Heuristic Approaches for Personnel Scheduling in a Call Center with Social Fairness Considerations

Anaís Veloso Silva, Telmo Pinto(✉), and Cláudio Alves

Centro ALGORITMI/LASI, Universidade do Minho, Braga, Portugal
pg34575@uminho.pt, {telmo,claudio}@dps.uminho.pt

Abstract. This work addresses a personnel scheduling problem arising in a call center company operating 24 h a day, 7 d a week, in which personnel scheduling is currently performed manually. The main objective is to develop and implement heuristic methods that can produce high-quality schedules in less time than the current manual process. The proposed approach starts by generating an initial solution through a constructive heuristic, which is then improved using local search procedures. The aim is to assign employees to work shifts while maximizing shift stability for each employee within each planning period. A second local search procedure was also implemented to reduce disparities in the number of non-working days assigned to employees, thus incorporating a social fairness perspective into the scheduling process. The constructive heuristic and both local search methods were tested on a set of instances including real instances derived from a call center, as well as randomly generated instances. The proposed approach was assessed by comparing the initial solutions with those obtained after applying the local search procedures.

Keywords: Personnel scheduling · Heuristics · Local search · Social fairness

1 Introduction

Personnel scheduling problems involve multiple decisions and have a significant impact on both financial performance and customer and employee satisfaction. In order to reduce labour costs, which arise in part from personnel scheduling decisions, and to maximise effectiveness, organisations seek work schedules that cover demand efficiently [3,10]. An inadequate personnel scheduling process may require the hiring of temporary workers, thereby increasing costs, may lead to penalties due to non-compliance with legal constraints, may increase waiting times, and may result in employee and customer dissatisfaction. Therefore, providing flexible schedules that promote employee well-being and customer satisfaction, while simultaneously ensuring an optimised personnel assignment that avoids understaffing or overstaffing and satisfies demand, is essential for organisations.

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16760, pp. 258–274, 2027.
https://doi.org/10.1007/978-3-032-30521-3_17

In the context of customer satisfaction, call centers play a central role, as they are primarily responsible for the interaction between organisations and customers and, therefore, constitute an important operational resource. The efficiency of these services is critical, since long waiting times may result in reduced customer satisfaction; accordingly, call centers must keep waiting times as low as possible. On the other hand, in order to maintain low labour costs, overstaffing should be avoided. Personnel scheduling in this sector, where several objectives must be considered and these are often conflicting, is a complex task. For this reason, call center services increasingly rely on decision-support tools to assist in this process. However, in many organisations, both in the service sector and in industry, personnel scheduling is still performed manually.

This paper addresses an optimisation problem arising in the context of personnel scheduling in a call center service company that operates 24 h a day, 7 d a week, and in which the scheduling process is carried out manually on a monthly basis [14]. In this context, the main objective is the development of a heuristic method capable of solving call center personnel scheduling problems in relatively short computational times, by constructing high-quality work schedules including shifts and days off. That is, the aim is to generate schedules that promote employee well-being and satisfy the organisation's requirements in such a way that automatic scheduling becomes preferable to manual scheduling.

The remainder of the paper is organised as follows. Section 2 reviews the state of the art. Section 3 describes the problem under study. Section 4 presents the constructive heuristic approaches. Section 5 introduces the evaluation functions, neighbourhood structures, and local search methods. Section 6 reports the computational results, and Sect. 7 concludes the paper.

2 Related Work

Personnel scheduling problems have been widely studied in both industry and the service sector [4,5]. The service sector, in particular, is labour-intensive and, therefore, effective workforce utilisation and labour cost control are essential to organisational performance [15]. Within the service sector, call centers play a central role in customer acquisition and retention, as they constitute one of the main interfaces between organisations and customers. Since most call center costs are labour-related, personnel scheduling is particularly important to their operational performance [9].

When a customer contacts a call center, the call is answered immediately if an agent is available; otherwise, the customer must wait until one becomes available. Since call arrival rates vary over time and are usually estimated from historical data, forecasts are inherently uncertain. As a result, call centers must balance low waiting times, which are critical to customer satisfaction, with low labour costs, which require avoiding overstaffing. Their objective is therefore typically to provide the required service quality at minimum labour cost [8].

A common performance measure is the service level, defined as the proportion of calls whose waiting time does not exceed a given threshold [2]. In many public-service call centers, minimum service levels are imposed by external authorities,

and failing to meet them may lead to penalties [2]. For these reasons, assigning shifts and days off in call centers is a complex task involving multiple, often conflicting, objectives, which explains the growing use of decision-support tools in this sector [15].

The literature on call center personnel scheduling includes a wide range of Operations Research methods, such as optimisation, queueing theory, and simulation [13]. A common approach consists of first determining the minimum number of employees required in each period of the day to satisfy demand and service-level requirements, and then assigning employees until all required positions are covered.

A method for minimising personnel costs in a single-skill call center is presented in [1], that is, a call center in which employees have the same skill set, have received the same training, and, in principle, provide the same quality of service, under uncertainty and service-level requirements. The implemented algorithm combines simulation and integer linear programming, with both the objective function and the constraints being evaluated by simulation.

In multi-skill call centers, where calls of different types are handled by employees with different skill sets, organisational requirements may be satisfied through several possible skill combinations, making the personnel scheduling process more challenging [2]. For example, [8] presents a hybrid method for solving an integrated personnel and task scheduling problem in call centers and contact centers. This method combines heuristic algorithms with simulation to solve the personnel scheduling and task assignment problems simultaneously.

