# A Greedy Algorithm for the Nurse Rostering Problem with Extra-Shifts

Ingrid Schmitdinger Vieira, Alexandre Checoli Choueiri

***Abstract*—The Nurse Rostering Problem (NRP) concerns the complex task of scheduling nurses' shifts in hospitals, taking into account many constraints and restrictions. The challenge of manually creating these schedules is considerable due to the multitude of scenarios and variables involved. In this paper we deal with a special variant of the NRP, where nurses are allowed to perform extra-shifts in order to fulfill minimal staff demands. We propose two variants of a pseudo-greedy algorithm to tackle the problem, with an extra component of randomness. This component implies that different runs output different solutions, giving an iterative dimension to the classical greedy template. Computational experiments on artificially created instances indicates that the algorithm is a viable option to be used on hospitals, providing its low total running times and fast solution convergence.**

***Index Terms*—Hospitals, work shifts, optimization, meta-heuristics.**

## I. Introduction

HOSPITALS require a working schedule for their nurses, and this schedule has a significant impact on the quality of health services provided. Although the schedule can be created manually, it is a challenging and complex task because several constraints need to be considered. These restrictions include the fact that hospitals operate continuously non-stop, the requirement that each nurse have a minimum number of days off, and other practical restrictions. These constraints can conflict with each other, making it even more challenging to create a workable schedule [7].

Creating a schedule usually takes a long time and changes can occur due to unexpected events such as health issues. The complexity and importance of this problem have attracted the attention of the combinatorial optimization scientific community. Due to the limited resources and investments of the Brazilian public sector, most hospitals manually generate personnel rosters. This paper seeks to connect real-world challenges with actionable solutions, introducing two algorithms designed to address the unique needs of a hospital setting. In this research, a specific hospital's characteristics serve as a reference point, and the project offers an effective strategy for enhancing nurse scheduling efficiency.

The paper is structured as follows: It begins with a Literature Review, showcasing a range of studies in this field and highlighting common constraints noted in similar research. Section III details the Problem Description and introduces the Greedy Algorithm, including examples and pseudocodes. This section also explains the primary data structures and two variations of our proposed solution. We discuss the Objective Function for this project in the same section. Section IV covers the solution's quality, computational times, and two conducted tests, along with their results and comparative analysis.

This paper was produced by the IEEE Publication Technology Group. They are in Piscataway, NJ.

Manuscript received November 16, 2023; revised November 16, 2023.

## II. Literature Review

Nurse Rostering (also referred to as Nurse Scheduling) is the process of creating a schedule by assigning some nurses to different shift types, e.g. day, and night, during a predetermined planning horizon, where many limitations such as hospital regulations and employee contracts as well as management and individual preferences are taken into account. The output of this process is a roster of working shifts for all the involved nurses, which is expected to result in an increase of job satisfaction and staff utilisation while reducing stress and outsourcing costs [8].

When creating an algorithm for nurse rostering problem in hospitals, constraints can be categorized as either hard or soft [2]. Hard constraints must be met for the solution to be feasible, while soft constraints should be met as much as possible, but the solution can still be feasible if some are violated. The goal is to develop a personnel plan that satisfies all hard constraints and as many soft constraints as possible, with the option to prioritize which soft constraints are more important [3]. Therefore, below follows the division of restrictions, common to many NRP problems.

### *A. Hard Constraints*

- Maximum limit on shift types: Each nurse is constrained by a predefined maximum number of shift types that can be assigned to them within the planning period.
- Maximum consecutive shifts: Nurses are subject to a constraint on the maximum number of consecutive shifts they can work within the planning period.
- Maximum consecutive workdays: A limit is imposed on the number of consecutive workdays a nurse can have without a day off.
- Free time between working shifts: There is a requirement for a specified minimum duration of free time between consecutive working shifts for each nurse.
- Minimum days off: It is necessary to ensure a minimum number of days off for a specific period within the planning horizon.

- Minimum employee count per shift: A predefined minimum number of employees must be scheduled for each shift.
- Constraints among groups/types of nurses: Constraints are applied to regulate the assignment of nurses, such as prohibiting certain nurses from working together or specifying that certain nurses must work together based on their groups or types.
- Staff demand and nurse requirements: Consideration is given to the requirements and demand for different types of nurses or staff for each shift, taking into account their skill levels and categories, with constraints imposed on the minimum, maximum, or exact number required.

### *B. Soft Constraints*

- Weekend day off: Preference is given to scheduling at least one day off on weekends, particularly favoring Sundays.
- Individual nurse preferences: Consideration is given to the predefined days off or preferences specified by each nurse, ensuring their preferences or requirements are taken into account during scheduling.
- Multiple shift assignments: The possibility of assigning a nurse to more than one shift per day is allowed, although efforts are made to minimize such occurrences as much as possible.

