Optimization of Patient Flow in Emergency Departments using Genetic Algorithms

A PROJECT WORK

Submitted in fulfillment of the award of Degree of Master of Science in Computer Science

Submitted

By

SIVA SHANMUKHA HEMANTH KASYAP AGASTYARAJU

UNDER THE SUPERVISION OF

Dr. Shahram Rahimi

Committee Members

Dr. Bidyut Gupta, Dr. Christos Mousas

Department of Computer Science
Southern Illinois University Carbondale
December 2017

INDEX

ABSTRACT

The common problem faced by most of the Emergency Departments (ED) in the United States is overcrowding of patients. To address this issue, the project demonstrates a Genetic Algorithm to minimize the waiting time of patients by optimally allocating a set of service providers at each stage. Based on the concepts of natural genetics and natural selection theories proposed by Charles Darwin, Genetic Algorithm is a popular optimization technique. An emergency department consists of a set of stages where patients wait for their treatment. Each stage consists of a set of service providers such as Registration, Nurse, Physician and other staff. To achieve optimization, the Genetic Algorithm uses a fitness function. The input of this function is the set of service providers required at each stage and the output contains the average waiting time of the patient.

CHAPTER 1
INTRODUCTION

1.1 Motivation

This project "Towards Patient Flow Optimization in Emergency Departments Using Genetic Algorithms" aims to utilize a Genetic Algorithm and a simulation model for the optimal allocation of service providers in emergency departments to improve patient flow by reducing the length of their stay, while minimizing health care delivery costs.

A **Genetic Algorithm** is a heuristic search method used in artificial intelligence and computing. It is used for finding optimized solutions to search problems based on the theory of natural selection and evolutionary biology. **Genetic Algorithms** are excellent for searching through large and complex data sets.

We propose a model in which first a JSON (**JavaScript Object Notation)** string is given as the input that contains the name of the stage the number of minimum and maximum providers required at each shift of the stage. JSON is a minimal, readable format for structuring data. It is used primarily to transfer data between a web server and web application, as an alternative to XML. This JSON string is then deserialized to obtain the values of minimum and maximum providers. These values are given as inputs to the genetic algorithm which calls the simulation API.

1.2 Problem Definition

The long waiting time of patients in the emergency departments is due to lack of sufficient resources, poor quality and overcrowding. This leads to increased mortality rates, lower patient satisfaction and increased costs. In addition to this, emergency departments must ensure the delivery of effective, timely and safe patient care while keeping the costs under control. The ability to predict the patient overload is very difficult in emergency departments due to variable resources such as nurses, unforeseen patient demand level and relatively fixed physical capacity.

To overcome this, our project is based on a Genetic Algorithm that creates a population which contains a set of chromosomes. Each chromosome consists of a set of values and each of these values is called a gene. This gene is formed by randomly selecting a number from the range of minimum and maximum providers that are to be allocated at each shift of a stage. Hence, the chromosome represents the number of service providers required at every stage. These set of values are sent to the server as an input to the fitness function. The fitness function then returns the length of stay of the patient i.e. the amount of time for which the patient stays in the emergency department. The goal is to minimize the length of stay of the patients alongside optimally allocating service providers at all stages.

1.3 Objective of Project

For finding a global maxima or minima, Genetic Algorithms present a general approach within a bounded search space. GA are widely used in optimization problems as they only need a way to evaluate the performance of its solution guesses without any prior information. The final solution of a GA odes not converge at a single optimal solution, but will provide, on average a good solution. GA is usually extensively modified to suit an application.

To obtain the length of stay, considering various factors that might happen in the emergency departments such as availability of resources, flow schemes, admission, patient routing and scheduling, and to model the real-time behavior of the system, a Discrete Event Simulation (DES) Model has already been developed and maintained at a server.

CHAPTER 2
LITERATURE SURVEY

2.1 Discrete Event Simulation (DES)

To picturize and quantitatively examine the performance of a process, DES model is used. It is a computer model that simulates the dynamic behavior of a real process. Then in different scenarios the DES simulation can be run to validate and improve the model to make the simulation behave as comparative as conceivable to the real system. The simulation lets the user to try any changes they want on the system and check the results before applying them on the real system. So, making decisions and finding solutions for the real system is possible with no cost.

