TreatmentFlow:
Finding the optimal patient triage system for hospital usage

Emese Elkind and Adam Neto

Abstract

The average wait time in Ontario emergency departments is 2.2 hours. Triage is the sorting mechanism that hospitals use to see which patient they need to care for first, ensuring that patients needing urgent care wait the least amount of time for treatment. In Ontario, patients are assigned a priority number from 1-5 based on urgency. Triage nursing is known to be one of the most stressful nursing positions. To address this, we propose *TreatmentFlow*, an AI-based digital solution for optimizing the flow of patients through the emergency department. We examine three problems in triage and patient flow and apply AI techniques including Deep Learning, Constraint Optimization and Bayesian Networks to solve them.

Abstract is not finished yet!

I. Introduction

HEALTHCARE is constantly evolving. With emergency room wait times averaging 2.2 hours in Ontario [1], Artificial-Intelligence-driven triage systems could be the key to prioritizing patients more efficiently and reducing delays. Triage is the sorting mechanism that hospitals use to determine the urgency to treat each patient with. This is determined by the patient's conditions and symptoms when they enter the hospital. Triage ensures that patients needing urgent care wait the least amount of time for treatment. In Ontario, patients are assigned a Canadian Triage and Acuity Scale (CTAS) priority number from 1-5 based on urgency of care (Table 1):

Table 1: Canadian Triage and Acuity Scale (CTAS) priority leveling [6].

CTAS Levels	Description
1	Resuscitation
2	Emergent
3	Urgent
4	Less Urgent
5	Non-Urgent

Triage nurses check vital signs and symptoms and gather patient history data to assess where they fit on this priority list. This is known to be one of the most stressful positions a nurse can have in the hospital [2]. Technology is increasingly being used in hospitals to assist with triage and relieve some of this stress on nurses with artificial intelligence (AI) and machine learning (ML) becoming novel topics for triage.

We propose *TreatmentFlow*, an AI-based algorithm for optimizing the flow of patients through the emergency room. We examine three problems in triage and apply AI techniques to solve them. Firstly, we aim to automate the assignment of patient priority using triage symptom information, relieving some triage decision-making pressure for nurses. The method presented to solve this is a Deep Learning approach. In this approach, we use an Artificial Neural Network model to learn from data to be able to predict the CTAS priority level of incoming patients given symptom data.

Next, we seek to optimally assign patients to beds based on patient priority, further simplifying the decision-making tasks of the nurses. This is approached with two different methods: mixed integer programming (MIP) constraint optimization, and greedy patient assignment. In the MIP approach, we construct a model of variables and constraints to mimic a hospital and optimize (minimize) the total wait time of patients, weighted by priority level. This weighting implies that the model considers that the wait times of high-priority patients are more detrimental than the wait times of lower-priority patients. To test the value of this MIP solution, we compare its results to those of a greedy patient assignment algorithm. This comparison evaluates either solution's performance and efficiency at scale, including objective (sum of weighted wait times) minimization, and execution time.

Doctors in the emergency department often ask for each other's opinions to ensure correct diagnosis. Double-checking opinions is important but can sometimes waste time and be ineffective. We propose a more efficient solution: to predict possible diagnoses and produce a doctors-only bedside document that includes the probability of them, thus aiding in the efficient, correct diagnosis of patients. By providing the probabilities of patient conditions, doctors will be able to more efficiently deliberate between methods of treatment. Our solution involves utilizing probabilistic inference and Bayesian Networks to produce each patient a probability table to display potential conditions given a set of triage symptoms.

II. Methodology

Data sets containing patient symptom information were used to train *TreatmentFlow*. *Hospital Triage and Patient History Data* and *SymbiPredict* are the data sets which were found on Kaggle [3] and Mendeley [4] respectively. *Hospital Triage and Patient History Data* is data found in a study of admission and discharge data found at Yale New Haven Health system between March 2014 and July 2017; it contains 972 variables, including patient information and symptoms and about 50,000 rows of patients. This data was cleaned and processed by reducing the size of the file from 103.45

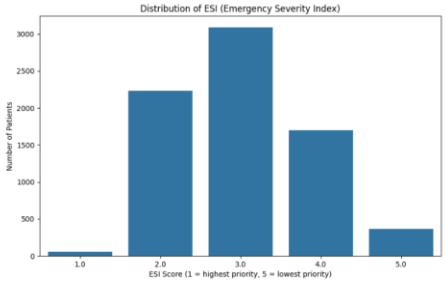


Figure 1: Distribution of Emergency Severity Index in new_hospital_data_75.csv.