Due to the complexity of the problem at hand, it is not feasible to find a solution using only exact algorithms, due to the large amount of computational resources required [7]. As a result, the academic community has turned their attention to heuristic methods, which, while not offering a guaranteed optimal result, can provide an approximation, such as the technique used for the hybrid tabu search [5] and the genetic algorithms [6], among other studies. Furthermore, matheuristics, which blend the efficiency and adaptability of heuristic methods with the comprehensive search capabilities of exact methods, have emerged as a promising solution to tackle the problem.

Cheang et al. [2] performed an analysis of the existing literature regarding the modeling and solution methods employed in NRPs. Their review emphasized the distinctiveness of the solution approaches and the presence of benchmark problems for different fundamental models of NRPs. In their research, Awadallah et al. [4] introduced a hybrid approach that combines the hill climbing optimization method with an artificial bee colony. This approach replaces the employed bee operator with a hill-climbing optimizer. To assess its effectiveness, the proposed method was evaluated by comparing it to other hybridization approaches previously reported in the literature.

[3] proposes a novel hybrid algorithm combining the strengths of Integer Programming (IP) and Variable Neighbourhood Search (VNS) algorithms to design a hybrid method for solving the NRP. In [9] a local search approach is introduced which is based on a neighborhood operating on partial solutions completed by means of a greedy procedure so as to avoid the generation of infeasible solutions. Both a tabu search procedure and an iterated local search procedure are proposed in their paper.

Burke et. al. [8] presents the results of developing a branch and price algorithm and an ejection chain method for nurse rostering problems. Branch and price is a branch and bound method in which each node of the branch and bound tree is a linear programming relaxation which is solved using column generation. A recent work by Boovarsdóttir [10] adopted a research perspective by investigating how to deal with the multi-objective nature of NR problems. The authors present a general methodology to assist practitioners to set the weights of the different objective in case of single cost function expressed as the weighted sum of the multiple constraints violations.

Jin et al. [11] proposes two types of hybrid metaheuristic approaches for solving the nurse rostering problem, which are based on combining harmony search techniques and artificial immune systems to balance local and global searches and prevent slow convergence speeds and prematurity. The results show that they identify better or best known solutions compared to those identified in other studies for most instances. The results also show that the combination of harmony search and artificial immune systems is better suited than using single metaheuristic or other hybridization methods for finding upper-bound solutions for nurse rostering problems and discrete optimization problems.

Lin et al. [12] studied a NRP with joint normalized shift and day-off preference satisfaction, which contained manpower, day-off, and shift requirements. They used the work shift weight and day-off weight for each nurse to calculate his or her shift preferences. Furthermore, Lin et al. [31] developed a genetic algorithm with immigrant scheme (GAIS) and compared the results of the genetic algorithm (GA), GAIS, and GA with recovery scheme based on 20 to 100 nurses. Their results showed that GAIS was better than GA with recovery scheme.

## III. Development

In this Section we describe the specific variant of the NRP studied, as well as the algorithm tailored to solve it.

### *A. Problem Description*

This study addresses the Nurse Rostering Problem (NRP) with a focus on specific constraints that cater to the hospital where the analysis and study were conducted. Unlike other studies in the field, the assignment of nurses to the night shift is not considered in this study, as night shift professionals are exclusively allocated to the night shift at the aformentioned hospital, while daytime shift professionals do not work during the night. Additionally, it is important to highlight that, contrary to previous studies where nurse allocations could vary across different shifts, in this study, once a nurse is assigned to a specific shift, whether it be morning or afternoon, they remain in that shift for the entire planning horizon period.

Nurses have the flexibility to carry out their duties in shifts other than their assigned ones. They are permitted to work overtime, which, in our situation, is not measured by time but by additional shifts. Therefore, if a nurse is assigned overtime, it implies that they will be working on both morning and afternoon shifts. However, it is of the utmost importance to emphasize that any additional work beyond what is expected of the nurse should be minimized. The minimization of overtime hours is part of the objective function of the problem, which incurs additional costs for the hospital. Therefore, in the analysis of this problem, we have the morning shift (M), the afternoon shift (A), and the combination of both in case of overtime, referred to as (M/A).

In this paper we are solving the NRP with **4 hard** and **3 soft** constraints, as follows:

- Hard Constraints:

1) Ensure that each employee has a minimum number of days off within the planning horizon period.
2) Limit a maximum number of consecutive workdays without a day off.
3) Vacations are pre-defined, and there cannot be any shift assignments during an employee's vacation period.
4) The employee will carry out their work during the entire assigned shift period, with the possibility of working overtime.

