Proceedings of the Annual General Donald R. Keith Memorial Conference
West Point, New York, USA
April 23, 2026
A Regional Conference of the Society for Industrial and Systems Engineering

Creating Balanced Nursing Schedules for Medical-Surgical Units

Aiden Small, Austin Parker, Brock Bewley, Garrett Munoz, and Kaden Alvarado

Department of Industrial Engineering, University of Arkansas, Fayetteville, Arkansas 72701

Corresponding author's Email: ajsmall2004@gmail.com

Author Note: All five authors are senior year industrial engineering capstone students at the University of Arkansas.

Abstract: Nurse scheduling is a highly manual, time-consuming process that often requires scheduling managers to consider human factors alongside labor and contract requirements. Building a flawed schedule can lead to increased labor hours and unsatisfied employees, contributing to the high turnover rate commonly seen in the healthcare industry. This project develops a nurse scheduling tool that generates schedules using a hybridized genetic algorithm to solve an integer linear program, rather than relying on a commercial solver. The model reduces average staffing deviation by 53% while generating schedules in under five minutes. This model is embedded in a user interface that allows managers to make post-processing decisions that a mathematical model cannot account for. All backend applications are written in locally hosted Python. Overall, this tool reduces scheduling time by 10-15 minutes and enables managers to focus more on patient care.

Keywords: Integer Linear Programming, Perceptual Hashing, Nurse Scheduling, Heuristic Optimization

1. Introduction

Healthcare organizations have faced the nurse scheduling problem for decades, and the issue has intensified in recent years due to workforce shortages caused nurse burnout reaching levels of near 50% (Lasater & Aiken, 2026). As healthcare systems attempt to maintain high-quality patient care with limited staffing resources, effective nurse scheduling and labor allocation have become increasingly important operational challenges. Nurse scheduling must account for staffing ratios, nurse qualifications, and operational requirements while managers construct schedules using a limited resource pool to satisfy patient care and staffing guidelines. In many hospitals, schedules are still produced manually without standardized or data-driven analytical tools, which can lead to imbalanced staffing ratios, uneven experience distribution, and costly overtime.

Baptist Memorial Healthcare (BMHCC) experiences similar challenges when balancing nursing schedules. This project focuses on the scheduling procedure used in the medical-surgical units at Baptist Memorial Hospital–DeSoto. Nurse scheduling operates on a rolling six-week planning horizon. At the beginning of each cycle, Group C core employees establish baseline staffing coverage. During Weeks 1–2, Group A nurses (PTE/FTE) select preferred shifts through a self-scheduling period, while Group B nurses (PRN) fill out remaining shifts during Weeks 3–4. In Week 5, nurse managers review and balance the schedule for Weeks 7–12, making manual adjustments to ensure staffing ratios and employee availability. The finalized schedule goes live in Week 7, after which the next scheduling cycle begins.

Although this system allows some flexibility through self-scheduling, the final schedule still relies heavily on manual adjustments and managerial experience, which can result in uneven labor allocation or insufficient nurse coverage on certain shifts. To address this issue, this project develops a data-driven decision support tool that assists nurse managers in generating balanced nursing schedules. The system integrates a multi-objective integer linear programming (ILP) model to identify schedule configurations that optimize unit performance, nurse satisfaction, and cost. Because solving the full ILP directly can lead to poor computational performance and requires costly commercial solvers, the system instead applies a two-stage heuristic approach using simulated annealing to generate feasible schedules and a genetic algorithm to refine nurse assignments.

2. Data

For this project, data is defined as a pre-balanced nurse schedule for a given six-week scheduling period. To convert this data into a usable format, we developed a standalone Python tool that can parse all data contained in each sheet and generate standalone data files that are easily manipulated. An unprocessed schedule sheet contains two types of data that our utility understands and interprets: in-text data, which represents information about each employee and their shifts, and images, which contain information such as PTO and availability. Due to their structural differences, both types are handled separately within the utility.