MB to 19.8 MB by converting the rdata file into a csv file and only taking every 75th row, to ensure some level of random sampling while maintaining a manageable dataset size for analysis. The *SymbiPredict* file is a csv file containing almost 5,000 rows of symptoms and the given prognosis. This file was not processed and did not need to be cleaned since it is only 1.32 MB and already in the csv format.

A. Constraint Satisfaction

The Constraint Satisfaction component of *TreatmentFlow* is a set of solutions for hospital bed assignment based on incoming patients and available resources. The two solutions are (1) a mixed integer programming (MIP) constraint optimization solution and (2) a greedy bed assignment solution. These solutions are compared in terms of efficiency and results on a scale.

Both solutions rely on a system of hospital objects formed in the file "HospitalClasses.py" these necessary objects include Bed, Patient, and HospitalRecords, each of which store and track information for instances of their category. For example, each Bed object stores their occupant (if any), each Patient stores statistics like CTAS priority level and arrival time, and HospitalRecords stores a database of patients. In our first iteration of *TreatmentFlow's* constraint satisfaction component, we track the *inverse* of CTAS priorities, meaning we treat 5 as the highest priority and 1 as the lowest to simplify current calculations. We are also using a randomly generated set of patients, arrival times, and priorities, whereas we will chain all three components together for a given set of patients in our final product.

In the greedy solution to hospital bed assignment found in the "GreedyBedAssignment.py" file, we examine the current state of the hospital minute-by-minute over a 24-hour period of with the assistance of custom HospitalRecords attributes such as "unserved" (the priority queue of unserved patients) and "serving" (the running list of patients occupying a bed). For each minute of the simulation, the scheduler assigns the next available highest-priority patient to any available bed and keeps them there for the duration of their service time. This assignment considers the patient's CTAS priority level (inverse) and arrival time at the hospital. Given the number of patients (p) and number of beds (b), the hospital simulator method "run_hospital()" features a time complexity of O(p * b), which is very efficient at scale. While this approach is algorithmically efficient, it is naïve in the fact that it does not guarantee an optimal solution but instead assigns patients based on currently available information.

Constraint optimization problems involve modeling a set of variables and assigning them values such that all constraints on variables are satisfied. They then use this model to optimize for a given objective function, which is either minimized or maximized. In the mixed integer programming (MIP) approach to hospital bed assignment found in the "MIPBedAssignment.py" file, we build a constraint satisfaction/optimization model using Google OR-Tools' cp_model (Constraint Programming model) class. This library allows for the construction of constraint satisfaction models and provides a built-in solver that optimizes for a given objective function. Our model includes variables for each bed at each minute of the day, each of which can be set to any one of the patient identifications from the list of patients. The model builds several constraints

concerning each patient: the arrival time constraint ensures that each patient cannot be assigned a bed before they arrive at the hospital; the contiguity constraint ensures that a patient does not shift between beds, and implicitly ensures that a patient can only be assigned to one bed at a time; and the tenure constraints ensure that a patient is not assigned to a bed for any shorter or longer than their service time, unless their service time extends past midnight (in which case they are assigned until midnight). It is noteworthy that the greedy assignment solution also satisfies all the same constraints. Finally, the MIP solution forms an objective function that minimizes the sum of the wait time of each patient (multiplied by inverse of CTAS priority level, which essentially translates to "patient cost"), applying a penalty of patient cost * total minutes per day (1440 min) to patients that are not assigned. When the solver is run, it finds solutions that satisfy all constraints, minimizing the objective function, and thus optimizing the bed assignment. We have also included user-entered parameters to set the number of beds, patients, and time limit of the MIP solver's execution. Note that if an optimal solution is not found before the time limit runs out, the solver displays the most optimal solution found so far.

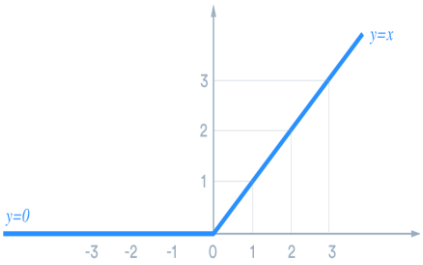
To test the value and validity of the MIP and Greedy solutions to bed assignment, we repeatedly compared the results of the same set of patients and beds. Across various scales, we performed this comparison with a new, random set of patients and beds at each denomination. The results are displayed in the next section of this report.