- Soft Constraints:

1) A proportion of days off should be assigned on weekends, with the preference of Sundays.
2) Consider the pre-defined days off for each employee.
3) At each shift for each period of the planning horizon, there is a minimum number of employees that should be covered.

In order to adhere to Brazilian labor laws, certain hard constraints in this study need to be adjusted to specific values. One such constraint pertains to the maximum number of consecutive workdays without a rest, which is limited to six days. Additionally, the determination of the minimum number of days off within the planning horizon should adhere to the ratio of 5 days off for every 28 days in the planning period.

## B. Greedy Algorithm Description

In this section we describe the main greedy algorithm in details, along with a variation. To this end, we first describe the data structures used to perform calculations, followed by the algorithm mechanics and the structure of the objective function.

*1) Main data structures:* For the purpose of organizing data in a structured manner and optimizing the algorithm's processing time, the data has been systematically allocated into matrices. Three primary matrices are employed: morning_shift, afternoon_shift, and vacation_matrix. Each of these matrices is binary in nature and are initialized with zero values. In all three matrices, rows correspond to nurses while columns represent days. A value of '1' in the morning_shift and afternoon_shift indicates that a nurse is scheduled to work on a particular day, whereas '0' signifies they are not. The vacation_matrix is utilized to denote when a nurse is on vacation, indicated by a '1' value within the matrix.

An auxiliary matrix, termed overall_matrix, plays a crucial role in the algorithm's function. Each primary matrix, specifically morning_shift and afternoon_shift, is associated with an overall_matrix. Thus, there are two distinct overall_matrix instances: one for the morning shift and another for the afternoon. Every overall_matrix comprises three rows, and its length is consistent with the planning horizon.

- The first row displays the total number of nurses assigned on a specific day.
- The second row designates the minimum number of nurses that should work on the day.
- The third row captures the disparity between the current staffing and the necessary staffing (difference between row 1 and 2).

Consequently, positive values in the third row imply a staffing surplus. A value of 0 denotes the precise staffing prerequisite has been attained, and negative values highlight a staffing deficit. As the algorithm operates and nurses are allocated to their respective shifts, the overall matrices undergo corresponding updates.

Figure 1 and 2 display a schedule spanning a 14-day planning horizon for six nurses. In this scenario, Nurses 1, 3 and 5 are allocated to the morning_shift (Figure 1), while Nurses 2, 4 and 6 are designated to the afternoon_shift (Figure 2). In this illustrative example, the requisite staffing level for each day is two nurses. On days when this minimum staffing threshold is not met, nurses from the alternate shift are assigned to perform extra shifts. Extra shifts are identified when a nurse has '1' values in both the morning and afternoon shift matrices.

| | Planning horizon | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Nurse 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| Nurse 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 3 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| Nurse 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 5 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| Nurse 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Fig. 1: Morning Matrix - Example

| | Planning horizon | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Nurse 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 2 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| Nurse 3 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 4 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| Nurse 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 6 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |

Fig. 2: Afternoon Matrix - Example

As illustrated in figure 3, Nurse 6 is the only one scheduled for leave during the outlined planning period. She has time off from days 5 to 9, as indicated by a ”1” in the vacation matrix. As a result, she is available for work only from days 1 to day 4, and from day 10 to day 14. During these days, she fulfills her responsibilities in the designated shift, which is the afternoon shift.

| | Planning horizon | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
| Nurse 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Nurse 6 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

Fig. 3: Vacation Matrix - Example

2) *Description of the Algorithm:* For the scope of this project, we have chosen the Greedy Algorithm as our primary method. A greedy solution is built incrementally, starting from an empty set and selecting optimal moves at each step. We have added a stochastic component to this template in order to enable an iterative searching. The pseudo-code for the greedy procedure is shown in Algorithm 1, whereas the iterative greedy is displayed by Algorithm 2.

The algorithm assigns each nurse (line 1) to a shift. This definition is the greedy deterministic component of the method (defined in the routine DefineShiftPriority(), line 2). The priority is defined by using the overall_matrix: a nurse is assigned to the shift with the greatest staff shortfall, and at each iteration a shift is assigned to a nurse. All the nurses are evaluated sequentially in order, starting from *nurse 1* to *nurse n*, and their schedules are determined. This process persists until each nurse's shift has been allocated.