In [12], the authors propose a simple method for assigning shifts to employees in a multi-skill call center, taking into account the stochastic nature of the call arrival process. This method is limited to call centers with at most five skills. However, it achieves short execution times and high accuracy, producing near-optimal solutions.

In [6], the authors address an integrated personnel and task scheduling problem in a multi-skill call center under several service-quality-related constraints, with the objective of minimising personnel costs. Their approach extends the method proposed in [1] to the multi-skill setting by combining iterative cutting-plane algorithms with integer programming. The authors first present an integer programming formulation for the personnel scheduling problem with service-level constraints over a planning horizon divided into periods and then adapt the ideas of [1] by initially solving a simpler simulation-based version of the problem. Since computational difficulties arise, heuristic procedures are also proposed.

Finally, [7] addresses a personnel scheduling problem in multi-skill call centers handling several types of calls, although the arrival rate of each call type is low. The proposed approach combines a service-level approximation method with a branch-and-bound algorithm to minimise personnel costs subject to minimum service-level requirements.

The proposed approach follows a heuristic optimisation framework combining a constructive heuristic with neighbourhood-based local search procedures,

which are commonly adopted in personnel scheduling problems to generate good-quality solutions within short computational times.

3 Problem Description

The problem under study arises in the context of personnel scheduling in a call center service company. The aim is to assign employees to work shifts over a one-month planning horizon divided into two periods of approximately two weeks each. When the month has an even number of days, both periods have the same length. Otherwise, the second period contains one additional day.

One of the company's objectives is to create some continuity in the assignment of employees to the same shift over a two-week horizon, that is, to maximise the number of times each employee remains assigned to the same work shift for up to two consecutive weeks. The company operates 7 d a week, 24 h a day, and has 48 shifts, starting every 30 min: the first shift begins at 00h00 and the last shift begins at 23h30. A shift corresponds to 9 consecutive hours, including a one-hour break, and can be defined by its starting time.

Employees are distributed across sites in two different cities and organised into teams according to their skill sets. There are 7 teams: 5 operate 24 h a day, 7 d a week; one operates from 06h30 to 23h00 every day; and another operates from 06h30 to 18h30 on weekdays and from 08h00 to 17h00 on weekends. Each employee must have at least 11 h of rest between shifts, may work at most 5 consecutive days per week, and must have at least 2 d off. The company also seeks to avoid overtime, except when required for operational reasons.

Personnel scheduling is currently performed monthly using a spreadsheet. Since both data input and schedule generation are carried out manually, the process requires considerable time and resources and may be difficult to manage efficiently, particularly for large or complex teams. In this context, the purpose of the proposed heuristic approach is to support automatic personnel scheduling while optimising criteria such as minimising shift changes for the same employee within a two-week period. The goal is to produce schedules that, in terms of both quality and required time, are preferable to those obtained manually.

4 Heuristic Approaches

To solve the employee-to-shift scheduling problem described in the previous section, a heuristic optimisation approach was adopted. Since the problem under study is highly constrained and large-scale, the proposed methodology combines a constructive heuristic for generating feasible initial solutions with neighbourhood-based local search procedures for solution improvement. All generated solutions were required to be feasible.

4.1 Solution Representation

In the context of this problem, a feasible solution (S, P) can be represented by a set of employees $S = (S_1, S_2, \ldots, S_{|S|})$. Each employee is associated with the shifts – represented by their starting times – to which they are assigned over the planning horizon of a given month, or with an indication of no assignment. Thus, $P = (P_1, P_2, \ldots, P_{|S|})$, where $P_i = (p_1^i, p_2^i, \ldots, p_d^i)$ for $i = 1, 2, \ldots, |S|$, and p_j^i corresponds to the starting time of the shift assigned to employee i on day j, with $j = 1, 2, \ldots, d$, and d denotes the number of days in the month under consideration.

Figure 1 shows a feasible solution for scheduling 4 employees over a one-week horizon. Each employee is associated with the starting times of their assigned shifts or an indication of no assignment. Employee A is assigned to the 08h00 shift on consecutive working days, with two consecutive days off, marked X. Employee C also works the same shift across the week but has two non-consecutive days off. Employee B has one unassigned day, and Employee D has two unassigned days, both marked with a hyphen. A day is considered unassigned when an employee is neither allocated to any shift nor given a day off. This differs from a non-working day, which encompasses any day on which an employee does not work, whether due to a scheduled day off or the absence of an operational requirement.

S	Employee A	Employee B
P	(08:00, 08:00, 08:30, 08:00, 08:30, X, X)	(10:00, 11:00, 11:00, 10:00, –, 10:00, 10:00)

S	Employee C	Employee D
P	(14:00, 10:00, 10:00, 14:30, X, 14:30, X)	(15:30, 15:30, –, 15:30, –, 15:00, 15:00)

Fig. 1. Example of a solution representation.

4.2 Initial Solution

Initially, a randomly ordered list of available employees, L_{AVAIL}, is created for each team, and its order is kept fixed throughout the first 2-week period of the month. In addition, three other lists are maintained: the list of busy employees, L_{BUSY}, the list of resting employees, L_{REST}, and the list of employees on a day off, L_{OFF}. The organisation's staffing requirements for each shift starting time on each day of the planning horizon are denoted by $MinReq_{h,d}$.

For a given shift, the list of available employees includes those who are not assigned a day off, whose working hours do not overlap with that shift, and who satisfy the minimum 11-hour rest requirement. The list of busy employees contains those whose working hours overlap with the duration of the shift. The

list of resting employees contains those who are still within the 11-hour rest period following a previous shift. Finally, the list of employees on a day off contains those who cannot be assigned to any shift during that period.

