfied because $p_{i1} \times e_{j1l} = 2$, while the remaining demand for task 2 becomes $z_{2l} = 0.5$.

In the other case, if the staff simultaneously performs task 1 and task 2 as a pair task in the same timeslot, the effort is divided equally with $e_{j1l} = e_{j2l} = 0.5$. The staff member contributes $p_{i1} \times e_{j1l} = 1$ unit of capacity to task 1 and $p_{i2} \times e_{j2l} = 0.5$ units to task 2. Consequently, the residual demands are $z_{1l} = b_{1l} - p_{i1} \times e_{j1l} = 1$ and $z_{2l} = b_{2l} - p_{i2} \times e_{j2l} = 0$.

4. Algorithm for Task Pattern Generation

Task patterns play a critical role in solving multi-task call center problems using the set covering model. However, obtaining an optimal solution requires the exhaustive enumeration of all possible task pattern combinations, which makes the computation extremely complex and impractical. Since the number of task patterns grows exponentially with the increase in timeslots and tasks, we make certain trade-offs. Specifically, to ensure that the obtained approximate solution meets our expectations (though not necessarily optimal), we propose a heuristic algorithm to significantly reduce task patterns.

The proposed task pattern generation method considers both staff skill levels and task demand. Specifically, we selected three types of task patterns that demonstrated higher efficiency based on these factors, without considering other possible patterns, thereby significantly reducing the computational complexity. This approach guarantees that the generated task patterns are feasible for practical scheduling applications.

Algorithm 1 presents a heuristic algorithm for generating task patterns, which significantly reduces the number of patterns. Algorithm 2 provides a further improvement of the task patterns.

4.1. Algorithm 1

Figure 2 is the flow chart of Algorithm 1. The detailed steps are as follows (see Appendix A for a detailed pseudocode):

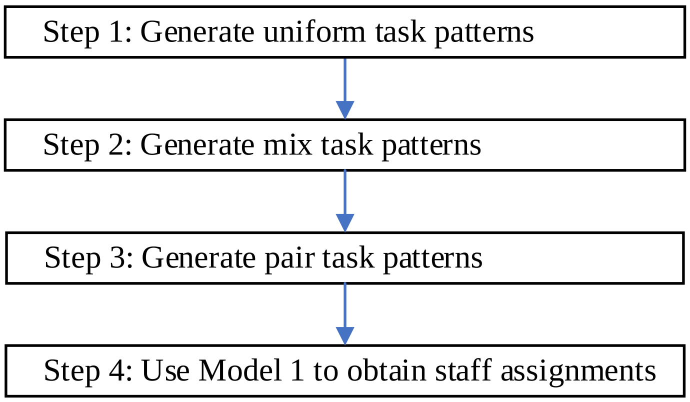
Step 1: Generate uniform task patterns
Step 2: Generate mix task patterns
Step 3: Generate pair task patterns
Step 4: Use Model 1 to obtain staff assignments

Figure 2. Flowchart of Algorithm 1

Since Algorithm 1 described below refers to Table 2 as an example, we present Table 2 again here for ease of reference.

Table 2 (Reproduced) Example of Task Demands

Task	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
Task 1	0	2	2	2	2	2	2	2	2	2	2
Task 2	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Task 3	1	1	1	1	1	1	1	1	1	1	0
Task 4	0	0	1	0	0	0	0	0	0	0	0

(Unit: person · hour)

Step 1: Generate Uniform Task Patterns.

First, the uniform task pattern requires that only one type of task be performed in one day. Specifically, all tasks are listed, and for each task *k*, two variances of task patterns are generated based on different break times: *k*-A (with a break during TS12) and *k*-B (with a break during TS13). Taking Table 2 as an example, Table 6 illustrates the uniform task patterns.

Task pattern 1-A means that task 1 is performed throughout TS8–TS18, with a break during TS12, while 1-B means the break is scheduled during TS13.

Step 2: Generate Mix Task Patterns.

Next, note that in Table 2, task 4 exhibits demand exclusively in TS10, with no demand in the remaining timeslots. Assigning staff exclusively to this task for the entire day would cause inefficient resource utilization. Therefore, we generate mix task patterns, which allocate task 4 specifically to TS10 and assign other single-task types to the remaining timeslots (given that some staff members have skills in both tasks).

Step 2-1: Identify tasks, denoted as *k*, that occur in three or fewer timeslots. Let t_k represent the timeslot where task *k* occurs.

Step 2-2: If there are staff members capable of performing both tasks k and k', generate a new task pattern. Assign task k to the timeslots t_k, and assign task k' to the other timeslots. k' is a task that is processed during most of the timeslots and has a demand of at least 1 in each timeslot.

As shown in Table 7, in task pattern (1+4)-A, the staff primarily performs task 1 and switches to task 4 at TS10, with a break scheduled during TS12.

Step 3: Generate Pair Task Patterns.

The next step is to generate pair task patterns. As shown in Table 2, task 2 has a consistent demand of 0.5 across most timeslots, which is lower than the average proficiency level of 1. Assigning a staff member exclusively to this task may result in resource waste. Therefore, we generate pair task patterns, allowing staff to handle two different tasks simultaneously within the same timeslot: one with low demand (b_{kl}<1)—provided that the staff member possesses the required skills for both tasks. This approach enhances overall resource utilization.

Step 3-1: Identify task k whose demand strictly between 0 and 1 person · hour (i.e., $0 < b_{kl} < 1$) in most timeslots.

Step 3-2: If some staff members can perform both tasks k and k' (a task with $b_{k'l} \geq 1$ in most timeslots), generate a new task pattern by assigning both tasks k and k' to the same timeslot as a task pair.

Table 6. Uniform Task Patterns (U)

Task Pattern	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
1-A	1	1	1	1	break	1	1	1	1	1	1
1-B	1	1	1	1	1	break	1	1	1	1	1
2-A	2	2	2	2	break	2	2	2	2	2	2
2-B	2	2	2	2	2	break	2	2	2	2	2
3-A	3	3	3	3	break	3	3	3	3	3	3
3-B	3	3	3	3	3	break	3	3	3	3	3
4-A	4	4	4	4	break	4	4	4	4	4	4
4-B	4	4	4	4	4	break	4	4	4	4	4

Table 7. Mix Task Pattern (M)

Task Pattern	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
(1+4)-A	1	1	4	1	break	1	1	1	1	1	1
(1+4)-B	1	1	4	1	1	break	1	1	1	1	1
(3+4)-A	3	3	4	3	break	3	3	3	3	3	3
(3+4)-B	3	3	4	3	3	break	3	3	3	3	3

As shown in Table 8, in task pattern (1&2)-A, staff members can perform task 1 and task 2 simultaneously within a single timeslot, with a break scheduled during TS12.

Table 8. Pair Task Pattern (P)

Task Pattern	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
(1&2)-A	1 2	1 2	1 2	1 2	brea k	1 2	1 2	1 2	1 2	1 2	1 2
(1&2)-B	1 2	1 2	1 2	1 2	1 2	brea k	1 2	1 2	1 2	1 2	1 2
(3&2)-A	3 2	3 2	3 2	3 2	brea k	3 2	3 2	3 2	3 2	3 2	3 2
(3&2)-B	3 2	3 2	3 2	3 2	3 2	brea k	3 2	3 2	3 2	3 2	3 2

Step 4: Use Model 1 to obtain the staff assignments.

Finally, we replace the fully enumerated task patterns with the derived task patterns and incorporate them into the set covering model. For the given set of generated task patterns, the obtained solution is optimal. However, since many task patterns are not considered during the generation process, the solution is an approximate solution to the original problem. In Section 4.3, we will evaluate the quality of this solution.

Next, we present numerical experiments where the task patterns generated by the proposed method are applied to Model 1.

4.2. Numerical experiment for Team 1

All experiments in this paper are conducted using a PC with Intel(R) Core (TM) i9-12900HQ 2.80GHz CPU, 32GB of RAM. The software environment included Python 3.9, the PuLP library, and the CBC solver. The experimental data is provided by SENKO Business Support Corporation, Japan.