**Algorithm 1** Greedy

```
1: for (all nurses) do
2:     p = DefineShiftPriority()
3:     s = GenerateRandomStart()
4:     if nurse in VacationNurses() then
5:         VacationGreedy(p, nurse)
6:     else
7:         RandomGreedy(s,p,nurse)
8:     end if
9: end for
```

Nurses with scheduled vacation days during the planning horizon are treated differently in the scheduling process than those without any set vacation days. Specifically, for a nurse with planned vacation – indicated by her presence in the VacationNurses() function (line 4) – the algorithm employs the VacationGreedy() function (line 5), which has its own unique assignment logic.

This process begins by scheduling shifts starting the day immediately after the nurse's return from vacation (day 10 in Figure 4), continuing until the nurse reaches the threshold of

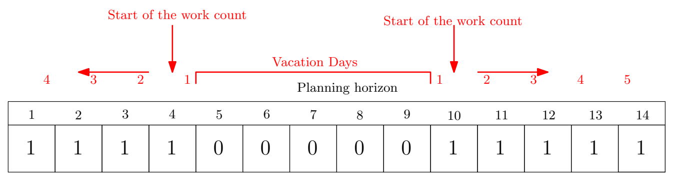


Fig. 4: Shift assignment for a nurse using VacationGreedy() - Example

the maximum consecutive working days without a break. Subsequently, the algorithm backtracks to allocate shifts counting backward from the day right before the vacation starts (day 4 in Figure 4). This strategy ensures that the nurse is optimally scheduled around their vacation, minimizing unnecessary days off. Once the nurse reaches the defined maximum working days without a day off value, a day off is scheduled, denoted by a zero in the shift assignment. This back-and-forth assignment continues until the start and end points of the planning horizon are met.

In our investigation, we put forth two distinct versions of the greedy algorithm. Both versions are intended for the regular staff, which refers to nurses without vacation plans for the planning horizon. The first variant, denoted as RandomGreedy(), operates as follows: After the shift with the most pressing staffing need is determined by DefineShiftPriority(), a random number is generated, ranging between 1 and the total number of days within the planning horizon, from GenerateRandomStart() (line 3 of Algorithm 1). On a randomly selected day, represented by day 3 in Figure 5, the nurse is assigned a ”0” value, indicating a day off. Following that, she starts her work, receiving ”1” values in her schedule until she hits the maximum working days without a day off value, as specified by the input data.

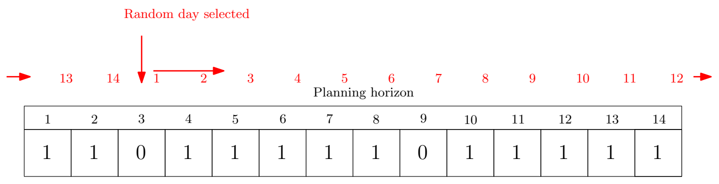
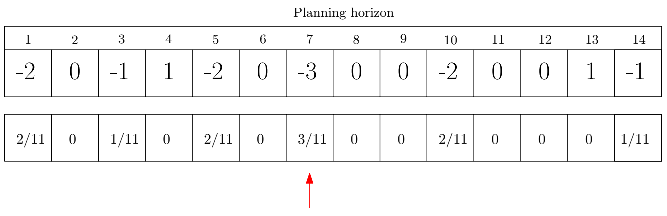


Fig. 5: Shift assignment for a nurse using RandomGreedy() - Example

After reaching this threshold of maximum working days without a day off, set at 5 days for this scenario, a day off is assigned, symbolized by a “0” value in the matrix. Work assignment then resumes post this break and continues until the planning horizon concludes and the days reset. Once a nurse's work schedule is completely outlined, the algorithm revisits and updates the overall matrices from both shifts. It then assesses whether the morning or afternoon shift has a more significant staffing requirement.

In the subsequent variant of the algorithm, named ProbabilityGreedy(), each day is accorded a percentage based on the variance in the count of available nurses. If opted to work with the ProbabilityGreedy(), this would be replacing the RandomGreedy() in line 7 of the Algorithm 1. Explicitly, days observing a shortage in nursing staff, denoted by negative figures, are given percentages in tandem with the depth of the deficit. A pronounced shortfall is paired with a heightened percentage, while a lesser one is coupled with a diminished percentage. Conversely, on days where the staffing level aligns or exceeds the stipulated threshold, characterized by values of 0 or above, a zero percentage is assigned, marking appropriate staffing levels. After formulating this percentage distribution, instrumental to the algorithm's operation, a number is selected at random between 0 and 1. As the algorithm aggregates the percentages from the set, when the sum reaches or surpasses this random value, the mechanism gets into action to delegate workers for the designated day.