On the first day, the list of available employees contains all employees. For each of the 48 shift starting times, employees are assigned in the order in which they appear in L_{AVAIL} until the minimum staffing requirement is met. Whenever an employee is assigned to a shift, that employee is removed from L_{AVAIL} and added to L_{BUSY}. After completing the corresponding 9-hour shift, the employee is moved to L_{REST}.

From the second day onward, the list of available employees is scanned again, and employees are assigned, whenever possible, to the same shift starting time as on the previous day, subject to the staffing requirements. If the required number of employees is not reached, the corresponding shift starting times are recorded in $ShiftsToAssign$, and the remaining available employees are assigned in their current order to those shifts.

After each shift, employees who complete their working period are moved from L_{BUSY} to L_{REST}, and, after the 11-hour rest period, they are returned to L_{AVAIL} or moved to L_{OFF} if they become eligible for days off.

During the first two weeks of the month, employees with an even identification number are assigned 2 consecutive days off, whereas employees with an odd identification number are assigned 2 non-consecutive days off. During the last two weeks, this rule is reversed in order to promote a more equitable assignment of days off over the monthly planning horizon.

For clarity, the constructive heuristic is described in three parts: the first day of the first period, the remaining days within each period, and the first day of the second period. In all algorithms, the main input parameters are the randomly ordered list of available employees by team, L_{AVAIL}, and the daily staffing requirements, $MinReq_{h,d}$. The algorithms also make use of the lists L_{BUSY}, L_{REST}, L_{OFF}, and $ShiftsToAssign$, as well as the auxiliary variables $numWorking$ and $numRequired$. The main functions used are listed in Table 1. In the pseudocode, days are indexed from 0, so that $d = 0$ denotes the first day of the month.

4.3 First Day of the First Period

The procedure for the first day is described in Algorithm 1. The input parameters are the daily staffing requirements, $MinReq_{h,d}$, the randomly ordered list of available employees, L_{AVAIL}, which in this case contains all employees of a given team, and the lists L_{BUSY} and L_{REST}, both initialised as empty. For each shift starting time h, employees are assigned whenever the number of employees currently working, $numWorking$, is smaller than the minimum required number, $MinReq_{h,d}$. Assigned employees are selected in the order in which they appear in L_{AVAIL} and moved to L_{BUSY}. In addition, for each shift starting time, the lists of busy and resting employees are updated so that employees are moved from L_{BUSY} to L_{REST} after completing their shifts, and from L_{REST} back to L_{AVAIL} after the 11-hour rest period.

Table 1. Functions and corresponding functionalities

Function	Meaning
$Size(x)$	Returns the size of list x, i.e., the number of elements in the list.
$SelectEmployee(x)$	Selects the first element of list x.
$AssignShift(j, h)$	Assigns employee j to shift h.
$ReturnShiftEnd(j)$	Returns the shift ending time of employee j.
$ReturnRestEnd(j)$	Returns the time at which employee j completes the 11-hour rest period.
$ReturnDayOffEnd(j)$	Returns the time and day at which employee j completes the assigned day(s) off.
$ReturnPreviousShift(j)$	Returns the starting time of the last shift worked by employee j.
$History(j)$	Returns the record of all shifts worked by employee j.
$ReturnId(j)$	Returns the unique identification number of employee j.
$AssignDaysOff(j, h, d, x)$	When $d \in \left[1, \frac{numDays}{2} - 1\right]$, i.e., during the first two weeks of the month: If $x = 1$, 2 consecutive days off to j from h. If $x = 2$, 2 non-consecutive days off to j from h. When $d \in \left[\frac{numDays}{2}, numDays - 1\right]$, i.e., during the last two weeks of the month: If $x = 1$, 2 non-consecutive days off to j from h. If $x = 2$, 2 consecutive days off to j from h.

4.4 Remaining Days Within Each Period

The procedure for the remaining days within each period is described in Algorithm 2. This algorithm represents the behaviour of the heuristic during the first two weeks and the last two weeks of the month, excluding the first day of each period. The input parameters include the daily staffing requirements and the lists of available, busy, resting, and off-duty employees. In addition, the list of shifts to be assigned is created during the execution of the algorithm.

Similarly to the procedure used on the first day of the first period, the staffing requirements are scanned for each shift starting time to determine whether additional employees must be assigned. The heuristic first attempts to assign employees to the same shift as on the previous working day, using $ReturnPreviousShift$, in order to promote schedule stability. This process is repeated until either all requirements are satisfied or no suitable employees remain available.

If the staffing requirements are not fully satisfied, the corresponding shift starting times are recorded in $ShiftsToAssign$, with repetition whenever more