Our utility first creates a list of names containing all employees who will be included in our model. During this step, the tool excludes employees in the schedule who are out of scope for our project or are not scheduled regularly enough to warrant consideration in our model, such as nurse interns.

The next step creates a function that generates single-column matrices required for model parameterization. Information about each employee, such as type (Registered Nurse, Patient Care Technician, etc.), weekly hour requirements, and special contract designations (such as weekenders), is output into single-column matrices that populate their respective parameters. Our function uses a type of regex to pass strings into the function, creating a matrix based on any set of characters. Keeping the function dynamic allows for less code, faster runtimes, and easier adaptability. Once the target string is set, the function creates a binary matrix based on whether the string is present in the target column.

Our tool helps nurse managers in the balancing phase of the scheduling process, meaning that both self-scheduling periods have occurred before schedule processing. Our Python utility creates a matrix indicating when each employee requested to work specific shifts. Shifts within our tool and BMHCC's pre-existing scheduling software use two different indices: BMHCC's has a column for each day, and ours has a column for each 12-hour shift. To handle this, we find the index for the starting day of the scheduling block and add it to the current day of the current week. Our output index then adjusts to add one for the day and night shift, accounting for the two-to-one transition between sets. After indexing, the function operates similarly to the previous matrices, setting one if an employee has requested a shift and zero otherwise.

Hashing is a technique used by computers to help decipher images and translate their contents into a string of characters. There are a variety of methods that employ different strategies to complete this task, so, given the nature of our data, we chose a method that focuses on image structure. Perceptual hashing (pHash) is a relatively quick, low-compute process that mimics human thought processes when hashing an image. To generate the hash, the method first converts an image to grayscale and reduces its aspect ratio to ensure all images are in a consistent format. A discrete cosine transform is then applied, a process that involves comparing the image to a set of cosine waves and determining the extent to which each cosine wave is present. Each comparison is assigned a coefficient and placed into a matrix, with only the top-left 64 coefficients considered. The algorithm selects these 64 coefficients to reduce computational costs while preserving the image's most important structural components. The final step in generating the hash involves finding the median of these coefficients, then comparing each coefficient against it and assigning a binary value based on whether the coefficient exceeds the median.

A key concept in our team's use of perceptual hashing is the creation of a threshold value that determines whether an image matches another contained in a predefined image bank. To do this, we use Hamming Distance to compute the difference between two hashes. Hamming Distance counts the number of differences in a hash and gives them a numeric value. If hash A contains four zeros, and hash B contains two zeros and two ones, then the Hamming Distance is two, meaning we can set a threshold on the allowable range of Hamming Distance differentiation. To ensure the distances were set correctly, we tested all image types in our data set and computed their hashes. We then compared these distances to perceptually similar images to determine the correct threshold values. Getting this value correct allows our tool to match images through minor distortions without creating false positives. Through testing, our team landed on a threshold of 5 that reliably matches images even after file compression while maintaining accurate matching.

The final output of the image hashing process is a matrix that covers all shifts across the six-week scheduling period. A CSV file is created that houses this binary matrix in the same fashion as the previously discussed self-scheduling data, with a separate file for each image. Indexing operates the same way as the self-scheduling matrix but includes an extra step to locate the image. Image location involves using a function from openpyxl to obtain the top-left cell the image was anchored to, then using the previously discussed indexing logic to locate that cell relative to the start of the shift matrix. The logic then operates as before, assigning one to both the day and night shifts if an image is present and zero otherwise.

3. Solution Generation

To identify high-performing solutions, our team developed an Integer Linear Program (ILP) that embeds the various internal policies and contract restrictions that BMHCC considers when scheduling nurses. This model includes several hard constraints, such as meeting acceptable staffing levels and scheduling employees to shifts they can work, that define the restrictions a feasible solution must adhere to. To identify optimal solution(s), this model uses a multi-objective approach that targets six main metrics: shifts at the minimum number of RNs, the total number of overtime occurrences, the lowest total experience across all shifts, the total number of times pro re nata (PRN) nurses are scheduled, the number of times nurse assignments on the unbalanced schedule are maintained, and the number of times that a nurse was not given a shift for which they said they were available.