The information needed for the simulation is as follows:

- Arrival time and the quantity of the entities. There is no limitation such as Poisson distribution on the arrival time.
- Service time in each stage which has no limitation such as exponential distribution.
- The capacity in each stage or the maximum number of entities.
- The size of queues used before stages.
- Service Providers scheduling.

DES models track the times that entities go through the stages of the system and keep records of the time of each event. So, they can analyze each entity's wait time and service time with its own unique attribute. In the Complex Case of this study we can see how DES can simulate an ED elaborately. To run the simulation and obtain the average length of stay, a JSON string that contains the name of the stage the number of minimum and maximum providers required at each shift of the stage is given as an input.

2.2 Genetic Algorithms

Ever since the creation of universe, evolution has been a key part in its functioning. New organisms have evolved from their ancestors; and this evolution is governed by a simple law which Charles Darwin named as – "*Survival of the Fittest* ".

In the early 60's, Genetic Algorithms were developed by John Holland and his co-workers in the University of Michigan. Inspired by natural genetics and natural selection, Genetic Algorithms use a search technique used in computing to find true or approximate solutions to optimization and search problems. GAs have the potential to create an initial population of feasible solutions, and then recombine them in a way to guide their search to only the most promising areas of the state space They are widely used in Function Optimization and Control Applications. It is better than conventional AI in that it is more robust. Unlike older AI systems, they do not break easily even if the inputs changed slightly, or in the presence of reasonable noise. Also, in searching a large state-space, multi-modal state-space, or n-dimensional surface, a genetic algorithm may offer significant benefits over more typical search of optimization techniques.

GA's use a direct analogy of natural behavior. They work with a population of individuals called as chromosomes each representing a possible solution to a given problem. A chromosome is composed from genes and assigned a fitness value that tells how good a solution to the problem it is. The highly fit individuals are given an opportunity to reproduce by performing a crossover with other individuals in the population. This operation produces an offspring that shares the features from both the parents. The least fit members of the population are not likely to be selected for reproduction. A whole new population of possible solutions is thus produced by selecting the best individuals from the current generation, and mating them to produce a new set of individuals. This new generation contains a higher proportion of the characteristics possessed by the good members of the previous generation.

In this way, over many generations, good characteristics are spread throughout the population, being mixed and exchanged with other good characteristics as they go. By favoring the mating of the more fit individuals, the most promising areas of the search space are explored. If the GA has been designed well, the population will converge to an optimal solution to the problem. The power of GAs comes from the fact that the technique is robust, and can deal successfully with a wide range of problem areas, including those which are difficult for other methods to solve. Where specialized techniques exist for solving certain problems, they are likely to out-perform GAs in both speed and accuracy of the result. The flowchart of a Genetic Algorithm is as shown in the figure below:

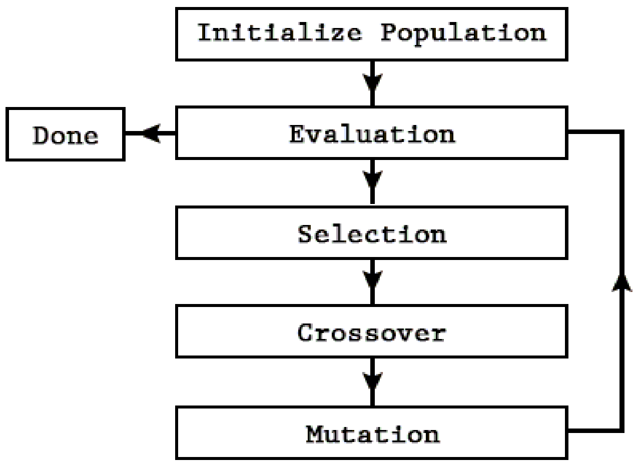

Figure 2.1: An overview of Genetic Algorithms

The main ground for GAs, then, is in difficult areas where no such techniques exist. Even where existing techniques work well, improvements have been made by hybridizing them with a GA. Therefore, any Genetic Algorithm consists of a fitness function, a selection technique, and crossover and mutation operations which work on fixed probabilities.