Algorithm 1 First day of the first period

Input: $L_{AVAIL}, L_{BUSY}, L_{REST}, MinReq_{h,d}$
for all $h \in [0, 48[$, with $d = 0$ **do**
  $numWorking \leftarrow \text{SIZE}(L_{BUSY})$
  if $numWorking < MinReq_{h,d}$ **then**
    $numRequired \leftarrow MinReq_{h,d} - numWorking$
    for $w \in [0, numRequired[$ **do**
      $e \leftarrow \text{SELECTEMPLOYEE}(L_{AVAIL})$
      $(S_F, P_F) \leftarrow \text{ASSIGNSHIFT}(e, h)$
      $L_{BUSY} \leftarrow L_{BUSY} \cup \{e\}$
      $L_{AVAIL} \leftarrow L_{AVAIL} \setminus \{e\}$
  for all $b \in L_{BUSY}$ **do**
    $shiftEnd \leftarrow \text{RETURNSHIFTEND}(b)$
    if $h = shiftEnd$ **then**
      $L_{REST} \leftarrow L_{REST} \cup \{b\}$
      $L_{BUSY} \leftarrow L_{BUSY} \setminus \{b\}$
  for all $r \in L_{REST}$ **do**
    $restEnd \leftarrow \text{RETURNRESTEND}(r)$
    if $h = restEnd$ **then**
      $L_{AVAIL} \leftarrow L_{AVAIL} \cup \{r\}$
      $L_{REST} \leftarrow L_{REST} \setminus \{r\}$
Output: $(S, P) = \bigcup(S_F, P_F)$

than one additional employee is required. The remaining employees in L_{AVAIL} are then assigned, in their current order, to the shift starting times stored in that list. In this process, a counter k is used to track the number of assigned employees for each shift, ensuring that the required number of employees is reached.

As in the first day, the lists of busy and resting employees are continuously updated to reflect the completion of shifts and the end of rest periods. Days off are assigned based on each employee's work history. Employees who have worked five consecutive days are identified through $History$, and the function $AssignDaysOff$ is then applied according to the employee's identification number and the current period of the month. As defined in Table 1, the interpretation of parameter x differs between the first and second halves of the month, ensuring a balanced distribution of consecutive and non-consecutive days off.

Finally, the list of employees on a day off is updated so that employees who complete their days off are returned to the list of available employees.

4.5 First Day of the Second Period

The procedure for the first day of the second period is similar to that used on the first day of the first period, since the objective is to promote schedule stability throughout the month, *i.e.*, to avoid large variations in the shifts assigned to each employee over periods of up to two weeks. However, at this stage, the list of available employees no longer contains all employees of a given team. Therefore,

Algorithm 2 Remaining days within each period

Input: $L_{AVAIL}, L_{BUSY}, L_{REST}, L_{OFF}, MinReq_{h,d}, numDays$

for $d = 1$ to $numDays - 1$ **do**
  if $d \neq \frac{numDays}{2}$ **then**
    for $h = 0$ to 48 **do**
      $numWorking \leftarrow \text{Size}(L_{BUSY})$
      if $numWorking < MinReq_{h,d}$ **then**
        $numRequired \leftarrow MinReq_{h,d} - numWorking$
        $k \leftarrow 0$
        $ShiftsToAssign \leftarrow \emptyset$
        for all $v \in L_{AVAIL}$ **do**
          $prevShift \leftarrow \text{ReturnPreviousShift}(v)$
          if $k < numRequired$ **then**
            if $h = prevShift$ **then**
              $(S_F, P_F) \leftarrow \text{AssignShift}(v, h)$
              $L_{BUSY} \leftarrow L_{BUSY} \cup \{v\}$
              $L_{AVAIL} \leftarrow L_{AVAIL} \setminus \{v\}$
              $k \leftarrow k + 1$
        while $k < numRequired$ **do**
          $ShiftsToAssign \leftarrow ShiftsToAssign \cup \{h\}$
          $k \leftarrow k + 1$
        for all $q \in ShiftsToAssign$ **do**
          if $\text{Size}(L_{AVAIL}) > 0$ **then**
            $v \leftarrow \text{SelectEmployee}(L_{AVAIL})$
            $(S_F, P_F) \leftarrow \text{AssignShift}(v, q)$
            $L_{BUSY} \leftarrow L_{BUSY} \cup \{v\}$
            $L_{AVAIL} \leftarrow L_{AVAIL} \setminus \{v\}$
      for all $l \in L_{BUSY}$ **do**
        $shiftEnd \leftarrow \text{ReturnShiftEnd}(l)$
        if $h = shiftEnd$ **then**
          $L_{REST} \leftarrow L_{REST} \cup \{l\}$
          $L_{BUSY} \leftarrow L_{BUSY} \setminus \{l\}$
      for all $g \in L_{REST}$ **do**
        $restEnd \leftarrow \text{ReturnRestEnd}(g)$
        if $h = restEnd$ **then**
          $hist \leftarrow \text{History}(g)$
          $histSize \leftarrow \text{Size}(hist)$
          if $histSize = 5$ **then**
            $empId \leftarrow \text{ReturnId}(g)$
            if $empId \bmod 2 = 0$ **then**
              $(S_F, P_F) \leftarrow \text{AssignDaysOff}(g, h, d, 1)$
              $L_{OFF} \leftarrow L_{OFF} \cup \{g\}$
              $L_{REST} \leftarrow L_{REST} \setminus \{g\}$
            else
              $(S_F, P_F) \leftarrow \text{AssignDaysOff}(g, h, d, 2)$
              $L_{OFF} \leftarrow L_{OFF} \cup \{g\}$
              $L_{REST} \leftarrow L_{REST} \setminus \{g\}$
          else
            $L_{AVAIL} \leftarrow L_{AVAIL} \cup \{g\}$
            $L_{REST} \leftarrow L_{REST} \setminus \{g\}$
      for all $f \in L_{OFF}$ **do**
        $offEnd \leftarrow \text{ReturnDayOffEnd}(f)$
        if $(h, d) = offEnd$ **then**
          $L_{AVAIL} \leftarrow L_{AVAIL} \cup \{f\}$
          $L_{OFF} \leftarrow L_{OFF} \setminus \{f\}$

Output: $(S, P) = \bigcup(S_F, P_F)$

reordering only this list would have a limited effect on the variability of the resulting assignments. For this reason, all employee lists are randomly reordered before the procedure is applied.