B. Deep Learning

To automate the assignment of patient priority using triage symptom information to relieve the pressure for nurses making wrong decisions a Deep Learning model assigns patient priority through this automated triage system.

Supervised machine learning has been chosen as the machine learning technique for this task because it uses labeled data to train algorithms to predict outcomes. The model is given labeled training to learn the relationship between the input and the outputs. The two main types of supervised machine learning are classification and regression [5]. Classification was chosen for this task to perform multi-class classification task, where the model predicts which of 5 possible priority levels (1-5) a patient should be assigned [6]. The output is the priority levels (1-5) which is discrete therefore the final layer of the algorithm will have 5 neurons. Common classification algorithms are linear classifiers, support vector machines (SVM), decision trees, k-nearest neighbor and random forest [5], Multi-layer Perceptron (MLP) classifier was chosen as a model for this dataset because it can handle mixed data types and is scalable for large datasets.

Hospital Triage and Patient History Data is used for the deep learning aspect of *TreatmentFlow*, with this data loaded into the file and preprocessed. The data is first observed using the load_data function, and Pandas [7] is used to observe the file format. A graph is made to observe the given CTAS scores, and a bell curve with level 3 being the most common priority level is shown. The data is preprocessed by dropping columns from the data set that are non-



medical demographic data, as to not contribute to any data leakage. The data in the *Hospital Triage and Patient History Data* csv file presents itself with numerical and categorical features which is split using Numpy [8] to ensure that the model is trained on high-quality data. Data is split into 80% training data and 20% test data sets with the input represented by 'x' and are the columns of the file and the target output represented by 'y' which are the CTAS values. The MLP classification algorithm that is being used is a Deep Neural Network (DNN), and is implemented using Scikit Learn [10] with 3 hidden layers. To determine whether a neuron is activated or not the ReLU activation function is used for the hidden layers. The ReLU activation function is the most common in deep learning and is a mathematical function to introduce non-linearity. If the input is positive the output is positive and if the input is negative the output is zero.

The model is evaluated using the testing data set (20% of the whole data set) which is saved in a confusion matrix to visualize the results.

C. Bayesian Networks

Bayesian Network-based diagnostic tools to produce bedside documents for doctors only, which display the probabilities of certain conditions based on symptom inputs

III. Results

A. Constraint Satisfaction

Table 1: Ratio of objective values produced by the MIP approach and Greedy approach. For each cell, a ratio lower than zero means the MIP approach produced a better solution than the Greedy Approach within the 120 second time limit on the solver. A cell value of "None" means the MIP solver produced no solution. (x axis: Patients, y axis: Beds)

	5	**10**	**20**	**50**	**100**	**250**
5	1	1	1	2.6045	0.729	None
10	1	1	1	24344	None	None
15	1	1	1	inf	None	None
20	1	1	0	84078	None	None

Table 2: Ratio of execution times of MIP approach (with 120 second time limit on solver) and Greedy approach. (x axis: Patients, y axis: Beds)

	5	10	20	50	100	250
5	29015	33523	17991	58924	26433	18235
10	21085	59913	57062	100775	62713	24673
15	52333	67038	80641	111624	81349	34696
20	68041	88934	115251	119926	61192	45063

Table 3: Greedy approach's patients' average wait time in minutes, and average number of unserved patients over 100 attempts. (x axis: Patients, y axis: Beds)

	5	10	20	50	100	250	500
5	0, 0	0, 0	4, 0	51, 6	39, 66	33, 221	24, 471
10	0, 0	0, 0	0, 0	0, 0	40, 11	24, 188	33, 441
25	0, 0	0, 0	0, 0	0, 0	0, 0	30, 21	24, 345
50	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	24, 34
100	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0	0, 0