Planning horizon

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -2 | 0 | -1 | 1 | -2 | 0 | -3 | 0 | 0 | -2 | 0 | 0 | 1 | -1 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 2/11 | 0 | 1/11 | 0 | 2/11 | 0 | 3/11 | 0 | 0 | 2/11 | 0 | 0 | 0 | 1/11 |

Fig. 6: ProbabilityGreedy() - Example

As shown in Figure 6, the first row provides another example of discrepancies in nurse staffing, which represents the difference between the number of allocated and required nurses for each day in the given iteration. As previously mentioned, negative values indicate understaffing, zero values show that staffing needs have been met, and positive values indicate overstaffing. The second row presents percentages attributed exclusively to the negative values in the row above. Each negative value is divided (in absolute terms) by the sum of all the negative values (also in absolute terms).

Subsequently, a random number between 0 and 1 is generated. For the purpose of this example, let's assume the number 0.7 is chosen. In this variant of the algorithm, we begin tallying the percentages attributed to negative values from left to right. Once the cumulative sum meets or exceeds the randomly generated number, the corresponding day is selected to initiate the nurse shift count.

$$\text{Cumulative sum} = \frac{2}{11} + 0 + \frac{1}{11} + 0 + \frac{2}{11} + 0 + \frac{3}{11} \tag{1}$$

$$\text{Cumulative sum} = \frac{8}{11} = 0.7273 \tag{2}$$

$$\text{Cumulative sum} \geq 0.7 \tag{3}$$

In the provided example, the cumulative sum meets or exceeds the randomly generated number on day 7, indicated by the red arrow on figure 6. This indicates that the counting will proceed according to the GreedyAlgorithm(). Once a nurse's shift schedule is established, the algorithm updates the overall matrices and proceeds to analyze the next nurse's shift requirements.

Regardless of which of the two algorithms is used, to ensure that there is no shortage of nursing staff, a specific mechanism can be employed. This algorithm incorporates a feature that allocates extra shifts to nurses. The process operates as follows: On days where there is a deficit, signifying a disparity between the required and allocated nurses, the system cross-references the opposite shift to identify available nurses. Consequently, one of the nurses from the alternate shift is designated to undertake an additional shift on the day experiencing a staffing shortfall.

As previously mentioned, the stochastic component of the greedy algorithm (line 3 of pseudocode 1) provides a possibility to enhance a solution by iterative restarts. This is done by algorithm 2. We run the greedy procedure *k* times (line 2), collecting the best solution (line 5) one at each run.

**Algorithm 2** IterativeGreedy

```
1: $t \leftarrow 1$
2: while $t \leq k$ do
3:     $X_{\text{current}} = \text{GreedyAlgorithm}()$
4:     if $X_{\text{current}} < X_{\text{best}}$ then
5:         $X_{\text{best}} \leftarrow X_{\text{current}}$
6:     end if
7:     $t \leftarrow t + 1$
8: end while
```

*3) Objective Function:* The objective function is termed to penalize soft constraints that were not satisfied. Each soft constraint is a component of the function, as described below.

1) $V_1$: Total nurses without a Sunday day off.
2) $V_2$: Requested days off not granted.
3) $V_3$: Extra shifts proportion sum.
4) $V_4$: Extra shifts count.
5) $V_5$: Total difference between required and allocated nurses.

For each component there is a penalty weight, that can be user-defined to set priorities on the achievement of each constraint. The significance of a constraint can be modulated by adjusting its penalty weight, a higher weight emphasizes its importance, whereas a reduced weight diminishes it. The objective function's calculation is outlined by equation 4. It takes into account penalties $P_1$ through $P_5$, which should be ranked based on their significance.

$$\min Z = V_1 \cdot P_1 + V_2 \cdot P_2 + V_3 \cdot P_3 + V_4 \cdot P_4 + V_5 \cdot P_5 \tag{4}$$

## IV. Computational Results

In this Section we present the computation results of the algorithms. We have separated the section in three parts, each of them designed to analyse the algorithm from a different perspective: in the first part, we compare the two variants according to their efficacy in obtaining good quality solutions, and also their running times. On the second part we determine, for each algorithm, the number of iterations for which the objective function stops improving. Finally, on the last part we check whether the objective function penalty mechanism is working accordingly (changing the penalty values of Equation 4 directly changes its correspondent value in the objective function).

## A. Quality of solutions and computational times

In our study, we've designed five test instances to evaluate the algorithms. The intent behind these designs was to ensure each instance varied significantly from the others, allowing us to test the algorithms under various scenarios and constraints. For example, in the fourth instance, some nurses are on scheduled vacation, leading to a challenging situation where staffing levels are insufficient to meet minimum requirements. This results in the need for many additional shifts. Below we describe the characteristics of each instance:

1) No nurses on vacation, requiring many additional shifts.
2) No nurses on vacation and no need for extra shifts.
3) Some nurses on vacation with few extra shifts needed.
4) Some nurses on vacation, requiring many extra shifts.
5) Relaxed scenario with ample available nurses.