2.2.1 Fitness

The fitness function provides a way for the GA to examine the performance of each chromosome in the population. The function must be properly selected as the fitness function is the only relation between the GA and the application itself. The fitness function must reflect the application appropriately with respect to the way the parameters are to be minimized.

2.2.2 Selection

The selection operator selects chromosomes from the current generation to be parents for the next generation. The probability of each chromosomes selection is given by:

$$Ps(i)=f(i)/\sum_{J=1}^{N} f(j)$$

where ***ps (i)*** and ***f (i)*** are the probability of selection and fitness value for the ith chromosome respectively. Parents are selected in pairs. Once one chromosome is selected, the probabilities are renormalized without the selected chromosome, so that the parent is selected from the remaining chromosomes. Thus, each pair is composed of two different chromosomes. It is possible for a chromosome to be in more than one pair.

2.2.3 Crossover

Crossover is the GA's primary local search routine. The crossover/reproduction operator computes two offspring for each parent pair given from the selection operator. These offspring, after mutation, make up the new generation. A probability of crossover is predetermined before the algorithm is started which governs whether each parent pair is crossed-over or reproduced. Reproduction results in the offspring pair being exactly equal to the parent pair. The crossover operation converts the parent pair to binary notation and swaps bits after a randomly selected crossover point to form the offspring pair.

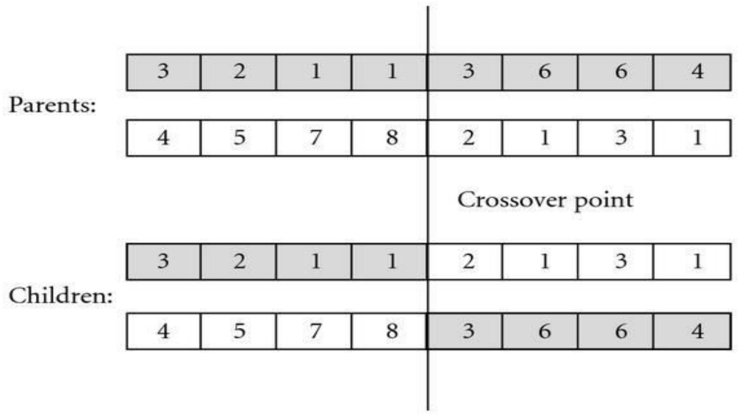

Fig 2.2: Crossover Operation

2.2.4 Mutation

Mutations are global searches. A probability of mutation is again predetermined before the algorithm is started which is applied to each individual bit of each offspring chromosome to determine if it is to be inverted.

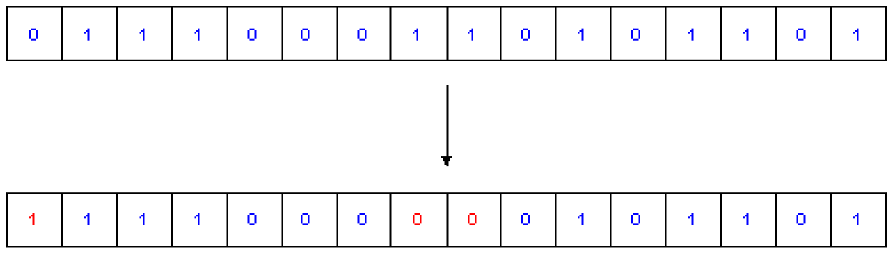

Fig 2.3: Mutation Operation

2.3 JSON

In computing, JavaScript Object Notation or JSON is an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value) i.e. data interchanging. An example of where this is used is web services responses. Previously, web services used XML as their primary data format for transmitting back data, but since JSON appeared, it has been the preferred format because it is much more lightweight.

JSON is a language-independent data format. It was derived from JavaScript, but as of 2017 many programming languages include code to generate and parse JSON-format data. The official Internet media type for JSON is application/json. JSON filenames use the extension. json.