In addition, days off are taken into account in this procedure. The mechanism for assigning days off follows the same rule used for the remaining days of the second period: employees with an odd identification number are assigned two consecutive days off, whereas employees with an even identification number are assigned two non-consecutive days off.

5 Local Search Strategies

5.1 Evaluation Functions

For this case study, two objective functions were defined. The objective function $z_1(S, P)$ is defined as the percentage of times employees maintain the same shift over each two-week period, i.e., within each of the two periods of the month. Thus, $z_1(S, P)$ is computed as the average of the ratios between the number of identical shift starting times assigned to each employee in each period and the total number of shifts assigned to that employee in the corresponding period. More specifically, the value of $z_1(S, P)$ is computed as follows:

1. For each employee, compute the number of identical shift starting times in the first period of the month;
2. For each employee, divide this value by the total number of assigned shifts in that period, obtaining a relative frequency;
3. Compute the average of these values over all employees;
4. Repeat steps 1–3 for the second period of the month;
5. Compute the average of the two values obtained.

Therefore, $z_1(S, P)$ is defined in Equation (1) where $t_{i,1}$ and $t_{i,2}$ denote the number of identical shift starting times assigned to employee i in periods 1 and 2, respectively, and $Total_{i,1}$ and $Total_{i,2}$ denote the total number of shifts assigned to employee i in those periods. A solution is considered better when the average percentage of identical shifts assigned to each employee across both periods is higher.

$$z_1(S, P) = \frac{1}{2} \cdot \frac{\sum_{i=1}^{|S|} \left(\frac{t_{i,1}}{Total_{i,1}} + \frac{t_{i,2}}{Total_{i,2}} \right)}{|S|} \cdot 100 \tag{1}$$

After obtaining initial solutions for several instances, it was observed that the workload distribution among employees could be significantly unbalanced. In order to promote fairness and equity, a second objective function, $z_2(S, P)$, was defined, corresponding to the mean absolute deviation of the number of non-working days assigned to each employee. That is, it measures the average absolute deviation between the number of non-working days assigned to each employee and the average number of non-working days across all employees in the team. In Equation (2), $z_2(S, P)$ is defined, where x_i denotes the number of non-working days assigned to employee i and $\bar{x}$ denotes the average number of

non-working days across the team. A solution (S_A, P_A) is considered better than a solution (S_B, P_B) if $z_2(S_A, P_A) < z_2(S_B, P_B)$.

$$z_2(S, P) = \frac{\sum_{i=1}^{|S|} |x_i - \bar{x}|}{|S|} \tag{2}$$

5.2 Neighborhood Structures

Two neighbourhood structures were defined for generating neighbouring solutions (S', P') from a solution (S, P), corresponding to two distinct local search methods.

$\mathbf{NS}_1$: Shift swap between two employees. The neighbouring solutions of a solution (S, P) in neighbourhood $\mathbf{NS}_1(S, P)$ consist of all feasible solutions obtained by swapping one shift between two employees on a given day or set of days within a month.

$\mathbf{NS}_2$: Swap of one unassigned day and one shift between two employees. The neighbouring solutions of a solution (S, P) in neighbourhood $\mathbf{NS}_2(S, P)$ consist of all feasible solutions obtained by swapping one shift and one day with no assignment between two employees on a given day or set of days within a month. It should be noted that a day is considered to have no assignment for a given employee if that employee is neither assigned to any shift nor allocated a day off on that day.

The main differences between the two neighbourhood structures lie in the conditions that must be satisfied in each employee's schedule on a given day in order for exchanges, and consequently changes, to be performed, as well as in the criterion used to select the best solution, namely the corresponding evaluation function (z_1 for the first neighbourhood structure and z_2 for the second one).

5.3 Local Search Algorithms

Two local search algorithms were implemented, one for each objective function and neighbourhood structure. In both cases, the procedure terminates when no improving solution exists in the neighbourhood. In these approaches, the next current solution is selected as the neighbouring solution with the best value of the corresponding evaluation function.

Local Search Method 1 (LS1). Starting from the initial solution (S, P), algorithm LS1 searches for neighbouring solutions using neighbourhood structure $\mathbf{NS}_1$. Initially, the shift assigned to employee x_1 on the first day is swapped with the shift assigned to employee x_2 on that day, and the value of the evaluation function z_1 is computed for the resulting solution. This procedure is repeated for the remaining employees, i.e., the shift of employee x_1 on day 1 is swapped with the shifts of all other employees assigned to a shift on that day, and the value of z_1 is computed for each neighbouring solution obtained. The best solution is then selected and becomes the current solution.

After the new current solution is selected, the procedure described above is repeated for the shift of employee x_2 on the first day of the month and, in

the same way, for all employees assigned to a shift on that day. Once the best solution for that day has been identified, it is no longer modified, i.e., the shifts assigned to each employee on day 1 remain fixed until the end of the local search. The procedure is repeated for all employees assigned to a shift and for all days of the month for which such exchanges are possible, until the stopping criterion is satisfied. The solution obtained is therefore a local optimum.

Local Search Method 2 (LS2). Starting from the initial solution (S, P), algorithm LS2 searches for improved solutions through neighbourhood structure $\mathbf{NS}_2$. Initially, an exchange is performed between the schedule of employee x_1 on day 1 and the schedules of employees $x_2, x_3, x_4, \ldots, x_n$ on that same day, and, in this case, the value of the evaluation function z_2 is computed for each neighbouring solution. The best solution is then selected and becomes the current solution. This procedure is repeated for the schedule of employee x_2 on day 1, for all remaining employees, and for all other days of the month under analysis.