In this experiment, Model 1 is used to solve the daily scheduling problem for Team 1 (Days 1-5). The team is composed of 23 staff members and is responsible for 16 tasks. The penalty for understaffing task *k* in each timeslot, denoted as w_k, is set to 0.1.

Table 9 presents the task demands in Team 1 on Day 4 (Appendix B provides the remaining input data for Team 1).

The proficiency levels and the task demand table are first input into Algorithm 1, as described in Section 4.1, to generate the task patterns. Subsequently, all input data is incorporated into the set covering model. The resulting daily staff scheduling outcomes are presented in Table 10.

As illustrated in Table 10, staff 1 is assigned a uniform task pattern, working on TK11 from TS8-TS17, with a break during TS12. Staff 13 is assigned a mixed task pattern, mainly working on TK8 from TS9 to TS18 but reassigned to TK16 during TS10, TS17, and TS18. In addition, staff 7 is assigned a pair task pattern and work on both TK1 and TK4 simultaneously within each timeslot, with a break during TS12. Since staff 10, 14, 16, 19, and 20 have a day off, they are not shown on the table.

Additionally, staff 9 and 21 do not resume work after a break, resulting in an unnatural allocation. This issue will be addressed in Section 6.

Table 11 presents a comparison of the number of task patterns and computation times for Team 1 over Days 1 to 5, under three task pattern strategies: U, UM, and UMP. Here, U refers to uniform task patterns; UM includes both uniform and mix task patterns; and UMP incorporates uniform, mix, and pair task patterns. From the perspective of task pattern quantity, even when excluding pair task patterns (i.e., without applying UMP), obtaining an optimal solution through the set covering model would require enumerating up to 16^{11} combinations. In contrast, under the UMP setting based on experimental data, the number of task patterns is reduced to a maximum of only 64.

Regarding computation time, it is evident that as task pattern complexity increases, both the number of task patterns and the corresponding computation time increase accordingly. For example, during Day 2, the computation times for the U, UM, and UMP strategies are 0.63s, 2.02s, and 3.54s, respectively. Over the five-day period, the average computation times for U, UM, and UMP are 0.77 s, 1.82 s, and 3.06 s, respectively.

Having discussed runtime, we now examine the solution quality of Algorithm 1 in Section 4.3.

Table 9. Task Demands in Team 1 on Day 4

Task ID	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
Task1	1	1	1	0	1	1	1	1	1	0	0
Task2	0	3	3	3	1	2	3	3	3	3	1
Task3	0	1	1	1	0	1	1	1	1	1	0
Task4	0	0.5	0.5	0.5	0	0.5	0.5	0.5	0.5	0.5	0
Task5	0	0	2	2	1	1	2	2	2	2	0
Task6	0	0	0	0	0	0	0	0	0	0	0
Task7	0	0	0	0	0	0	0	0	0	0	0
Task8	0	2	2	2	1	1	1	1	1	1	0
Task9	0	1	1	1	0	1	1	1	1	1	0
Task10	0	0	0	0	0	0	0	0	0	0	0
Task11	2	3	3	3	1	2	3	3	3	3	0
Task12	0	1	1	1	1	0	1	1	1	1	0
Task13	0	0	0	0	0	0	0	0	0	0	0
Task14	0	1	0	0	0	0	0	0	0	0	0
Task15	0	2	2	2	1	1	2	2	2	2	0
Task16	0	0	1	0	0	0	0	0	0	1	1

(Unit: person · hour)

Table 10. Daily Scheduling of Staff Assignments on Day 4

Staff ID	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
1	TK11	TK11	TK11	TK11	break	TK11	TK11	TK11	TK11	TK11	0
2	TK11	TK11	TK11	TK11	break	TK11	TK11	TK11	TK11	TK11	0
3	0	TK11	TK11	TK11	TK11	break	TK11	TK11	TK11	TK11	0
4	TK1	TK1	TK1	TK1	TK1	break	TK1	TK1	TK1	TK1	0
5	0	TK2	TK2	TK2	break	TK2	TK2	TK2	TK2	TK2	TK2
6	0	TK2	TK2	TK2	TK2	break	TK2	TK2	TK2	0	0
7	0	TK4 TK1	TK4 TK1	TK4 TK1	break	TK4 TK1	TK4 TK1	TK4 TK1	TK4 TK1	0	0
8	0	TK3	TK3	TK3	break	TK3	TK3	TK3	TK3	0	0
9	0	TK7	TK7	TK7	break	0	0	0	0	0	0
11	0	TK5	TK5	TK5	TK5	break	TK5	TK5	TK5	0	0
12	0	TK5	TK5	TK5	break	TK5	TK5	TK5	TK5	TK5	0
13	0	TK8	TK16	TK8	break	TK8	TK8	TK8	TK8	TK16	TK16
15	0	TK8	TK8	TK8	TK8	break	TK8	TK8	TK8	TK8	TK8
17	0	TK12	TK12	TK12	TK12	break	TK12	TK12	TK12	TK12	0
18	0	TK9	TK9	TK9	break	TK9	TK9	TK9	TK9	TK9	0
21	0	TK15	TK15	TK15	TK15	break	0	0	0	0	0
22	0	TK15	TK15	TK15	break	TK15	TK15	TK15	TK15	TK15	0
23	0	TK14	TK15	TK15	TK15	break	TK15	TK15	TK15	TK15	0

Table 11. Comparison of the Number of Task Patterns and Computation Time in Team 1

Day	Strategies	Number of Task Patterns	Computation Time (seconds)
1	U	32	0.80
	UM	38	1.94
	UMP	38*	1.94
2	U	32	0.63
	UM	56	2.02
	UMP	64	3.54
3	U	32	0.75
	UM	56	1.53
	UMP	64	3.56
4	U	32	0.80
	UM	56	1.57
	UMP	64	3.64
5	U	32	0.85
	UM	48	2.03
	UMP	56	2.63

***** no pair task pattern was generated due to zero demand for Task 4 on Day 1.

4.3. Evaluation of solution quality for algorithm 1

Here, we evaluate solution quality by comparing Algorithm 1 with a linear model that offers a theoretical lower bound.

Table 12 presents the understaffing results of three task pattern strategies—U, UM, and UMP—in Algorithm 1. To better understand the performance boundaries of Algorithm 1, a linear model is introduced as a benchmark for comparison.

The linear model is formulated by directly assigning workforce allocation for each timeslot, without using task patterns. The advantage of this model lies in its ability to assign theoretically optimal staffing levels for each timeslot without relying on predefined task patterns. However, it also has two major limitations: first, it does not restrict the number of tasks assigned within a single timeslot, which may result in more than three tasks being allocated simultaneously; second, frequent task switching may occur throughout the day, which is inconsistent with practical operational requirements for task continuity. Nevertheless, this linear model can serve as a theoretical lower bound for Algorithm 1, allowing the evaluation of how closely its solution approaches the bound.

As shown in Table 12, we first observe that as the number of task patterns increases from U to UMP, the total understaffing steadily decreases.

Furthermore, we compare the UMP-based solutions of Algorithm 1 with those of the linear model. On Day 1, both Algorithm 1 and the linear model successfully satisfied 100% of the task demands. On Day 4, based on the total task demand of 132 person · hours calculated from Table 9, the UMP solution in Algorithm 1 and the linear model yielded weighted understaffing values of 0.25 and 0.10, respectively (w_k is set to 0.1). This indicates that the solutions fulfilled 98.1% and 99.2% of the task demands, respectively. For Days 2, 3, and 5, the UMP-based solutions covered 95.5%, 94.3%, and 95.8% of the task demands, while the linear model achieved 97.9%, 98.7%, and 97.2%, respectively. These results clearly demonstrate that the solutions obtained using UMP in Algorithm 1 are close to those of the linear model, validating the effectiveness of the proposed heuristic algorithm.