In order to compare the quality of solutions and computational times, we ran both algorithms on each of the five instances 30 times, collecting data on the objective function value and each algorithm's processing time. To check whether there are differences on the values, we have compared the means using *t-student* tests.

Results of the p-values and the means are showed in Table I (for the objective function mean values), and in Table II (for the running time mean values).

| | **Instances** | | | | |
|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** |
| **p-value** | 0.946 | 0.000 | 0.205 | 0.685 | 0.902 |
| **Mean random** | 112.71 | 105.32 | 118.16 | 618.26 | 181.48 |
| **Mean prob.** | 112.74 | 102.48 | 117.87 | 618.42 | 181.45 |

TABLE I: Mean difference of objective values (p-values and means)

| | **Instances** | | | | |
|---|---|---|---|---|---|
| | **1** | **2** | **3** | **4** | **5** |
| **p-value** | 0.000 | 0.000 | 0.000 | 0.044 | 0.000 |
| **Mean random** | 4.68 | 5.20 | 7.42 | 14.32 | 6.32 |
| **Mean prob.** | 5.01 | 5.70 | 7.90 | 14.70 | 6.79 |

TABLE II: Mean difference of running times (p-values and means)

As presented in Table I, the hypothesis of equal means is rejected in only one out of five instances (Instance 2), with a significance of 5% (p-values of 0.000). This suggests that, except for one case, there is no statistically significant difference in the average quality of the solutions produced by the algorithms across the majority of instances tested.

The time analysis displayed by Table II, however, shows a different scenario. In 4 out of 5 instances the difference of the mean values for computational time, supported by the p-values, indicates that there is in fact a statistical difference among them, considering also a 5% of significance.

The conclusion so far is that it is more convenient to use the RandomGreedy(), since there will be no losses on solution quality, as well as an improve in computational times.

## B. Objective function converge

For our study, we've illustrated the performance of the objective function across 1000 solutions in graphical form. This was done for both the RandomGreedy() and the ProbabilityGreedy(). As shown in Figure 7, the x-axis displays the 1000 iterations, indicating the number of times both algorithms were run. The plotted values correspond to the mean of 10 generated solutions, for each Instance. The y-axis depicts the best objective function value achieved up to that point in the iterations. In essence, the previous best solution is only supplanted by a superior one.

Examination of the graphs reveals that both greedy algorithm variants exhibit rapid enhancement in the objective function across all five instances initially. In three out of five instances, ProbabilityGreedy() achieved superior results, while RandomGreedy() performed better in the remaining two instances.

For all instances, there is a negligible enhancement in the objective function beyond the 600th iteration, suggesting a stabilization of results at this juncture. If efficiency in processing time is a critical factor for the issue being addressed, it would be judicious to conclude the iterations at a point where subsequent enhancements to the value of the objective function become insignificant. This adjustment could serve to curtail redundant computations in favor of more optimized processing time.

## C. Penalty mechanism

We are also interested in determining whether the penalty mechanism in the objective function has a statistically significant effect on satisfying soft constraints. To accomplish this, we conducted an analysis using factorial ANOVA. For the same problem instance, we incrementally changed one penalty value while keeping all others constant. This process was repeated for three different values across all penalty weights. For each variation, we ran the algorithm and collected data on all the components of the objective function. A portion of the experiment's results is displayed in Table III.

| $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ | $V_1$ | $V_2$ | $V_3$ | $V_4$ | $V_5$ |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 5 | 21 | 10 | 78 | 0 |
| 1 | 1 | 1 | 1 | 1 | 8 | 19 | 4 | 73 | 5 |
| 1 | 1 | 1 | 1 | 1 | 7 | 19 | 10 | 78 | 0 |
| 1 | 1 | 1 | 1 | 1 | 4 | 22 | 10 | 78 | 0 |
| 1 | 1 | 1 | 1 | 1 | 5 | 26 | 4 | 73 | 5 |
| 1 | 1 | 1 | 1 | 1 | 5 | 21 | 10 | 78 | 0 |
| 100 | 1 | 1 | 1 | 1 | 1 | 20 | 9 | 79 | 1 |
| 100 | 1 | 1 | 1 | 1 | 2 | 27 | 9 | 79 | 1 |
| 100 | 1 | 1 | 1 | 1 | 2 | 28 | 13 | 75 | 3 |
| 100 | 1 | 1 | 1 | 1 | 3 | 26 | 9 | 79 | 1 |
| 100 | 1 | 1 | 1 | 1 | 3 | 21 | 9 | 79 | 1 |
| 100 | 1 | 1 | 1 | 1 | 3 | 25 | 11 | 77 | 1 |