ISBN: 97819384962-5-7

3.1 Integer Linear Program

Sets:

- N – All nurses working on a floor {1, 2, …, N}
- S – All 12-hour shifts contained within a 6-week period {1, 2, …, 84}
- W – All week indexes contained in the scheduling period {1, 2, …, 6}

Parameters:

- L – The minimum number of RNs that should be on each shift
- U – The maximum number of RNs that should be on each shift
- R_n – 1 if nurse n is an RN. 0 otherwise
- H_n – The number of shifts per week nurse n is expected to work
- E_n – The years of experience possessed by nurse n
- P_n – 1 if nurse n is a PRN. 0 otherwise
- D_n – 1 if nurse n can work days. 0 otherwise
- M_n – 1 if nurse n can work nights. 0 otherwise
- K_n – 1 if nurse n is contracted as a weekender. 0 otherwise
- Y_s – 1 if shift s occurs during the day. 0 otherwise
- W_s – 1 if shift s occurs during a weekend. 0 otherwise
- $A_{n,s}$ – 1 if nurse n has marked themselves as available for shift s. 0 otherwise
- $G_{n,s}$ – 1 if nurse n was scheduled to work shift s in the unbalanced schedule. 0 otherwise
- $O_{n,s}$ – 1 if nurse n has approved time off for shift s. 0 otherwise
- $C_{n,w}$ – The maximum number of available shifts nurse n could work in week w
- $\omega_{m \in 1,2,\dots,6}$ – Normalized weighting factor assigned to evaluation metric m:

Evaluation Metric	(1) Minimum RNs	(2) Overtime	(3) Min Experience	(4) PRN Use	(5) Original Assignments	(6) Wasted Availability
ω Value	10,000	1,000	5	1	0.2	0.03

Decision Variable:

- $X_{n,s} \in \{0,1\}$ – 1 if nurse n is scheduled to work shift s. 0 otherwise

Evaluation Variables:

- $MRN_s \in \{0,1\}$ – 1 if shift s is at the minimum number of RNs. 0 otherwise
- $OT_{n,p \in 0,1,2} \in \mathbb{N}$ – The number of overtime shifts nurse n is scheduled to work in pay period p.
- $MTE_{b \in 1,2,\dots,10} \geq 0$ – The years of experience in excess of $10 * (b-1)$ for the least experienced shift in the schedule
- $OA_{n,s} \in \{0,1\}$ – 1 if nurse n is scheduled for their originally assigned shift s. 0 otherwise
- $WA_{n,w,i \in 1,2,3} \in \{0,1\}$ – 1 if nurse n has i shifts in week w marked available that they're not scheduled to work. 0 otherwise

Model Formulation:

$$\text{Max} \quad -\omega_1 \sum_{s \in S} MRN_s - \omega_2 \sum_{n \in N} \sum_{p=0}^{2} OT_{n,p} + \omega_3 \left(\sum_{b=1}^{5} \frac{1}{i} * MTE_b + \sum_{b=6}^{10} \frac{1}{i^{4/3}} * MTE_b \right) - \omega_4 \sum_{n \in N} \sum_{s \in S} X_{n,s} * P_n + \omega_5 \sum_{n \in N} \sum_{s \in S} OA_{n,s} - \omega_6 \sum_{n \in N} \sum_{w \in W} \sum_{i=1}^{3} i^2 * WA_{n,w,i} \tag{1}$$

(1) The objective function maximizes overall schedule value by weighting evaluation metrics according to their importance to BMHCC's management. The weights assigned to the MRN and OT metrics are set arbitrarily high to ensure solutions always prioritize them over all others. The remaining evaluation metrics are assigned smaller weights to normalize and adjust their values, encouraging the model to generate solutions that yield balanced performance. The sum indexes for MTE and WA are used to scale weights non-linearly, providing diminishing returns for experience and exponential punishments for repeatedly failing to give a nurse their available shifts.