To further narrow the performance gap with the linear model, the algorithm is improved in Section 5 to enhance the task pattern generation process.

Table 12. Comparison of Results in Team 1

Day	Strategies	Understaffing (Algorithm 1)	Understaffing (Linear Model)
1	U	0	0.00
	UM	0	
	UMP	0	
2	U	0.70	0.25
	UM	0.60	
	UMP	0.55	
3	U	0.80	0.15
	UM	0.70	
	UMP	0.65	
4	U	0.45	0.10
	UM	0.35	
	UMP	0.25	
5	U	0.65	0.30
	UM	0.50	
	UMP	0.45	

5. Improvement of Task Patterns

Based on the numerical experiment results presented in Section 4, further analysis reveals that in several timeslots while some tasks experience understaffing, others are overstaffed during the same timeslot.

Figure 3 illustrates the distribution of understaffing and overstaffing for Team 1 on Day 4. In this heatmap, positive values (represented in red) represent overstaffing, whereas negative values (represented in blue) indicate understaffing. As shown in the figure, overstaffing and understaffing occur simultaneously at TS17. To address this issue, the following algorithm is introduced to generate new task patterns aimed at reducing understaffing.

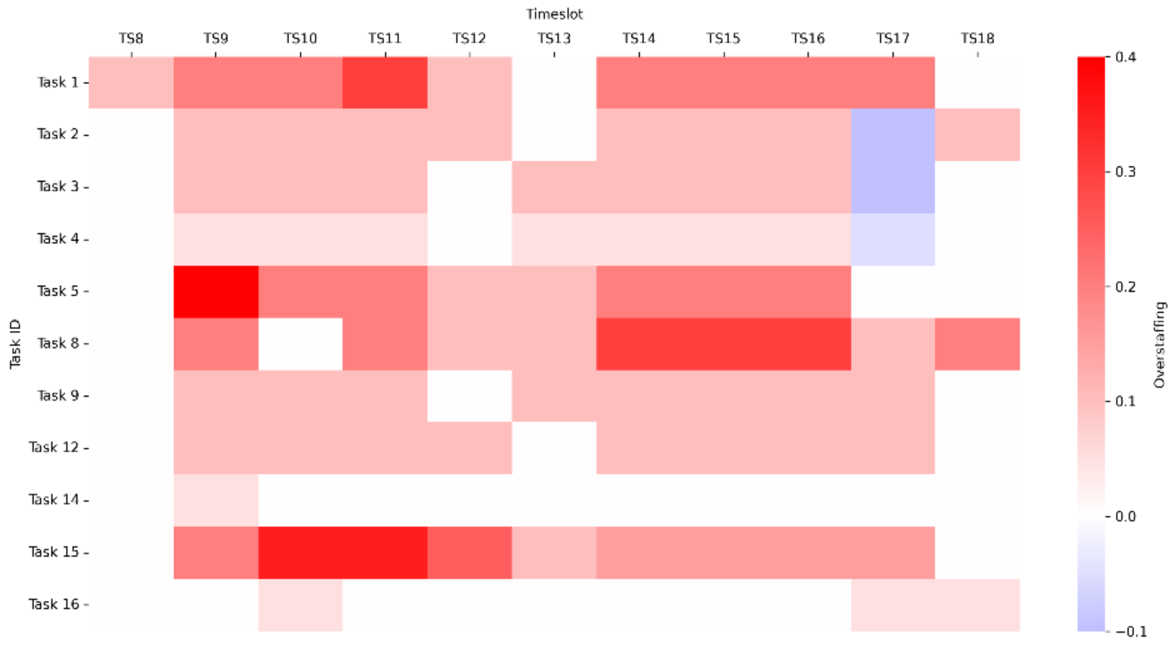

Figure 3. Distribution of overstaffing and understaffing

5.1. Algorithm 2

Figure 4 presents the flowchart of Algorithm 2. Steps 1 to 4 are the same as those in the algorithm described in Section 4, while the subsequent steps represent the task pattern improvements introduced in Algorithm 2 (see Appendix C for detailed pseudocode).

In this context, the following new symbol, c_{kl}, is introduced:

c_{kl}: amount of overstaffing for task k at timeslot l.

The following additional steps are as follows:

Step 5: Check whether the task pattern can be improved.

Step 5-1: For each task k, timeslot l, compute the values of understaffing z_{kl} and overstaffing c_{kl}.

Step 5-2: If $\exists\ l, k, k'$ such that $z_{kl} > 0$, $c_{k'l} > 0$, and at least one staff member has skills in both k and k', then proceed to Step 6. Otherwise, go to Step 7.

Step 6: Task pattern improvement.

For each combination of l, k and k' identified in Step 5-2, generate four new task patterns:

The first two patterns are mix task patterns, where task k is assigned at the timeslots l that satisfy both $z_{kl} > 0$ and $c_{k'l} > 0$, and task k' is assigned in the remaining timeslots l (see Table 13 for an example).

The other patterns are pair task patterns, where both tasks k and k' are assigned simultaneously at the timeslots l that satisfy both $z_{kl} > 0$ and $c_{k'l} > 0$, and task k' is assigned in the remaining timeslots l (see Table 14 for an example).

Note: This pair task pattern differs from that in Algorithm 1; pairing occurs only at timeslot l, while other timeslots remain assigned to k' or are set to break.

Step 6-2: Return to step 4.

Step 7: Stop.

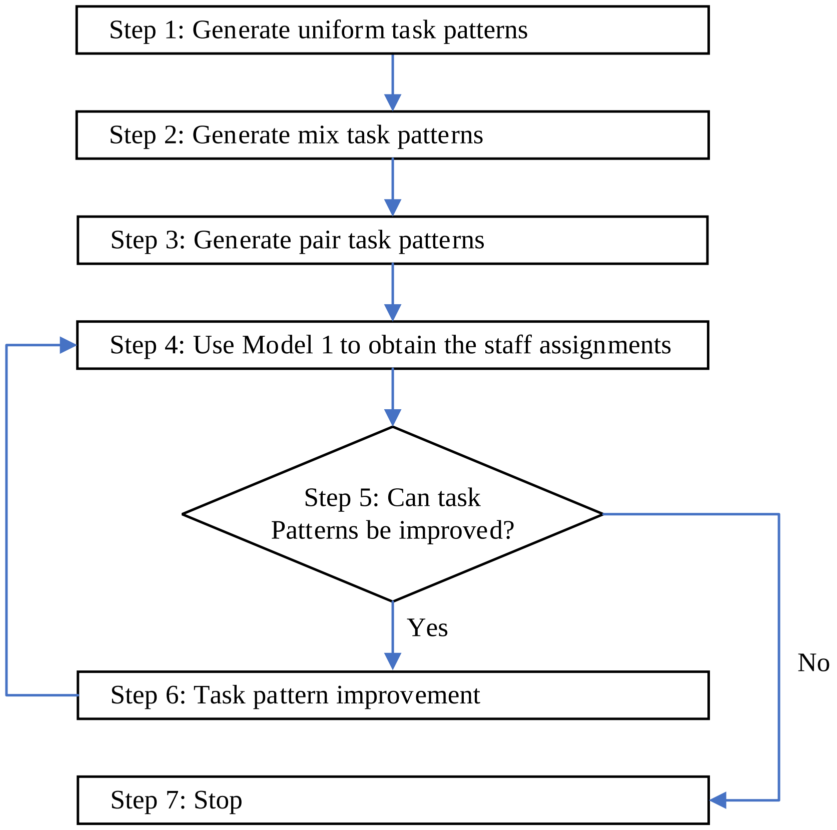
Step 1: Generate uniform task patterns
Step 2: Generate mix task patterns
Step 3: Generate pair task patterns
Step 4: Use Model 1 to obtain the staff assignments
Step 5: Can task Patterns be improved?
Yes
No
Step 6: Task pattern improvement
Step 7: Stop

Figure 4. Flowchart of pattern improvement

Table 13. Improved Mix Task Pattern Generation

Task Pattern	Timeslot									
	TS8	…	TS12	TS13	…	*l*-*1*	*l*	*l*+*1*	…	TS18
[k'+ k]-A	k'	…	break	…	…	k'	k	k'	…	k'
[k'+ k]-B	k'	…	…	break	…	k'	k	k'	…	k'

*Notation [ ] is used to avoid naming conflicts with the task patterns in Algorithm 1.