B. Deep Learning

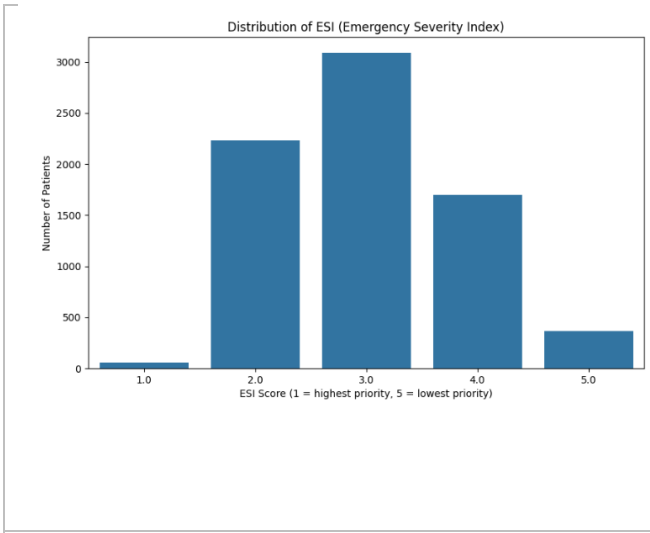


Figure 1: Distribution of CTAS values in new_hospital_data_75.csv.

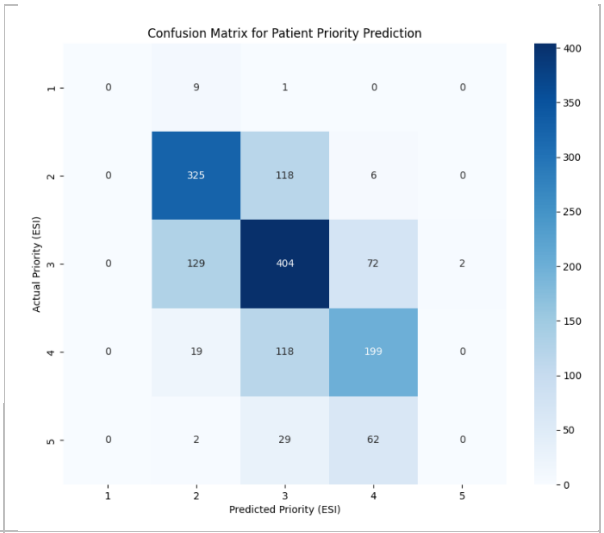


Figure 2: Confusion Matrix to analyze DNN model predicting CTAS values from new_hospital_data_75.csv.

C. Bayesian Networks

IV. Discussion

A. Constraint Satisfaction

As seen in the comparison, tables 1 and 3 in the results section above, both the MIP and Greedy approaches can (usually) successfully reduce patient wait times. At a small scale (of patients and beds), the two methods return very similar results in terms of their objective values (Table 1) but differ greatly in their execution time (Table 3). The execution times of both methods increases with the problem size, but the execution time of the greedy solution remains under a single second, while the execution time of the MIP solution often takes multiple minutes. This is because building an MIP model with more variables leads to the creation of significantly more constraints, since they are dependent on the relationships between variables. For the results above, it is important to note that the solving time of each MIP model was limited to a maximum of 120 seconds. Without this limitation, testing each scale until proving optimality would become infeasible, as the default (unlimited) solving time increases exponentially, and thus the execution time ratios would be far greater. This increase of both approaches' execution time makes the ratio increase with the problem size, but it is limited by the 120 second time limit on the largest problems, which explains the reduction on the later columns of table 2.

MIP solving is notably faster when the number of patients is close to or lower than the number of beds, as the increased solution space allows for more feasible optimal assignments. This is because there are more possible solutions that allow an optimal assignment of patients to beds. This chance for optimality comes from the fact that there is far more space at each individual minute, meaning that each patient is more likely to find an empty bed at their arrival time, and thus the MIP solver finds an optimal solution much faster. However, as the MIP solver more easily finds an optimal solution, the Greedy approach also has fewer calculations to perform, which stabilizes the ratio of either approach's execution time in table 3.

When comparing each solution's performance in terms of their objective values, the two solutions often perform similarly, with some randomization involved. This randomization of relative performance depends on the layout of patient arrival times in each randomized list of patients. As discussed above, the MIP solver finds optimal solutions much more often when the number of patients is near the number of beds, or equal, or lower. However, the greedy solution provides near optimal solutions in these scenarios as well in a fraction of the time. As shown in table 1, the Greedy approach produced an equal or superior objective value in all but one observation. This shows that as problem size increases, the computational inefficiency of the MIP solution begins to outweigh the potential improvements it has in bed assignment optimization. This makes it impractical for large-scale settings, and thus impractical for most hospitals.