TABLE III: Factorial ANOVA table

We have conducted five Factorial ANOVA tests, each corresponding to a different component of the objective function. For instance, to evaluate component $V_1$, we used as input columns $P_1$ through $P_5$ (factor) and only $V_1$ as a response variable. To evaluate $V_2$ we use the same $P_1$ through $P_5$ as

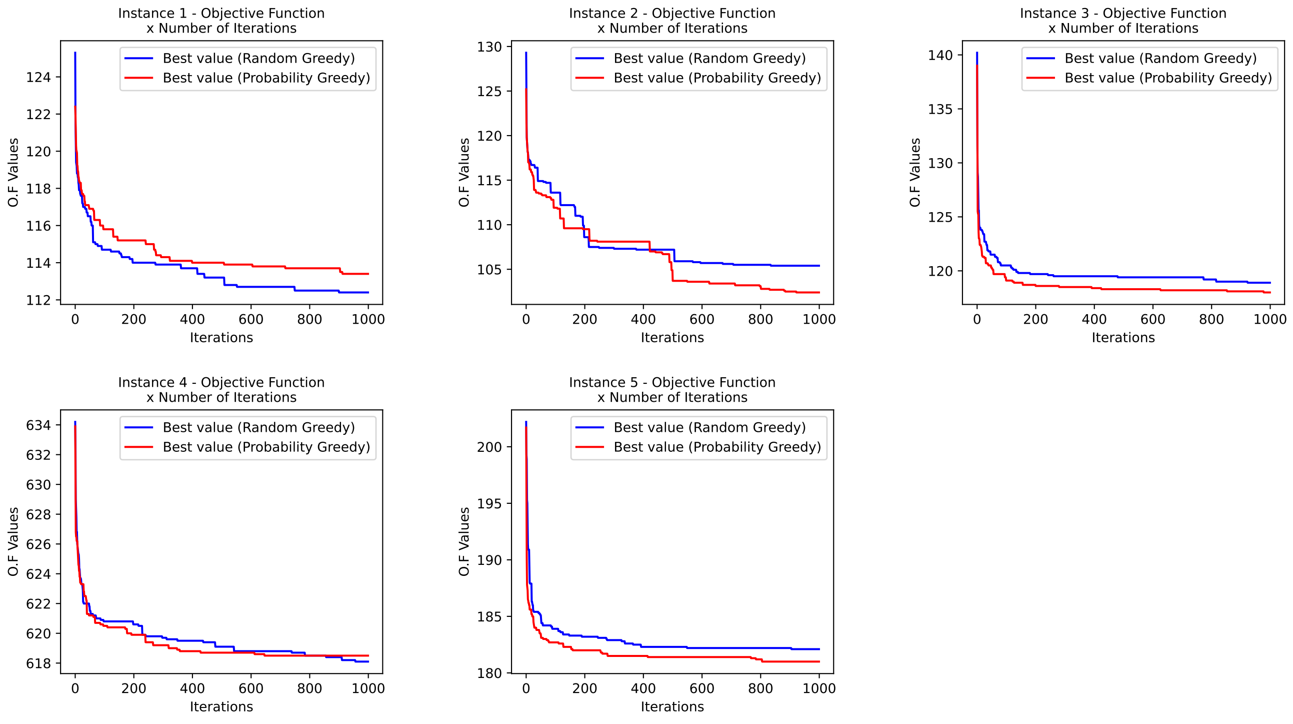


Fig. 7: Objective Function x Number of Iterations - Example

factors, but $V_2$ as the response variable. By conducting the tests in this way, we obtained five sets of results, each providing a p-value for the various factors. The results of these tests are presented in IV.

| | Penalty components | | | | |
|---|---|---|---|---|---|
| Objective value components | $\mathbf{P_1}$ | $\mathbf{P_2}$ | $\mathbf{P_3}$ | $\mathbf{P_4}$ | $\mathbf{P_5}$ |
| $\mathbf{V_1}$ | 5.69e-12* | 0.03047 | 0.0082 | 0.0027 | 0.8173 |
| $\mathbf{V_2}$ | 6.85e-05 | 3.31e-16* | 0.0196 | 4.29e-06 | 0.8472 |
| $\mathbf{V_3}$ | 0.3389 | 0.0398 | 1.27e-11* | 0.0466 | 0.5985 |
| $\mathbf{V_4}$ | 0.0467 | 5.02e-09 | 3.77e-05 | $< 2e-16$* | 0.7642 |
| $\mathbf{V_5}$ | 0.000482 | 4.91e-06 | 0.383924 | $< 2e-16$ | 0.515270* |

TABLE IV: p-values for Factorial ANOVA, penalties and objective components

We are primarily interested in the diagonal values of Table IV, as these indicate whether the weight affects the corresponding component of the objective function. For instance, the first row of Table IV shows a p-value of $5.69e-12$ for weight $V_1$ and the response variable $P_1$, suggesting that, even with a significance level of $1\%$, it influences $V_1$. This pattern persists, except for the last row, where the weight $P_5$ and component $V_5$ are considered.