Table 14. Improved Pair Task Pattern Generation

Task Pattern	Timeslot								
	…	*l*-*1*	*l*	*l*+*1*	…	TS12	TS13	…	TS18
[k'& k]-A	…	k'	k k'	k'	…	break	…	…	k'
[k'& k]-B	…	k'	k k'	k'	…	…	break	…	k'

5.2 Numerical experiments on algorithm 2 (Team 1)

Table 15 summarizes the experimental results for Team 1 over all five days. The results show that, except for Day 1 where the optimal value had already been achieved, the objective function values of Model 1 improved on the remaining days and approached those of the linear model. Notably, on Day 4, the result reached the theoretical lower bound. These findings provide strong evidence supporting the effectiveness of Algorithm 2.

Table 15. Improvement results for 5 days

Date	Algorithm 1	Algorithm 2	Linear Model
Day 1	0.00	0.00	0.00
Day 2	0.55	0.45	0.25
Day 3	0.65	0.40	0.15
Day 4	0.25	0.10	0.10
Day 5	0.45	0.30	0.25

The improved task patterns effectively reduce understaffing and bring the solution closer to the theoretical lower bound. In the next section, we introduce Model 2 to better incorporate real-world conditions into the set covering model.

6. Model Improvements (Model 2)

6.1. Overview of model improvements

By applying the above algorithms, the task patterns are greatly reduced and improved, and these task patterns are successfully incorporated into Model 1. The model serves as a fundamental analytical framework that clearly formulates the core structure and essential constraints of the scheduling problem in a simplified and tractable form. It enables us to clarify the key decision variables and objective functions, and by comparing its solution with the theoretical lower bound, we can confirm its feasibility and usefulness. Although Model 1 abstracts from certain real-world aspects, this abstraction is intentional, as it allows us to isolate and understand the fundamental mechanism of the problem. To better align the model with real-world requirements, we introduce the following extensions and refinements, leading to the development of Model 2.

(A) Neglected Tasks (NT):

Neglected Tasks (NT) refer to tasks that have task demands but remain entirely unassigned throughout the day.

For instance, as shown in Table 16, task 1 has a demand of 1 from TS10 to TS18, while task 2 only has a demand of 1 at TS18. In this setting, assume that a staff member has the skills to perform both task 1 and task 2. Under this assumption, Model 1 may produce two different scheduling outcomes, as illustrated in Table 17.

In Scenario 1, the staff member is assigned to task 1 from TS8 to TS18. Since task 2 receives no assignment throughout the entire day, it becomes an NT. In Scenario 2, the staff member is assigned to task 1 from TS8 to TS17 and to task 2 at TS18. Although task 1 experiences a slight understaffing during one timeslot, no NT occurs in this case.

In these two scenarios, the total understaffing is both equal to 1. However, in practical applications, it is undesirable for tasks with demand to be completely neglected. Therefore, the assignment shown in Scenario 2 is regarded as more desirable. To address this issue, a penalty mechanism is introduced to prevent NT.

Table 16. Example of Task Demands for NT

Task ID	Timeslot										
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18
Task 1	0	0	1	1	1	1	1	1	1	1	1
Task 2	0	0	0	0	0	0	0	0	0	0	1

Table 17. Two Scenarios of Task Assignment

Scenario	Task Assignment	Understaffing	
		Task 1	Task 2
1	TS8-TS18: Task 1	0	1
2	TS8-TS17: Task 1; TS18: Task 2	1	0

(B) Shift Flag:

Model 1 simplifies computation by assuming that all staff take breaks during TS12 or TS13. However, this assumption overlooks the diversity of real-world shifts—some shifts do not require breaks (e.g., staff 9 and 21 in Table 10), while others require break timeslots later in the schedule.

As illustrated in Figure 5, the color highlights the working status: green cells denote idle timeslots, and orange cells indicate active assigned timeslots. The diversity of shift patterns makes it inappropriate to fix all breaks at TS12 or TS13. For example, shifts 0 and 1 do not require breaks at all. For shift 2, which starts at TS8 or TS9 and extends beyond six hours, allocating breaks around TS12 or TS13 is reasonable. However, for shifts that begin later in the day, forcing breaks at TS12 or TS13 does not reflect practical operations. Therefore, Model 2 introduces a “Shift Flag” mechanism to flexibly map break periods according to the actual structure of each shift.

Figure 6 presents examples of shift flags, which are categorized into five types: 0, 1, 2, 3, and 4. In the figure, yellow cells indicate the possible break periods corresponding to each shift flag. Specifically, shift flag 0 indicates no break time; for shift flag 1, break times may be assigned to TS12 or TS13; for shift flag 2, to TS13 or TS14 and so on.

It should be noted that as the number of potential break timeslots increases, the number of task patterns also grows accordingly. This aspect is further discussed in the numerical experiments (Section 6.3).

Shift ID	Timeslot											Shift Flag
	TS8	TS9	TS10	TS11	TS12	TS13	TS14	TS15	TS16	TS17	TS18	
0	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	0
1	🟧	🟧	🟧	🟧	🟩	🟩	🟩	🟩	🟩	🟩	🟩	0
2	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	🟩	1
3	🟩	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟩	2
4	🟩	🟩	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	🟧	3
5	🟩	🟩	🟩	🟩	🟧	🟧	🟧	🟧	🟧	🟧	🟧	4

Figure 5. Examples of Shifts and Their Corresponding Shift Flags

Shift Flag	Timeslot								
	…	TS11	TS12	TS13	TS14	TS15	TS16	TS17	…
0	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥	🟥
1	🟥	🟥	🟨	🟨	🟥	🟥	🟥	🟥	🟥
2	🟥	🟥	🟥	🟨	🟨	🟥	🟥	🟥	🟥
3	🟥	🟥	🟥	🟥	🟨	🟨	🟥	🟥	🟥
4	🟥	🟥	🟥	🟥	🟥	🟨	🟨	🟥	🟥

Figure 6. Shift Flag and Break Time Allocation Table

(C) Balanced Understaffing Constraint During Break Timeslots:

Through multiple numerical experiments, we observed that Model 1 may lead to imbalanced task assignments around the break timeslot. As shown in Table 18, consider a situation where the total understaffing for task *k* during TS12 and TS13 is 2. In Scenario 3, the entire understaffing is concentrated in TS12. However, in practice, it is preferable to distribute the understaffing more evenly across the two timeslots (Scenario 4). To better reflect this operational preference, we introduce additional constraints aimed at discouraging the concentration of understaffing in a single timeslot.

Table 18. Comparison of Understaffing Distribution Between the Two Scenarios

Task	Scenario	Understaffing	
		TS12	TS13
k	3	2	0
	4	1	1

In the process of validating numerical results, we found that the imbalance mostly occurs between TS12 and TS13, with few occurrences in other timeslots. Therefore, we introduce constraints that attempt to impose restrictions only on TS12 and TS13 to keep the fluctuation in understaffing generally within the range of -1 to 1.

6.2. Model 2 formulation

Considering the three situations (A), (B) and (C), we develop Model 2 as follows.

Symbols

This section defines several new symbols specific to the current context in addition to those defined in Section 3.1.

h = penalty for NT. A large constant ensuring that NT incur a significantly higher penalty compared to understaffing. The value of *h* is chosen such that $h \gg w_k$.
M, *ε*: *M* is a sufficiently large positive number and *ε* is a sufficiently small positive number.
C_i: the shift flag ID of the shift assigned to staff *i*.
F_j: the shift flag ID associated with task pattern *j* (as shown in Table 20).

$$\delta_{ij} = \begin{cases} 1, & \text{if } C_i = F_j \\ 0, & \text{otherwise} \end{cases}$$

The symbols C_i and F_j are solely introduced to define δ_{ij} and are not involved in other parts of the model.