CHAPTER 3
DESIGN AND IMPLEMENTATION

3.1 Design

Every patient going to an ED would need to experience several stages or services until the point that they get discharged or admitted to the hospital. Much of the time, the first stage is registration. The second stage could be consulting a nurse who wants to make a preparatory diagnosis based on patient's symptoms and medical history. The following stage might be receiving first treatment by a nurse. Usually after a patient passes the first nurse, they must see a doctor for the next stage. The doctor would try to make a stronger and much clearer diagnosis of the patient's disease. Then the diagnosis might require further examination and laboratory diagnostic as the next stage.

In this case, the patients must wait for the Lab results to be presented to the doctor. Then some treatment would be applied to the patients depending on the doctor's decision. Finally, according to the patient's situation, the doctor would decide if the patient can be discharged or should be admitted to the hospital. So, there are several stages in the emergency department that all the patients must go through as illustrated in the following figure.

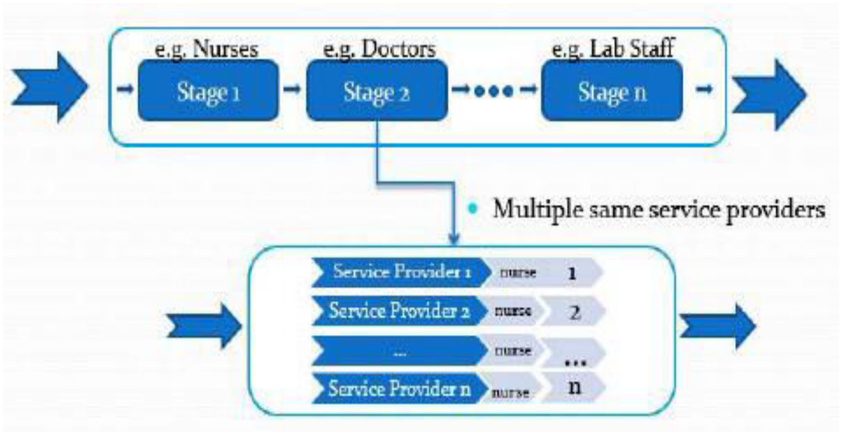

Figure 3.1: Multi-Stages Emergency Department

According to the patient's level of triage (severity of the health issue), there may be a need for a number of required stages to go through before being discharged or admitted. Also, stages are different in nature and each has its own properties such as service rates and number of service providers. Furthermore, emergency departments may be different in number of stages and service providers as well as service rates. To do modeling or optimization for an ED, the flow of different patient categories should be looked at and evaluated. In fact, understanding the interconnectivity and dependability of the stages are critical for successful modeling and optimization.

After a patient is served in stage i, she would move to the next stage, stage i+1. If there is any free service provider in stage i+1, the patient does not need to wait, and she can move directly to stage i+1 and get served. However, if there is no free service provider in stage i+1, the patient must wait until a service provider is free before she is provided services. If the number of patients waiting reaches a set capacity, the new incoming patients would have to be sent another hospital. In this case, the patient is considered as a LWBS which means "Left Without Being Seen". Moreover, if a patient's waiting time takes more than her tolerance level, she may decide to leave the emergency department without being seen in the next stage (LWBS classification). Patients have different wait time tolerances. Some patients might be able to wait more than one hour to be seen by a doctor, but some patients might leave the emergency department after twenty minutes wait. LWBS patients usually form a considerable portion of all patients attending Emergency Departments, consequently the simulation methodology presented in this work considers this group of patients. As mentioned earlier, the main goal of the optimization process presented in this work is to minimize the total wait time in ED.

First, a simple model is considered for the simulation of patient flow, understanding the total cost equation and later, it's complexity will be increased by applying DES simulation.

For the simple case, we assume that the emergency department is formed of a multi-stage, multi-server queue. Each stage provides a specific service by a set of providers. For example, there might be four nurses responsible for the registration stage, five doctors responsible for the preliminary diagnosis as the second stage, three nurses responsible for taking care of those patients who need lab examinations as the fourth stage, and so on.