Subject To:

$L \leq \sum_{n \in N} X_{n,s} * R_n \leq U$	$\forall_s$	The number of RNs on each shift must be between the lower and upper bounds.	(2)
$X_{n,s} + X_{n,s+1} \leq 1$	$\forall_{n,s=1,\ldots,83}$	Nurses cannot be scheduled for two consecutive shifts.	(3)
$X_{n,s} \leq (1 - O_{n,s})$	$\forall_{n,s}$	Nurses cannot work shifts that they have approved off.	(4)
$\sum_{s=14*(w-1)+1}^{14*w} X_{n,s} * (1 - P_n) = H_n$	$\forall_{n,w}$	FTE and PTE nurses must be given their expected number of shifts each week.	(5)
$\sum_{s \in S} X_{n,s} * P_n \geq 3 * P_n$	$\forall_n$	PRN nurses must work at least three shifts during each six-week period.	(6)
$X_{n,s} * Y_s \leq D_n$	$\forall_{n,s}$	Day shifts must be worked by day shift nurses.	(7)
$X_{n,s} * (1 - Y_s) \leq M_n$	$\forall_{n,s}$	Night shifts must be worked by night shift nurses.	(8)
$\sum_{s=14*(w-1)+1}^{14*w} X_{n,s} * W_s \geq 2 * K_n$	$\forall_{n,w}$	Weekender nurses must work at least two weekend shifts every week.	(9)
$\sum_{n \in N} X_{n,s} * R_n \geq L + (1 - MRN_s)$	$\forall_s$	MRN_s must take a value of one if the shift is at the minimum number of RNs.	(10)
$\sum_{s=(28*p)+1}^{28*(p+1)} X_{n,s} \leq 8 + OT_{n,p}$	$\forall_{n,p=0,1,2}$	$OT_{n,p}$ must absorb any shifts in excess of eight that a nurse works in each two-week pay period.	(11)
$\sum_{n \in N} X_{n,s} * E_n \geq \sum_{b=1}^{10} MTE_b$	$\forall_s$	The experience included in all MTE buckets must be below the total experience of the least experienced shift.	(12)
$MTE_b \leq 10$	$\forall_{b=1,2,\ldots,9}$	The first nine MTE buckets can contain at most ten years.	(13)
$X_{n,s} * G_{n,s} \geq OA_{n,s}$	$\forall_{n,s}$	$OA_{n,s}$ can only take a value of one if the nurse is assigned to a shift they had in the unbalanced schedule.	(14)
$\sum_{s=14*(w-1)+1}^{14*w} X_{n,s} * A_{n,s} + \sum_{i=1}^{3} i * WA_{n,w,i} \geq C_{n,w}$	$\forall_{n,w}$	WA must account for the difference between the number of available shifts a nurse is given in a week and the number they could have worked.	(15)
$\sum_{i=1}^{3} WA_{n,w,i} \leq 1$	$\forall_{n,w}$	At most one wasted availability category per nurse per week.	(16)

3.2 Heuristic Approach

Due to the complexity of our model and the extensive use of binary variables, the runtimes required by traditional optimization solvers don't scale well enough to justify their use within a production environment. After loading data for ~35 nurses, runtimes exceed the target 1-to-2-minute window provided by BMHCC, a trend that continues as more nurses are added. In addition, using a high-powered optimization solver incurs the cost of purchasing and maintaining a license, an expenditure BMHCC prefers to avoid if possible.

To remedy these issues, we developed a two-stage heuristic algorithm implemented in Python. Our heuristic is supported by three user-defined classes: nurses (used to store the data described in section 2), shifts (used to model 12-hour

scheduling blocks), and final 6-week schedules. These classes are linked via multiple composition relationships, with 6-week schedules storing an array of individual shifts, containing a list of nurses representing staff assignments. These classes also contain logic to validate schedule feasibility, compute schedule performance based on our ILP's objective function, and create feasible schedules by reading nurses from a given roster and adding them one at a time to the shifts with the greatest need.