Decision variables

$$u_k = \begin{cases} 1, \text{if task } k \text{ is an NT} \\ 0, \text{otherwise} \end{cases}$$

Model 2

Objective function

$$minimize\ h \sum_{k=1}^{K} u_k + \sum_{k=1}^{K} \sum_{l=8}^{L+7} w_k z_{kl} \quad (1')$$

Subject to

$$\sum_{j=1}^{J} x_{ij} = 1 \quad (\forall i = \{1,2,\dots,I\} \text{ with } s_i > 0) \quad (2)$$

$$\sum_{i=1}^{I} \sum_{j=1}^{J} x_{ij}\, a_{ls_i} p_{ik} e_{jkl} + z_{kl} \geq b_{kl} \quad (\forall k \in \{1,2,\dots,K\}, \forall l \in \{8,9,\dots,L+7\}) \quad (3)$$

$$x_{ij} = \{0,1\} \quad (\forall i \in \{1,2,\dots,I\}, \forall j \in \{1,2,\dots,J\}) \quad (4)$$

$$z_{kl} \geq 0 \quad (\forall k \in \{1,2,\dots,K\}, \forall l \in \{8,9,\dots,L+7\}) \quad (5)$$

$$x_{ij} \leq \delta_{ij} \quad (\forall i \in \{1,2,\dots,I\}, \forall j \in \{1,2,\dots,J\}) \quad (6)$$

$$\sum_{l=8}^{L+7} b_{kl} - \sum_{l=8}^{L+7} z_{kl} \geq -M \cdot u_k \quad (\forall k \in \{1,2,\dots,K\}) \quad (7)$$

$$\sum_{l=8}^{L+7} b_{kl} - \sum_{l=8}^{L+7} z_{kl} \leq M \cdot (1 - u_k) - \epsilon \quad (\forall k \in \{1,2,\dots,K\}) \quad (8)$$

$$z_{k,12} - z_{k,13} \leq 1 \quad (\forall k \in \{1,2,\dots,K\}) \tag{9}$$

$$z_{k,12} - z_{k,13} \geq -1 \quad (\forall k \in \{1,2,\dots,K\}) \tag{10}$$

$$u_k = \{0,1\} \quad (\forall k \in \{1,2,\dots,K\}) \tag{11}$$

Equation (1') primarily seeks to minimize the number of NT, while reducing understaffing. This dual objective aims to provide more efficient and balanced staff scheduling. Equations (2) - (5) are the same as in Model 1 (Section 3). Equation (6) ensures that each staff member is only allowed to select task patterns with the same shift flag of his shift. Equations (7) and (8) enforce the binary variable u_k to equal 1 if k is a NT, and 0 otherwise. Equations (9) and (10) ensure that the difference in understaffing between two consecutive break timeslots (TS12 and TS13) remains within the range of -1 to 1. The purpose of this constraint is to balance staffing levels during break timeslots.

6.3. Numerical experiment for Team 2

Team 2 consists of 22 staff members responsible for 30 tasks, and the task patterns are generated based on the heuristic algorithm proposed above.

Table 19 presents a part of the monthly shift of Team 2, where each staff member is assigned a specific shift ID for each day. Figure 7 illustrates the corresponding shift patterns, showing that Team 2 operates under 15 different shifts. Each shift is associated with a shift flag, which is visually distinguished by different colors in Figure 7. For example, shift IDs 1, 4, 5, and 15 belong to shift flag 0 (gray).

Table 19. Team 2: A part of the Monthly Shift

Staff ID	Timeslot						
	Day 1	Day 2	…	Day 27	Day 28	Day 29	Day 30
1	0	10	…	0	8	8	8
2	8	0	…	8	8	8	0
3	3	0	…	3	3	3	3
4	8	8	…	8	8	8	8
5	8	3	…	8	0	0	8
6	8	0	…	8	8	8	8
7	1	0	…	0	8	8	8
…	…	…	…	…	…	…	…
22	0	0	…	13	14	13	0

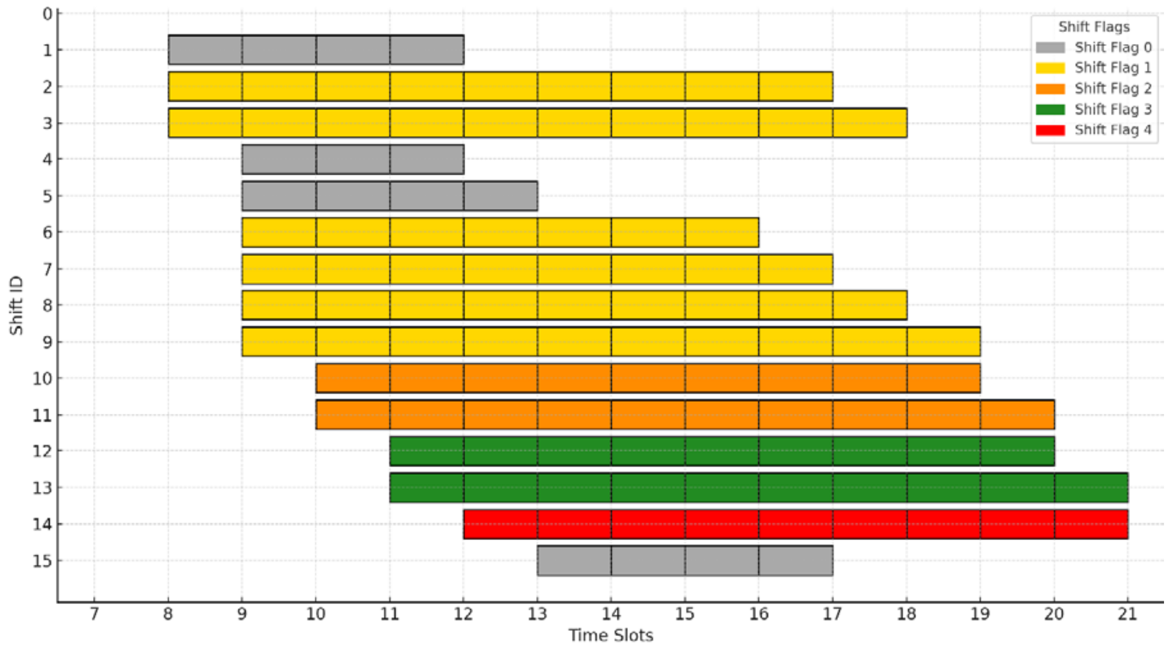

Figure 7. Team 2: Shift Pattern

In Team 2, five types of shift flags are defined: 0, 1, 2, 3, and 4 (as described in Section 6.1). Except for shift flag 0, which indicates no break timeslot, each shift flag corresponds to a specific set of permissible break timeslots. For example, shift flag 1 allows breaks during TS12 or TS13, while shift flag 2 corresponds to breaks during TS13 or TS14.

To illustrate how these shift flags influence task pattern construction in Model 2, we refer to the task patterns assigned on Day 1 (Table 20). It should be noted that Table 20 only presents the Uniform task patterns, and in Section 4, task patterns were previously denoted as 1-A and 1-B to represent task 1 with breaks assigned to TS12 and TS13, respectively. However, in this table, due to the incorporation of shift flags, the task patterns are labeled from 1-0 to 1-8 to distinguish each variation. In this manner, all types of task patterns—Uniform, Mix, and Pair—have been increased to 4.5 times their original number.

It is important to note that some task patterns share identical task assignments but differ in their shift flags (for example, 1-2 and 1-3, 1-4 and 1-5). This design is intentional, as each staff member is restricted to task patterns that match the shift flag of their assigned shift. For instance, in Table 19, staff member 2 is assigned to shift 8 on Day 1. According to Figure 7, shift 8 corresponds to shift flag 1, which means this staff member can only be assigned to task patterns such as 1-1 or 1-2, rather than 1-3 or any others.