However, since the aim is to reduce workload imbalance among employees, i.e., the deviation between the number of non-working days assigned to each employee in that month and the average number of non-working days of the employees in the corresponding team, it is not relevant to exchange the schedules of two employees who, on the day under consideration, are both assigned to some shift, both have a non-working day, or one of them has a day off assigned. Consequently, exchanges are only performed between two employees when one of them is assigned to a specific shift on the day under consideration and the other is associated with a day with no assignment. As mentioned above, this procedure is repeated for all days of the month for which such exchanges are possible, until the stopping criterion is satisfied. The solution obtained is therefore a local optimum.

6 Computational Experiments

6.1 Instances and Computational Setting

The proposed approaches were tested using 39 instances. Ten of these instances are based on real data from a real company, although some data were modified for confidentiality reasons. The remaining 29 instances were randomly generated, based on the instances used in [11]. These instances consider different employee teams and the corresponding staffing requirements for each day across different months of the year. Each instance consists of a spreadsheet specifying the minimum number of employees required for each 30-minute period – corresponding to the starting time of each of the 48 shifts – for every day of the month under consideration. Additionally, the spreadsheet includes information about the team to be scheduled, such as employee names, unique identification numbers, and skill sets. All structures and algorithms were implemented in the C# programming language and executed on a computer equipped with an Intel Core i7-8550U processor at 1.80 GHz and 16 GB of RAM.

6.2 Results and Discussion

Table 2 presents the computational results obtained. All instances (column i) are evaluated according to the same criteria: the number of employees in each instance (n), the number of days in the planning horizon (d), the execution time in seconds (t_{exc}), the values of the objective functions (z_1 and z_2), the percentage improvement obtained for z_1 using algorithm LS1 (imp_{LS1}), and the percentage improvement obtained for z_2 using algorithm LS2 (imp_{LS2}).

From the analysis of the results obtained, it can be concluded that, for the initial solutions, the average value of z_1 is approximately 71.1%. This means that, across the different instances, employees maintain the same shift, on average, in 71.1% of the cases within each period, i.e., every two weeks. For these same instances, considering the solutions generated by the CSI constructive heuristic, the average value of z_2 is approximately 1.7, which corresponds to the mean absolute deviation in the number of non-working days assigned to employees in a given month.

Still considering the initial solutions, the instance with the highest value of z_1 (98.9%) is composed of 46 employees and corresponds to a 31-day month. For this instance, the value of z_2 is 2.7, which is substantially lower than the average. The instance with the lowest value of z_2 (0.7) is also composed of 46 employees and corresponds to a 28-day month.

The results obtained with local search method 1 (LS1) show that the value of z_1 for the initial solution is improved in 33 out of the 39 test instances and remains unchanged in the remaining instances, whereas the value of z_2 remains unchanged for all instances. The 6 instances in which the value of z_1 remains unchanged are those in which there is excess staffing, i.e., the number of employees is too high relative to the organisation's requirements. Since the number of employees is much higher than the staffing requirements, the CSI heuristic is able to produce schedules in which employees maintain the same shifts a high number of times every two weeks and, therefore, the resulting solution tends to be of good quality. In contrast, LS1 is unable to further improve the solution, since shifts are already relatively well distributed in the initial solution. The fact that the value of the objective function z_2, under LS1, remains equal to that of the initial solution in all instances can be explained by the way the neighbourhood structures were constructed. Algorithm LS1 operates on the first neighbourhood structure and aims to maximise the number of identical shifts assigned to each employee every two weeks. Therefore, in this algorithm, exchanges are only performed between two employees if both are assigned to some shift on the day under consideration. On the other hand, objective function z_2 computes the sum of the deviations in the number of non-working days assigned to each employee. Since exchanges are only performed between employees who are both assigned to shifts, the value of z_2 remains unchanged, because no employee moves from being assigned to being unassigned or associated with a day off. Employees only exchange one shift for another. Thus, LS1 cannot affect the value of z_2, which is confirmed by the reported results.