Due to the short-sighted nature of the 6-week schedule's construction method, only adding a single nurse at a time, the performance of generated solutions tends to vary heavily based on the order in which nurses are added. To leverage this variability to identify high-performing schedules, we use a simulated annealing-based heuristic as the first phase of our solution-generation process, creating an initial population of solutions by improving roster ordering. For each individual in the initial population, the first phase completely randomizes the nurse insertion order, then repeatedly updates the nurse ordering by swapping random indexes. After updating the ordering, the algorithm then creates a new schedule using the roster ordering and evaluates its performance, accepting the new insertion order if the new schedule has a higher objective value than the current incumbent or randomly based on how much the objective is decreased by using the new ordering and the heuristic's internal temperature. These phases then repeat for a predefined number of iterations, with the temperature (and thus the probability of accepting a worse solution) decreasing with each iteration.

After the initial population is created using the simulated annealing algorithm, the heuristic proceeds to the secondary improvement phase, implemented through a modified genetic algorithm based on the principles described in "A genetic algorithm tutorial" (Whitley, 1994). The first step in this process involves culling poor-performing schedules by randomly assigning them to tournament pools, evaluating each schedule's objective value, and deleting all but the best-performing schedule in each pool. After the population has been culled, the algorithm randomly selects two surviving schedules and builds offspring using a semi-random inheritance process that draws individual nurse assignments from both parents. After the offspring has inherited nurse assignments for each shift from its parents, it is then mutated by randomly swapping nurses across shifts that occur during the same week, ensuring that feasibility is maintained. The crossbreeding and mutation phases are repeated several times, using different parents to rebuild the population to its initial size, after which the algorithm cycles and runs the culling process again.

Once the genetic algorithm reaches its maximum number of generations, the program returns the best schedule found across all generations and uses it as the final recommended solution. Due to the use of heuristics, this solution is not guaranteed to be optimal; however, after comparing the performance of schedules output by the heuristic to the best bounds obtained using the CBC optimization solver and 2-minute runtimes, we found that our heuristic's solutions, on average, have a MIP gap of ~5-10% compared to CBC's 20-30%. Given the improvement over BMHCC's current process, the time-saving benefits of the heuristic, and the ability to avoid licensing a traditional optimization solver, we found this tolerance more than justified within the scope of our project.

4. User Interface

Our team's goal was to develop an end-to-end solution that guides nurse managers through the entire schedule-balancing process from data input to post-processing. To support this workflow, our team created a Python-based user interface that guides users through data loading, solution generation, and schedule refinement while providing visibility into schedule quality and feasibility. From a technical perspective, this interface serves as a thin presentation layer that orchestrates the underlying code for data ingestion, heuristic solution generation, and schedule evaluation. User actions such as uploading a schedule, converting data, generating a solution, and applying edits invoke backend functions that transform inputs into model-ready matrices, run the solution pipeline, and compute feasibility and key performance indicator (KPI) summaries for reporting.

The interface is implemented using Streamlit, a lightweight Python framework for building interactive web applications without requiring separate front-end development. Streamlit manages widget state and triggers application reruns when inputs change, allowing the interface to present updated schedules and performance measures immediately after data conversion, schedule generation, or post-processing edits. The system also uses the pandas library to represent and transform tabular data throughout the workflow. Hospital exports, converted parameter tables, schedule matrices, and KPI summaries are each stored as pandas DataFrames, enabling consistent indexing by nurse identifier and shift index, as well as efficient aggregation and validation operations. The Openpyxl library is used to ingest Excel-based exports produced by the scheduling system. This allows the interface to read and preview the raw schedule file and supports the conversion process that maps spreadsheet content into the structured inputs required by the solution engine. The final Python library used to support our interface's back end is NumPy, which supports the numerical computations required to evaluate and manipulate schedule matrices efficiently. This package enables compact representation of nurse-by-shift assignments and provides fast aggregation operations that support feasibility checks and KPI calculations across the full six-week horizon.