We can therefore conclude that the penalty mechanism is effective for all weights, except for $P_5$. This behavior might partly be explained by the type of instance: since the weight is linked to the shortfall of nurses relative to the minimum required, if the minimum demand for nurses is so high that even when assigning all available nurses to extra shifts the shortfall persists, it becomes impossible to reduce component $P_5$, regardless of the weight value.

## V. Conclusions

In this study, we introduced two variants of the Greedy algorithm to tackle the nurse rostering problem (NRP): RandomGreedy(), which introduces a random factor, and ProbabilityGreedy(), which incorporates a probabilistic factor. The primary difference between the two lies in their respective strategies for determining the starting day for nurse work shift allocations. Our proposed methodologies take into account a wide array of hospital constraints—both hard and soft, of which are integrated into an objective function designed to assess the efficacy of the scheduling solutions.

To assess the performance of the proposed algorithms, we employed five distinct instances, each with its own set of requirements and complexities, in two different tests. One of the primary evaluations was the t-student test. This test found no statistically significant differences in the quality of solutions between the two variants of the Greedy algorithm in terms of the objective function. However, it was observed that RandomGreedy() demonstrated enhanced computational efficiency compared to ProbabilityGreedy().

Furthermore, the ANOVA test indicated that emphasizing certain components of the objective function—by assigning them greater weights—can effectively diminish the incidence of less optimal results.

Conclusively, the application of the Greedy algorithm-based solutions to the NRP across diverse operational scenarios has met our initial expectations and is deemed effective. The algorithm we developed has proven to be a significantly better approach than manually creating nurse rosters, as it not only enhances the quality of the work schedules but also results in substantial time savings.

## References

[1] Johannes Hendrik Oldenkamp. Quality in fives: on the analysis, operationalizationand application of nursing schedule quality. 1996.

[2] Brenda Cheang, Haibing Li, Andrew Lim, and Brian Rodrigues. Nurse rosteringproblems—-a bibliographic survey.European journal of operational research, 151(3):447–460, 2003.

[3] Erfan Rahimian, Kerem Akartunalı, and John Levine. A hybrid integer programming and variable neighbourhood search algorithm to solve nurse rostering problems. European Journal of Operational Research, 258(2):411–423, 2017.

[4] Awadallah, M.A.; Bolaji, A.L.; Al-Betar, M.A. A hybrid artificial bee colony for a nurse rostering problem. Appl. Soft Comput. 2015, 35, 726–739

[5] Burke, E.; De Causmaecker, P.; Berghe, G.V. A hybrid tabu search algorithm for the nurse rostering problem. In Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning, Canberra, Australia, 24–27 November 1998; pp. 187–194.

[6] Aickelin, U.; Dowsland, K.A. An indirect genetic algorithm for a nurse-scheduling problem. Comput. Oper. Res. 2004, 31, 761–778.

[7] El-Ghazali Talbi.Metaheuristics: from design to implementation. John Wiley & Sons, 2009.

[8] Edmund K. Burke, Tim Curtois, New approaches to nurse rostering benchmark instances, European Journal of Operational Research, Volume 237, Issue 1, 2014.

[9] F. Bellanti, G. Carello, F. Della Croce, R. Tadei, A greedy-based neighborhood search approach to a nurse rostering problem, European Journal of Operational Research, Volume 153, Issue 1, 2004.

[10] Boovarsdottir, E. B., Smet, P., & Berghe, G. V. (2020). Behind-the-Scenes Weight Tuning for applied nurse rostering. Operations Research for Health Care, 26, 100265.

[11] Suk Ho Jin, Ho Yeong Yun, Suk Jae Jeong, and Kyung Sup Kim. Hybrid andcooperative strategies using harmony search and artificial immune systems for solvingthe nurse rostering problem.Sustainability, 9(7):1090, 2017.

[12] Lin C.C., Kang J.R., Chiang D.J., Chen C.L. Nurse scheduling with joint normalized shift and day-off preference satisfaction using a genetic algorithm with immigrant scheme Int. J. Distrib. Sens. Netw. (2015), Article 595419