Table 21 summarizes the one-month scheduling outcomes for Team 2. As shown in the results, the number of UMP task patterns does not exceed 400, and the computation time for each day remains below 1.43s. To provide a more intuitive comparison of the NT values and understaffing with Model 1, a comparison is provided in Figure 8.

Table 20. Team 2: Uniform Task Pattern on Day 1

Task Pattern	Timeslot							Shift Flag
	…	TS12	TS13	TS14	TS15	TS16	…	
1-0	…	1	1	1	1	1	…	0
1-1	…	break	1	1	1	1	…	1
1-2	…	1	break	1	1	1	…	1
1-3	…	1	break	1	1	1	…	2
1-4	…	1	1	break	1	1	…	2
1-5	…	1	1	break	1	1	…	3
1-6	…	1	1	1	break	1	…	3
1-7	…	1	1	1	break	1	…	4
1-8		1	1	1	1	break		4
…	…	…	…	…	…	…	…	…
30-8	…	30	30	30	30	break	…	4

Table 21. Results for Team 2

Day	Pattern Number (UMP)	Calculation Time (seconds)	Number of NT	Understaffing
1	217	0.75	0	0.43
2	181	0.56	1	0.48
3	145	0.48	0	1.20
4	397	1.43	0	1.83
5	334	1.15	0	2.90
6	379	1.37	0	1.95
7	370	1.38	0	1.75
8	199	0.65	1	1.13
9	199	0.67	0	0.75
10	145	0.46	0	1.83
…	…	…	…	…
29	307	1.15	0	1.40

Figure 8 compares the performance of Models 1 and 2 in terms of understaffing and NT. The bar charts show the daily understaffing, with blue indicating Model 2 and orange representing Model 1. The line charts show the daily number of NT, with green for Model 2 and red for Model 1. The left vertical axis represents understaffing values, whereas the right vertical axis indicates the number of NT.

As intended, a certain trade-off was observed between NT and understaffing. As shown in Figure 8, Model 2 exhibits slightly higher understaffing compared to Model 1, primarily due to the introduction of additional constraints. Meanwhile, Model 2 demonstrates a significant advantage in reducing NT, effectively mitigating the issue of unassigned low-demand tasks. Analysis indicates that the number of NT instances in Model 2 is reduced by 90.48% compared to Model 1.

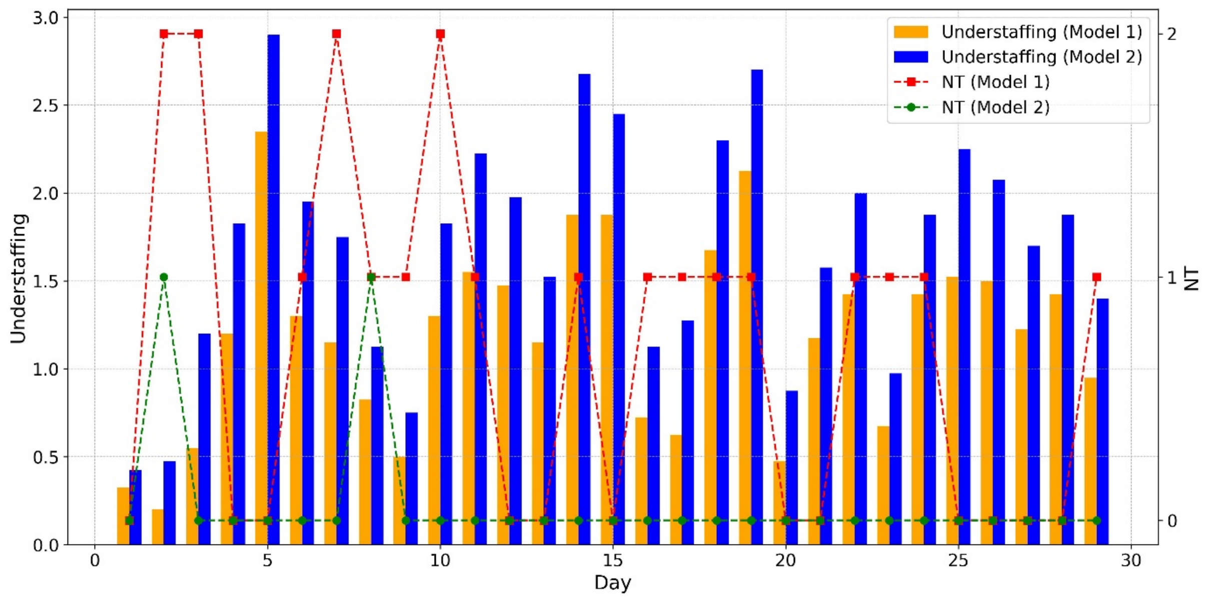

Figure 8. Comparison of understaffing and NT by model for Team 2

6.4. Results of Team 3

In addition to the numerical result mentioned above, we have conducted experiments on several other teams (one set per month) to validate the effectiveness and generalizability of the model. Here, we present only the results for Team 3.

Data for Team 3 includes 9 staff members and 22 tasks. Due to the relatively small number of staff in Team 3, the computation time and the number of NT are comparatively lower.

Figure 9 presents a comparison of understaffing and NT between models for Team 3. As shown in the figure, NT is reduced by 100%.

6.5. Summary

In this section, we present several enhancements to Model 2, including the introduction of shift flags, a penalty mechanism for NT, and additional constraints to balance understaffing during break timeslots. These improvements aim to enhance the model's practicality and adaptability to real-world scenarios. Numerical experiments provide strong evidence for their effectiveness: the incorporation of shift flags enabled more flexible break scheduling for staff. While there was a slight increase in understaffing, NT was significantly reduced—by 90.48% for Team 2 and eliminated (100%) for Team 3.

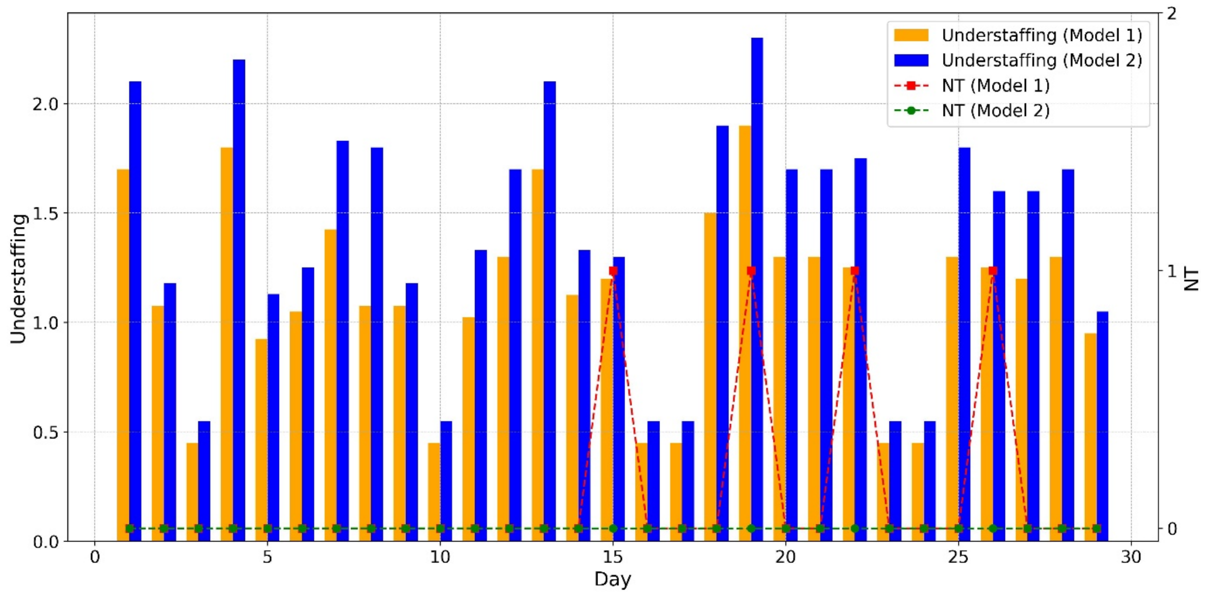

Figure 9. Comparison of understaffing and NT by model for Team 3

From a time-efficiency perspective, in the current field operations of SENKO Business Support Corporation, the daily shift scheduling for an average size team over a one-month period is typically completed by the team leader and takes more than 8 hours. In contrast, Model 2 accomplishes the same task in approximately 30 seconds. Given that the company manages about ten teams, Model 2 markedly reduces the total scheduling time.