Table 2. Computational results for CSI, LS1 and LS2

			CSI			LS1				LS2			
i	n	d	z_1	z_2	t_{exc}	z_1	z_2	t_{exc}	imp_{LS1}	z_1	z_2	t_{exc}	imp_{LS2}
1	46	28	74.0	0.70	0.71	80.0	0.70	4.90	8.1	73.9	0.35	1.90	50.0
2	46	28	76.0	0.87	0.66	83.3	0.87	4.40	9.6	75.7	0.57	1.43	34.5
3	25	28	70.8	1.00	0.50	76.2	1.00	3.79	7.6	70.3	0.68	0.27	32.0
4	9	28	69.5	2.22	1.45	72.9	2.22	0.40	4.9	69.2	0.89	0.27	59.9
5	17	28	74.9	0.76	0.35	80.3	0.76	1.30	7.2	75.5	0.53	0.32	30.3
6	8	28	72.6	1.38	0.27	79.8	1.38	0.36	9.9	73.3	0.63	0.29	54.3
7	10	28	75.6	1.60	0.27	78.9	1.60	0.37	4.4	75.0	1.00	0.30	37.5
8	11	28	61.4	1.09	0.37	67.2	1.09	0.61	9.4	61.2	0.91	0.29	16.5
9	24	31	70.6	1.21	0.30	76.6	1.21	2.78	8.5	70.4	0.88	0.53	27.3
10	46	31	75.8	1.24	0.30	79.7	1.24	19.52	5.0	74.9	0.76	1.42	38.7
11	19	31	58.8	1.74	0.20	67.3	1.74	2.45	14.5	58.7	1.53	0.47	12.1
12	46	31	94.6	3.72	0.39	94.6	3.72	1.01	0.0	78.7	0.63	3.03	83.1
13	46	31	89.1	3.61	0.33	89.1	3.61	1.20	0.0	77.2	0.57	3.88	84.2
14	46	31	92.4	3.33	0.29	92.4	3.33	1.32	0.0	79.9	0.59	3.20	82.3
15	46	31	98.9	2.70	0.32	98.9	2.70	1.50	0.0	90.5	0.65	3.91	75.9
16	46	31	92.4	3.52	0.33	92.4	3.52	1.28	0.0	78.7	0.61	4.23	82.7
17	25	31	87.0	2.20	0.39	90.0	2.20	1.25	3.4	86.3	1.88	3.35	14.5
18	25	31	82.4	1.92	0.44	85.4	1.92	1.39	3.6	82.0	1.68	3.45	12.5
19	25	31	78.5	1.64	0.45	81.6	1.64	1.18	3.9	77.0	1.00	3.46	39.0
20	25	31	74.0	1.44	0.32	79.1	1.44	1.18	6.9	73.7	1.04	2.94	27.8
21	25	31	76.1	1.52	0.32	80.4	1.52	1.09	5.7	76.1	1.44	3.57	5.3
22	24	31	82.1	2.13	0.30	85.3	2.13	1.11	3.9	82.1	2.04	1.14	4.2
23	24	31	79.8	1.92	0.37	82.7	1.92	1.10	3.6	77.6	1.33	0.74	30.7
24	24	31	71.5	1.71	0.28	77.8	1.71	1.21	8.8	69.0	0.71	0.67	58.5
25	24	31	71.6	1.42	0.44	75.6	1.42	1.52	5.6	70.7	0.83	0.96	41.5
26	24	31	74.1	1.75	0.82	79.2	1.75	1.60	6.9	71.5	0.75	1.20	57.1
27	17	31	49.5	1.47	0.37	57.8	1.47	1.34	16.8	49.5	1.35	0.87	8.2
28	17	31	42.8	0.71	0.37	53.8	0.71	1.15	25.7	42.7	0.59	0.80	16.9
29	17	31	37.7	1.12	0.22	50.6	1.12	0.77	34.2	37.8	0.76	0.51	32.1
30	17	31	43.6	1.29	0.20	54.0	1.29	0.72	23.8	43.5	0.94	0.47	27.1
31	17	31	48.8	1.18	0.22	60.9	1.18	0.74	24.8	48.8	1.06	0.46	10.2
32	46	31	95.7	2.78	0.45	95.7	2.78	0.65	0.0	88.8	1.26	0.49	54.7
33	25	28	94.6	1.44	0.24	95.8	1.44	1.65	1.3	91.9	0.16	0.53	88.9
34	25	31	72.9	0.92	0.33	76.1	0.92	0.74	4.4	72.7	0.76	0.46	17.4
35	19	31	57.5	1.63	0.71	66.6	1.63	1.30	15.8	55.8	0.68	1.10	58.3
36	19	31	50.2	1.47	0.42	59.3	1.47	1.43	18.1	50.0	1.16	0.91	21.1
37	19	31	48.9	1.32	0.39	59.9	1.32	1.17	22.5	48.3	0.68	0.92	48.5
38	19	31	52.1	0.95	0.31	54.5	0.95	0.82	4.6	51.4	0.63	0.81	33.7
39	19	31	54.8	1.53	0.29	60.7	1.53	0.74	10.8	54.8	1.42	0.49	7.2
Avg			71.1	1.7	0.40	76.2	1.7	1.9	8.8	68.4	0.9	1.4	38.9

Considering the application of algorithm LS2, the value of z_1 for the initial solution improves in only 3 of the 39 test instances and remains unchanged in 5 instances, while in the remaining cases the value of z_1 decreases. In contrast, the value of z_2 improves in all test instances. These results can also be explained by the way LS2 operates. Algorithm LS2 uses the second neighbourhood structure, which was designed to minimise the mean deviation in the number of non-working days assigned to each employee, either because the employee is not assigned to any shift or because the employee is associated with a day off. In this algorithm, exchanges are only performed between two employees when one of them is assigned to a shift on the day under consideration and the other is neither assigned to a shift nor associated with a day off on that same day. Hence, the value of z_1 is unlikely to improve under LS2, since shifts that contribute positively to increasing the value of z_1 may be exchanged. As a consequence, the value of objective function z_1 may even deteriorate.

The results obtained for all instances with respect to z_1 show an average improvement of 8.8% using LS1, with an average computational time of approximately 0.4 seconds. The smallest average improvement is observed for an instance with a team of 25 employees and a 28-day month (1.3%), with a computational time of 1.65 seconds. The largest average improvement is observed for an instance corresponding to the scheduling of 17 employees over a 31-day month (34.2%), obtained in approximately 0.77 seconds.