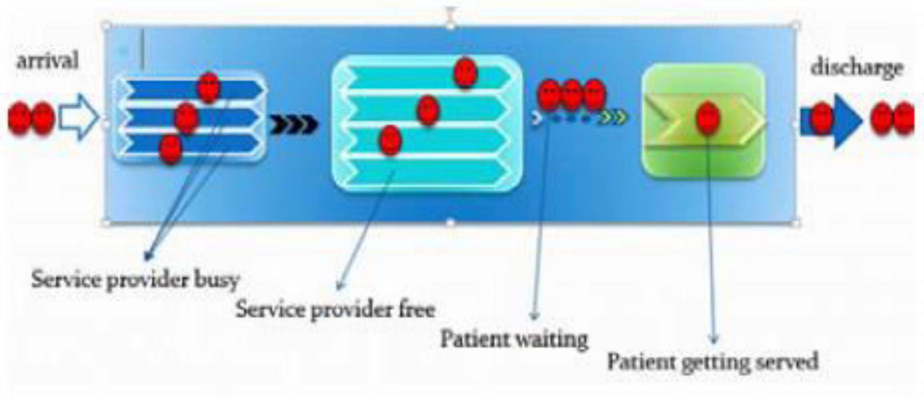

Figure 3.2: Patient flow through different stages

3.1.1 Service Rate

Depending on the type of the provided service and the number of its service providers, a stage has an average service rate of "μ". For instance, in the first registration stage, it takes 5 to 10 minutes for each nurse to register a patient and take the patient's information. However, it might take over 30 minutes for a doctor to see the patient and make the first diagnosis in the second stage. It even may take much longer for the lab stage to prepare the blood test results.

3.1.2 Arrival Rate

Every queue requires an arrival rate. For our emergency department the arrival rate is the rate of the number of patients entering the first stage. In this simple model, it is assumed that the patient's arrival rate follows the Poisson probability distribution with the same mean arrival rate "λ" for all stages. This property is the "equivalence property" of multi-stage queuing systems. Here we explain the model with a simple ED case with 3 stages.

N_i= the number of service providers in stage $i = 1, 2, 3$ (such as the number of nurses or doctors)

μ_i = the average service rate for stage $i = 1, 2, 3$

λ_i = arrival rate $i = 1, 2, 3$

φ_i = the occupancy rate of stage $i = 1, 2, 3$

Based on queuing theory we have: $\varphi_i = \dfrac{\lambda}{N_i \mu_i}$

We also assume that the stages are in statistical equilibrium. So we have: $\varphi_i < 1$

The probability of no patients in stage $i = 1, 2, 3$ is

$$P_{0i} = \frac{N_i!(1-\varphi_i)}{(\varphi_i N_i)^{N_i} + N_i!(1-\varphi_i)\sum_{n=0}^{N_i-1}\frac{1}{n!}(\varphi_i N_i)^n}$$

The probability of n patients in stage $i = 1, 2, 3$ is:

$$p_{ni} = \frac{(\varphi_i N_i)^n}{n!} p_{oi} \quad n = 1,2,\dots, N$$

$$\dots = \frac{N_i^{N_i}}{N_i!\varphi_i^{\,n}} p_{0i} \quad n > N$$

Consequently, the average number of patients in stage i is:

$$L_i = \frac{\varphi_i(\varphi_i N_i)^n}{N_i!(1-\varphi_i)^2} p_{0i} + \varphi_i N_i$$

There are different contemplations in regards to number of service providers in different stages that might be pertinent. For example, there could be minimum and maximum limits for the number of servers in each stage. So, the range of the number of servers in each stage may vary and should be considered when finding the optimum number of providers for each stage. To formulate this simple model, we consider patient wait time as a cost factor which depends on the average wait time of patients (Waiting Cost - WC). The longer the waiting time, the higher is the cost. Additionally, each service provider would incorporate cost which would vary depending on the assigned roles (Service Cost – SC). So, the general equation of the total cost (TC) could be represented as follows:

$$TC=WC+SC$$

To expand the above formula, we would need to calculate WC and SC separately.