From a practical standpoint, a web-based application is also developed to enable team leaders to complete daily shift scheduling through a simple and intuitive interface. Although numerical experiments demonstrate that the proposed model effectively reduces understaffing, a small number of NT and some understaffing remain, which require minor manual adjustments by team leaders to satisfy specific operational constraints.

The schedules generated by our model have already been adopted by the company and are currently in preparation for deployment in actual operations. Feedback from users further supports its practicality. First, team leaders emphasized that, compared with the previous manual scheduling process, the model greatly reduced the time required for preparing daily schedules. Second, although some minor manual adjustments are still required, the generated schedules are generally consistent with operational expectations, ensuring the feasibility of daily shift scheduling in practice.

7. Summary and Future Work

This study aims to apply a set covering model to address the daily shift scheduling problem in a multi-task call center. Since the traditional set covering model requires enumerating all possible task patterns, the model size grows exponentially with the number of tasks, resulting in significant computational burdens in practical applications. To address this problem, we proposed a heuristic algorithm that generated a restricted set of task patterns and integrated them into the solution process, thereby reducing the number of patterns required. To improve pattern quality, we developed Algorithm 2, which further reduced understaffing and narrowed the gap to the theoretical bound. To better reflect real-world conditions, we introduced Model 2 and validated its effectiveness through numerical experiments. The model has been adopted by the company and is now being prepared for practical deployment.

When the team size expands to several hundred members, Model 2 may incur excessive runtime or degradation of solution quality. At such extreme scales, alternative formulations or more efficient solution procedures may be required to maintain reliability. However, from an organizational perspective, large-scale teams are usually divided into several teams. In the context of this study, the largest team in the call center examined has just over one hundred members. Empirical evaluations indicate that, at this scale, Model 2 still performs well in both computational runtime and solution quality.

Additionally, we recognize that the current model still leaves room for improvement in solution quality. Moreover, although daily shift scheduling has been automated in this study, monthly schedules are still generated manually. Therefore, future work will focus on developing an integrated model that simultaneously addresses daily and monthly schedules.

Acknowledgment

This paper is based on results obtained from a project commissioned by SENKO Group Holdings Co., Ltd. and SENKO Business Support Co., Ltd.

References

[1] Aldor-Noiman, S., Feigin P. D., & Mandelbaum, A. (2010). Workload forecasting for a call center: Methodology and a case study. *arXiv preprint arXiv:1009.5741*.

[2] Aykin, T. (2000). A comparative evaluation of modeling approaches to the labor shift scheduling problem. *European Journal of Operational Research*, 125, 381–397.

[3] Avramidis, A. N., Chan, W., Gendreau, M., L’Ecuyer, P., & Pisacane, O. (2010). Optimizing daily agent scheduling in a multiskill call center. *European Journal of Operational Research*, 200(3), 822–832.

[4] Bhulai, S., Koole, G., & Pot, A. (2008). Simple methods for shift scheduling in multitask call centers. *Manufacturing & Service Operations Management*, 10(3), 411–420.

[5] Dantzig, G. B. (1954). A comment on Edie’s traffic delay at toll booths. *Journal of the Operations Research Society of America*, 2(2), 339–341.

[6] Demiriz, A., & Türker, T. (2018). An integrated approach for shift scheduling and rostering problems with break times for inbound call centers. *Mathematical Problems in Engineering*, 2018, 1–19.

[7] Ding, S., & Koole, G. (2022). Optimal call center forecasting and staffing. *Probability in the Engineering and Informational Sciences*, 36(2), 254–263.

[8] Ernst, A. T., Jiang, H., Krishnamoorthy, M., & Sier, D. (2004). Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153, 3–27.

[9] Fukunaga, A., Hamilton, E., Fama, J., Andre, D., Matan, O., & Nourbakhsh, I. (2002). Staff scheduling for inbound call and customer contact centers. *AI Magazine*, 23(4), 30.

[10] Hurkens, C. A. J., & Firat, M. (2012). An improved MIP-based approach for a multi-skill workforce scheduling problem. *Journal of Scheduling*, 15(3), 363–380.

[11] Lavoie, S., Minoux, M., & Odier, E. (1988). A new approach for crew-pairing problems by column generation with an application to air transportation. *European Journal of Operational Research*, 35, 45–58.

[12] Li, C., Liu, Z., Song, Y., Liu, H., Liu, H., & Liu, X. (2024). *Quantum computing approaches to optimize employee scheduling in multi-task call centers*. Lecture Notes in Mechanical Engineering, 3–9, Springer. doi: 10.1007/978-981-97-0194-0_1

[13] Liu, H., Liu, X., Li, C., & Song, Y. (2023). *A Two-stage Model for Multi-Task Call Center Shift Scheduling.* In Proceedings of the 9th International Conference on Engineering, Applied Sciences and Technology, 75–78.
doi: 10.1109/ICEAST58324.2023.10157399

[14] Liu, H., Song, Y., & Liu, X. (2025). Two-stage optimization for multi-task call center daily shift scheduling. To appear in *Asian Journal of Management Science and Applications.*

[15] Liu, X., Liu, H., Song, Y., & Li, C. (2023). *Improvement of Task Pattern Generation in Multi-task Call Centers.* In Proceedings of the 8th International Conference on Business and Industrial Research, 1269–1273.
doi: 10.1109/ICBIR57571.2023.10147714

[16] Mason, A, J., Ryan, D. M. R., & Panton, D. M. (1998). Integrated simulation, heuristic and optimization approaches to staff scheduling. *Operations Research*, 46(2), 161–175.

[17] Ministry of Health, Labour and Welfare. *Labour Standards Act of Japan*. https://www.mhlw.go.jp/english/ (accessed on 14 October 2025).

[18] Néron, E. (2002). *Lower bounds for the multi-skill project scheduling problem.* In Proceedings of the Eighth International Workshop on Project Management and Scheduling, 274–277, Valencia, Spain.

[19] Shen, H., Huang, J. Z., & Lee, J. W. (2008). Forecasting time series of inhomogeneous Poisson processes with application to call center workforce management. *arXiv preprint, arXiv:0807.4071.*

[20] Su, C., Li, Y., & Lee, J. (2024). A quantitative model for staffing problems in inpatient units with multi-type patients. *Queueing Models and Service Management,* 7(2), 45–62.

[21] Thompson, G. M. (1996). A simulated annealing heuristic for shift scheduling using non-continuously available employees. *Computers & Operations Research,* 23(3), 275–288.

[22] Zhao, M., Sun, J., & Nakade, K. (2025). A study of the multi-objective flexible job-shop scheduling model considering human factors. *Asian Journal of Management Science and Applications*, 8(2), 134-159. doi: 10.1504/AJMSA.2025.148898

[23] Zhang, J., Yamamoto, H., Sun, J., & Kajihara, Y. (2022). A study of optimal assignment with different workers' capacities for each process in a reset limited-cycle problem with multiple periods. *Asian Journal of Management Science and Applications*, 6, 163–188.