The computational results obtained with LS2 with respect to z_2 reveal an average improvement of 38.9%, with an average computational time of approximately 1.4 seconds. In this algorithm, the smallest average improvement is observed for an instance composed of 24 employees and a 31-day month (4.2%), with a computational time of 1.14 seconds. In turn, the largest average improvement is observed for an instance composed of 25 employees and a 28-day month (88.9%), obtained in 0.53 seconds. By analysing the instances in which the improvements in the value of z_2 are greatest, it can be seen that these are the same instances for which the value of z_1 remains unchanged under LS1. As noted above, in these instances the number of employees appears to be too high relative to the organisation's requirements and, consequently, several employees remain without any assignment because they are not required to meet demand. Thus, the initial value of z_2 is relatively high. However, after applying LS2, the value of z_2 is substantially reduced, since the algorithm operates on the second neighbourhood structure.

It is also worth noting that the smallest average improvement in z_1 obtained with LS1 and the largest average improvement in z_2 obtained with LS2 are observed for the same instance. More precisely, the smallest average improvement in z_1 under LS1 is obtained for instance 33, which is also the instance for which the largest average improvement in z_2 is achieved under LS2.

The computational results also reveal a trade-off between shift stability and workload fairness. In particular, LS1 improves the stability-related objective function z_1 without affecting z_2, whereas LS2 significantly improves the fairness-related objective function z_2 but may deteriorate z_1. This behaviour reflects the

conflicting nature of the two objectives: increasing schedule stability tends to preserve existing assignments, while improving workload balance often requires modifying them.

7 Conclusion and Future Work

This work addresses a personnel scheduling problem arising in a call center service operating 24 h a day, 7 d a week. In the call center under analysis, the scheduling process is currently performed manually. Therefore, the main objective of this paper was the development and implementation of a heuristic method capable of constructing high-quality solutions while reducing the time required for this process. To this end, heuristic approaches for assigning shifts to employees were proposed. First, a constructive heuristic was implemented to obtain initial solutions. In addition, two local search methods were implemented with the aim of improving these initial solutions. The constructive heuristic and the two local search methods were tested on real instances, modified for confidentiality reasons, as well as on randomly generated instances. The constructive heuristic produced solutions of relatively good quality, i.e., with satisfactory values for the two objective functions in the context of the problem. The two local search methods yielded significant improvements in the great majority of cases.

As future work, it would be interesting to adopt a multiobjective strategy, i.e., to combine the two objective functions into a single one by assigning weights to each criterion. Moreover, it would also be important to take employee preferences regarding shift and day-off assignment into account and, in this context, to analyse other performance measures, such as employee satisfaction.

Acknowledgments. This work has been supported by FCT – Fundação para a Ciência e Tecnologia within the R&D Unit Project Scope UID/00319/2025 - Centro ALGORITMI (ALGORITMI/UM) https://doi.org/10.54499/UID/00319/2025

References

1. Atlason, J., Epelman, M.A., Henderson, S.G.: Call center staffing with simulation and cutting plane methods. Ann. Oper. Res. **127**(1), 333–358 (2004)
2. Avramidis, A.N., Chan, W., Gendreau, M., L'ecuyer, P., Pisacane, O.: Optimizing daily agent scheduling in a multiskill call center. Eur. J. Oper. Res. **200**(3), 822–832 (2010)
3. Van den Bergh, J., Beliën, J., De Bruecker, P., Demeulemeester, E., De Boeck, L.: Personnel scheduling: a literature review. Eur. J. Oper. Res. **226**(3), 367–385 (2013)
4. Brucker, P., Qu, R., Burke, E., Post, G.: A decomposition, construction and post-processing approach for nurse rostering. In: 2nd Multidisciplinary International Conference on Scheduling: Theory & Applications, MISTA 2005, pp. 397–406. University of New York (2005)
5. Burke, E.K., De Causmaecker, P., Berghe, G.V., Van Landeghem, H.: The state of the art of nurse rostering. J. Sched. **7**(6), 441–499 (2004)

6. Cezik, M.T., L'Ecuyer, P.: Staffing multiskill call centers via linear programming and simulation. Manage. Sci. **54**(2), 310–323 (2008)
7. Chevalier, P., Van den Schrieck, J.C.: Optimizing the staffing and routing of small-size hierarchical call centers. Prod. Oper. Manag. **17**(3), 306–319 (2008)
8. Fukunaga, A., Hamilton, E., Fama, J., Andre, D., Matan, O., Nourbakhsh, I.: Staff scheduling for inbound call and customer contact centers. AI Mag. **23**(4), 30–30 (2002)
9. Gans, N., Koole, G., Mandelbaum, A.: Telephone call centers: tutorial, review, and research prospects. Manuf. Serv. Oper. Manage. **5**(2), 79–141 (2003)
10. Hast, J.: Optimal Work Shift Scheduling: A Heuristic Approach. Master's thesis, Aalto University, School of Science (2017)
11. Martins, R., Pinto, T., Alves, C.: An exact optimization approach for personnel scheduling problems in the call center industry. In: International Conference on Computational Science and Its Applications, pp. 407–424. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-37108-0_26
12. Pot, A., Bhulai, S., Koole, G.: A simple staffing method for multiskill call centers. Manuf. Serv. Oper. Manage. **10**(3), 421–428 (2008)
13. Robbins, T.R., Harrison, T.P.: A stochastic programming model for scheduling call centers with global service level agreements. Eur. J. Oper. Res. **207**(3), 1608–1619 (2010)
14. Silva, A.V.: Algoritmos heurísticos para escalonamento de pessoal em call centers [Heuristic algorithms for personnel scheduling in call centers]. Master's thesis, University of Minho (2021), in Portuguese
15. Türker, T., Demiriz, A.: An integrated approach for shift scheduling and rostering problems with break times for inbound call centers. Math. Probl. Eng. **2018**(1), 7870849 (2018)