To conclude the comparison, the MIP model can sometimes find better solutions than the Greedy method, but the drastically larger computational complexity of the model makes it infeasible at scale. Since the differences between solutions found are marginal when limiting the solver's execution time, the Greedy bed assignment method, despite being a naïve solution, remains the superior bed assignment method. Using the Greedy bed assignment tool, we can reduce the logistical delays of patient wait times to negligible values, assuming a realistic number of patients compared to capacity (Table 3). The functional limit for the ratio of patients to beds that allows this tool to minimize wait times is when the ratio is less than ten. Since this is a very extreme potential ratio of patients to hospital resources, this tool remains very useful, even at scale. Thus, this application of constraint satisfaction creates a significant benefit for the operation of emergency departments.

B. Deep Learning

The confusion matrix (Figure 2) is a table that is used to define the performance of a classification algorithm by comparing its predicted labels to the actual labels [9]. The y-axis of the matrix represents the true class, which are the actual labels, and the x-axis represents the predicted class, which are the model's output. The diagonal values from (1,1) to (5,5) represent the correct predictions, where the model's predicted priority matches the actual priority. The higher these values and the darker the coloring of the box, the better the model's accuracy. The values off the diagonal represent misclassifications, where the model predicted the wrong priority level. If a value is above the diagonal, it means the model underestimated priority. If a value is below the diagonal, it means the model overestimated priority. The model accuracy is 0.62 (62%) when trained on new_hospital_data_75.csv. The model seems to have the most difficulty distinguishing between priority 1 and 2, as well as 2 and 3, based on the number of misclassifications between them. There are very few cases in priority 0 and 4, which might indicate class imbalance. A good DNN model accuracy is generally considered to be above 70% according to industry standards, with anything between 70% and 90% being considered a realistic and valuable performance range. The *TreatmentFlow* model is not up to industry standards yet, therefore, future work includes finding the optimal hidden layer nodes to maximize model accuracy and thus patient priority levels.

C. Bayesian Networks

V. Conclusion

AI-driven triage systems like *TreatmentFlow* have the potential to revolutionize emergency room efficiency by addressing key challenges in patient prioritization, bed assignment, and diagnostic support. By leveraging Deep Learning for automated triage, Constraint Optimization for bed allocation, and Bayesian Networks for diagnostic assistance, *TreatmentFlow* aims to reduce emergency wait times, alleviate stress on healthcare workers, and improve patient outcomes. As hospitals continue to integrate AI technologies, solutions like this can play a critical role in optimizing resource management and ensuring timely, effective care for all patients.

Future work includes connecting the three parts of the project by having the output of one part being the input of the previous; specifically the CTAS priority levels predicted in the Deep Learning section passed into the Constraint Optimization section to optimally assign these patients to beds and once the beds are assigned, it will be passed into the Bayesian Network section to predict diagnoses for a doctor-aimed bedside document.

References

[1] *System performance*. Emergency Department Time Spent by Patients in Ontario – Health Quality Ontario (HQO). (n.d.). https://www.hqontario.ca/system-performance/time-spent-in-emergency-departments

[2] Emergency room triage: How does it work? - the daily scan. (n.d.). https://thedailyscan.providencehealthcare.org/2018/11/emergency-room-triage-how-does-it-work/

[3] Mafaufau, M. (2019, June 3). *Hospital triage and patient history data*. Kaggle. https://www.kaggle.com/datasets/maalona/hospital-triage-and-patient-history-data

[4] Tucker, J. (2024, April 2). *Symbipredict*. Mendeley Data. https://data.mendeley.com/datasets/dv5z3v2xyd/1

[5] Belcic, I., & Stryker, C. (2024, December 26). *What is supervised learning?*. IBM. https://www.ibm.com/think/topics/supervised-learning#:~:text=its%20training%20data.-,Types%20of%20supervised%20learning,next%20layer%20in%20the%20network.

[6] *Emscimprovement*. Emergency Nurses Association. (2002). https://media.emscimprovement.center/documents/Emergency_Severity_Index_Handbook.pdf

[7] *Pandas*. pandas. (n.d.). https://pandas.pydata.org/

[8] NumPy. (n.d.). https://numpy.org/

[9] *Confusion matrix*. Confusion Matrix - an overview | ScienceDirect Topics. (n.d.). https://www.sciencedirect.com/topics/engineering/confusion-matrix#:~:text=A%20confusion%20matrix%20is%20a,malignant%20tissue%20is%20considered%20cancerous.

[10] https://scikit-learn.org/stable/