Appendix A. Pseudocode for Algorithm 1

Algorithm 1: Heuristic Task Pattern Generation and Staff Assignments

```
Input: Staff set $\mathcal{I}$, task set $\mathcal{K}$, timeslot set $\mathcal{L}$, demand $b_{kl}$, staff proficiency $p_{ik}$, break rule: one break at TS12 (variant "–A") or TS13 (variant "–B").
Output: Staff assignments.
Step 0: Initialization
    Initialize the task pattern set $\mathcal{J} \leftarrow \emptyset$.
Step 1: Generate Uniform Task Patterns
1.1 For each task $k \in \mathcal{K}$:
    Construct two patterns:
        $k$-A: assign task $k$ to all $l \in \mathcal{L}$, break at TS12.
        $k$-B: same rule; but break at TS13.
1.2 Add $k$-A and $k$-B to $\mathcal{J}$.
Step 2: Generate Mix Task Patterns
2.1 Define the set of $\mathcal{L}_k^+$: $\mathcal{L}_k^+ \leftarrow \{l \in \mathcal{L} | b_{kl} > 0\}$.
2.2 Identify the set of sparse tasks: $\mathcal{K}_{sparse} \leftarrow \{k \in \mathcal{K}: |\mathcal{L}_k^+| \leq 3\}$.
2.3 Identify the set of main tasks: $\mathcal{K}_{main} \leftarrow \{k \in \mathcal{K}: |\{l \in \mathcal{L} | b_{kl} \geq 1\}| \geq 6\}$.
2.4 If $\mathcal{K}_{sparse} \neq \emptyset$ and $\mathcal{K}_{main} \neq \emptyset$:
        go to Step 2.5.
    else:
        go to Step 3.
2.5 For two tasks k and $k'$ with $k \in \mathcal{K}_{sparse}$, $k' \in \mathcal{K}_{main}$:
        if $\exists i \in \mathcal{I}$ with $p_{ik} > 0$ and $p_{ik'} > 0$, construct two task patterns:
            ($k$+$k'$)-A: assign task $k$ on timeslot $l \in \mathcal{L}_k^+$; otherwise, assign task $k'$; break at TS12.
            ($k$+$k'$)-B: same rule; break at TS13.
2.6 Add ($k$+$k'$)-A and ($k$+$k'$)-B to $\mathcal{J}$.
Step 3: Generate Pair Task Patterns
3.1 Identify the set of low demand tasks: $\mathcal{K}_{low} = \{k \in \mathcal{K}: |\{l \in \mathcal{L} | 0 < b_{kl} < 1\}| \geq 6\}$.
3.2 If $\mathcal{K}_{low} \neq \emptyset$:
        go to Step 3.3.
    else:
        go to Step 4.
3.3 For each pair (k, $k'$) with $k \in \mathcal{K}_{low}$, $k' \in \mathcal{K}_{main}$:
```

if $\exists i \in \mathcal{I}$ with $p_{ik} > 0$ and $p_{ik'} > 0,$ construct two task patterns:
(k&k')-A: in every timeslot, assign both tasks k and k'simultaneously; break at TS12.
(k&k')-B: same rule; break at TS13.
3.4 Add (k&k')-A and (k&k')-B to $\mathcal{J}$.
Step 4: Use Model 1 to obtain staff assignments
Use $\mathcal{J}$ and Model 1 to obtain staff assignments.

Appendix B. Data of Team 1

Table A.1. Proficiency Levels in Team 1

Staff ID	Task ID																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
4	2	2	0	0	0	2	0	2	0	0	0	0	0	0	0	1.5	1
5	2	2	0	2	0	2	0	0	0	0	0	0	0	0	0	1.5	1
6	0	2	0	0	0	2	0	0	0	0	0	0	0	1.5	0	0	1
7	2	2	2	2	0	0	0	0	0	0	0	0	0	0	0	1.5	1
8	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	1
9	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0	0	1
10	0	0	0	0	2	2	2	0	0	0	0	0	0	0	2	1.5	1
11	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	2	2	2	2	0	0	0	0	2	0	0	1.5	1
13	0	0	0	0	0	0	0	2	2	0	0	0	2	0	1.5	1.5	1
14	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	1.5	1
15	0	0	0	0	0	0	0	2	0	2	0	0	0	0	0	0	1
16	2	2	2	0	0	0	0	0	0	0	0	2	2	0	0	1.5	1
17	0	0	0	0	0	0	0	0	2	0	0	2	2	0	0	1.5	1
18	0	0	0	0	0	0	0	0	2	0	0	2	2	1.5	0	1.5	1
19	0	0	0	0	0	0	0	0	0	0	0	0	0	1.5	0	0	1
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
21	0	0	0	0	0	0	0	0	0.75	0	0	0	0	0	2	1.5	1
22	0	0	0	0	0	0	0	0	0	0	0	0	0	1.5	2	1.5	1
23	0	0	0	0	0	0	0	0	0	0	0	0	0	1.5	1.5	0	1

(Unit: person · hour)

Table A.2. Part of the Monthly Shift

Staff ID	Timeslot				
	Day 1	Day 2	Day 3	Day 4	Day 5
1	3	3	3	3	3
2	3	3	0	3	3
3	2	2	2	2	2
4	4	0	4	3	4
5	1	1	2	1	0
6	5	5	5	5	5
7	5	5	5	5	5
8	5	5	5	5	5
9	6	6	6	10	6
10	1	2	2	0	0
11	5	5	5	5	5
12	0	0	1	2	7
13	2	2	0	1	2
14	2	2	2	0	0
15	7	1	1	1	2
16	0	1	1	0	1
17	2	0	0	2	0
18	8	2	8	2	8
19	6	6	6	0	0
20	0	0	0	0	0
21	9	9	9	9	0
22	2	1	0	2	7
23	2	8	2	2	8

Appendix C. Pseudocode for Algorithm 2

Algorithm 2: Pattern Improvement

```
Input: Staff set $\mathcal{J}$, task set $\mathcal{K}$, timeslot set $\mathcal{L}$, demand $b_{kl}$, staff proficiency $p_{ik}$, break rule: one break at TS12 (variant “–A”) or TS13 (variant “–B”).
Output: Refined staff assignments.
Steps 0-3: Identical to those in Algorithm 1.
Step 4: Use Model 1 to obtain staff assignments
    Use $\mathcal{J}$ and Model 1 to obtain staff assignments.
Step 5: Check whether the task pattern can be improved
5.1 For every ($k$, $l$) compute:
        understaffing $z_{kl}$ (already in the solution).
        overstaffing $c_{kl}$ (surplus of assigned capacity over $b_{kl}$).
5.2 Define the set of under-staffed timeslots: $\mathcal{L}_{under} \leftarrow \{l \in \mathcal{L}: \exists k \in \mathcal{K}\ with\ z_{kl} > 0\}$.
5.3 For each $l \in \mathcal{L}_{under}$ and two tasks $k,\ k' \in \mathcal{K}$:
```

```
if $z_{kl} > 0$ and $c_{k'l} > 0$ and $\exists i \in \mathcal{J}$ with $p_{ik} > 0$ and $p_{ik'} > 0$:
    set $\hat{l} \leftarrow l,\ \hat{k} \leftarrow k\ and\ \hat{k}' \leftarrow k'.$
    go to Step 6.
else:
    go to Step 7.
```

Step 6: Task Pattern improvement (generate additional task patterns)

6.1 For $\hat{l}, \hat{k}$ and $\hat{k}'$ in Step 5, generate four new patterns:

(a) Mix $[\hat{k}+\hat{k}']$-A: assign $\hat{k}$ at timeslot $\hat{l}$; assign $\hat{k}'$ at the remaining timeslots; break at TS12.

(b) Mix $[\hat{k}+\hat{k}']$-B: same rule with (a); break at TS13.

(c) Pair $[\hat{k}\&\hat{k}']$-A: assign $\hat{k}$ and $\hat{k}'$ simultaneously at timslot $\hat{l}$; at remaining timeslots assign only k'; break at TS12.

(d) Pair $[\hat{k}\&\hat{k}']$-B: same rule with (c); break at TS13.

6.2 Add patterns (a)–(d) to the set $\mathcal{J}$.

6.3 Return to Step 4.

Step 7 Stop

Output the result of Step 4.