The user interface is designed around BMHCC's existing scheduling workflow in which nurse managers export a six-week report mainly denoting availability from their scheduling system. The interface ingests this export and converts it into the

parameter tables and binary matrices required by the solution-generation code. During conversion, the interface performs validation checks and summarizes key parsing results to ensure that availability and approved time off are interpreted correctly before generating a solution. After successful conversion, the interface triggers the heuristic solution pipeline to produce a recommended schedule. Outputs are presented as a schedule matrix supported by summary information including staffing coverage, constraint violations, and key performance indicators (KPIs) aligned with the objective components used to evaluate schedule quality. This reporting allows nurse managers to quickly identify where the recommendation meets policy requirements and where trade-offs were made. In addition to producing a recommended schedule, the interface supports post-processing edits, allowing managers to incorporate operational context not captured in the input data. The interface updates KPI totals as changes are made and provides optional hard-constraint feasibility checks to reduce the risk of introducing invalid assignments. Once acceptable changes have been applied, the interface exports the updated schedule for downstream use.

5. Results

To evaluate the effectiveness of our tool, we benchmarked its outputs against several of BMHCC's previously deployed schedules. Using absolute distance from seven RNs scheduled on a shift, our tools' schedules average 0.934, while previously implemented schedules average 1.988. This metric is calculated by taking the weighted absolute deviation of each shift from the target of seven RNs, then averaging these values across all floors. Lower values indicate that staffing levels are, on average, closer to the target of seven RNs per shift, achieving BMHCC's primary goal of reducing staff-level variance by approximately 53%. While traditional optimization solvers take upwards of 3 hours to solve our program, the heuristic offers solutions far faster and provides greater flexibility by allowing the number of iterations and/or schedules evaluated at each stage to be adjusted, thereby augmenting the runtime and solution quality. In addition to decreasing solve times, our tool also greatly reduces the time nurse managers spend scheduling. While our tool has not been used during a live scheduling period, we estimate that it will save 10-15 minutes on initial scheduling and balancing for even the most experienced nurse managers, allowing them to spend more time on administrative responsibilities. These time savings are found not only in the initial scheduling of employees who do not self-schedule, but also through the immediate visibility of under-performing shifts. Overall, our tool quickly provides BMHCC with near-optimal schedules while also improving visibility and helping to minimize human error.

6. Discussion

From the beginning of our project, we aimed to provide more than just a deliverable that meets our university's capstone requirements. While we were tasked with improving the scheduling-balancing process at the Desoto hospital, we wanted to create a product that streamlined it enough to warrant implementation across their entire network.

One way that we made this product stand out was through extensive documentation. Since BMHCC's workforce optimization team would be more likely to use our tool if they understood it well and could easily modify it, we first created an outline to help employees find specific parts of the code. If a manager wanted to update something like one of the model's constraints, we created detailed documentation showing the constraint's location, as well as its underlying logic. In the same vein, we also created documentation on how to manipulate the dynamic items of our product. Understanding that technologies and procedures within companies change frequently, we wanted our product to be as adaptable as possible to the changes that may come to BMHCC's operations. This documentation enables our stakeholders to manage the product's components as technical and strategic details evolve. These components include the number of replications the model runs when generating solutions, the set of images that our product hashes to parameterize the model, column names from the input data, and more. For example, if a column header were to change in the unbalanced schedule output from Shift Wizard and an error was thrown, we wanted managers to easily know how to adjust our product so it would work correctly again. With proper documentation in place, these adjustments can be made in a matter of minutes rather than leaving our stakeholders stuck trying to decipher unfamiliar code, ensuring that our product can be used in the long term.

7. References

Lasater, K. B., & Aiken, L. H. (2026). Hospital nurse understaffing persists with negative consequences. *JAMA Network Open*, *9*(2). https://doi.org/10.1001/jamanetworkopen.2025.58188

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and Computing, 4*(2), 65 - 84. Retrieved from https://doi.org/10.1007/BF00175354