3.1.3 Calculation of Service Cost

For each service provider, Service cost literally is the money we have to pay him or her for providing the service over a period of time. Depending on which stage the server is working, the level of professionalism and consequently the cost is different. So, for each service provider, we call the cost *Ci* in stage *i* (i = 1,2,3). Therefore, the total cost for stage *i* having *Ni* service provider is *NiCi*. Moreover, the total service cost for all the stages is:

$$SC = \sum_{i=0}^{3} N_i C_i$$

3.1.4 Calculation of Waiting Cost

Let *f* (*n*) *i* to be the function that calculates the waiting time cost for stage *i*, *n* to be the number of patients in stage *i*, and *Ni* to be the number of servers in stage *i*. It is trivial that if we have more service providers than patients in each stage, there would be no wait. However, if the number of service providers is less than the number of patients, we will have some patients waiting until providers become free. Let *CWi* to be the average cost for waiting in stage *i.* The *CWi* would be different for each stage since the consequences for waiting for each service may be different. For example, waiting cost for getting discharged is not as critical as waiting cost for being seen by a doctor. Here we have:

$$f_i(n) = \begin{cases} (n - N_i)CW_i & n > N_i \\ 0 & n < N_i \end{cases}$$

Since we have three stages in this example, the total waiting cost is:

$$\begin{aligned}
WC &= \sum_{n=0}^{\infty} f_1(n)p_{n1} + \sum_{n=0}^{\infty} f_2(n)p_{n2} + \sum_{n=0}^{\infty} p_{n3} \\
&= \sum_{n=0}^{\infty}(n-N_1)C_{w_1}p_{n1} + \sum_{n=0}^{\infty}(n-N_2)C_{w_2}p_{n2} + \sum_{n=0}^{\infty}(n-N_3)C_{w_3}p_{n3} \\
&= \sum_{i=1}^{3} C_{w_i} \sum_{n=N_i+1}^{\infty}(n-N_i)C_{w_i}p_{ni} \\
&= \sum_{i=1}^{3} C_{w_i} \sum_{n=N_i+1}^{\infty} np_{ni} - \sum_{i=1}^{3} C_{w_i}N_i \sum_{n=N_i+1}^{\infty} p_{ni} \\
&= \sum_{i=1}^{3}[C_{w_i}(\sum_{n=0}^{N_i} np_{ni})] - \sum_{i=0}^{3}[C_{w_i}N_i \sum_{n=N_i+1}^{\infty} \frac{N_i^{N_i}}{N_i!}\varphi_i^{n} p_{0i}] \\
&= \sum_{i=1}^{3}[C_{w_i}L_i - \sum_{n=0}^{N_i} np_{ni}] - \sum_{i=1}^{3}[C_{w_i} \frac{N_i^{N_i}}{(N_i-1)!} p_{0i} \sum_{n=N_i+1}^{\infty}\varphi_i^{n}] \\
&= \sum_{i=1}^{3}[C_{w_i}L_i - \sum_{n=0}^{N_i} np_{ni}] - \sum_{i=1}^{3}[C_{w_i} \frac{N_i^{N_i}}{(N_i-1)!} p_{0i} \frac{\varphi_i^{N_i+1}}{1-\varphi_i}]
\end{aligned}$$

As mentioned earlier, φ_i is the occupancy rate of stage *i*, and *pni* is the probability of having *n* patients in stage *i*. Consequently, by adding equations of TC and SC we can calculate total cost

$$TC = \sum_{i=0}^{3} N_i C_i + \sum_{i=1}^{3} [C_{w_i} L_i - \sum_{n=0}^{N_i} n p_{ni}] - \sum_{i=1}^{3} [C_{w_i} \frac{{N_i}^{N_i}}{(N_i - 1)!} p_{0i} \frac{{\varphi_i}^{N_i+1}}{1-\varphi_i}]$$

To find a global optimum solution for TC, we have to find the best set of numbers for service providers, which would minimize the value of equation. Considering the high complexity of searching the solution space for equation when there are several stages are considered, a Genetic Algorithm approach has been proposed.

3.2 Implementation

The genetic algorithm discussed in this work is implemented in C#. It is clear that the total cost equation should be used as the fitness function for the genetic algorithm search. The input of this function is a set of numbers representing service providers in each stage and the output is the total cost associated with this given set. The output is the total cost, including the cost of service providers (SC) and the waiting cost (WC). Here, the cost is considered as the length of stay of the patient. Each stage is also associated a range for the possible number of providers it may include and an average service rate for its providers.

The overall design of the genetic algorithm is illustrated in the following figure.

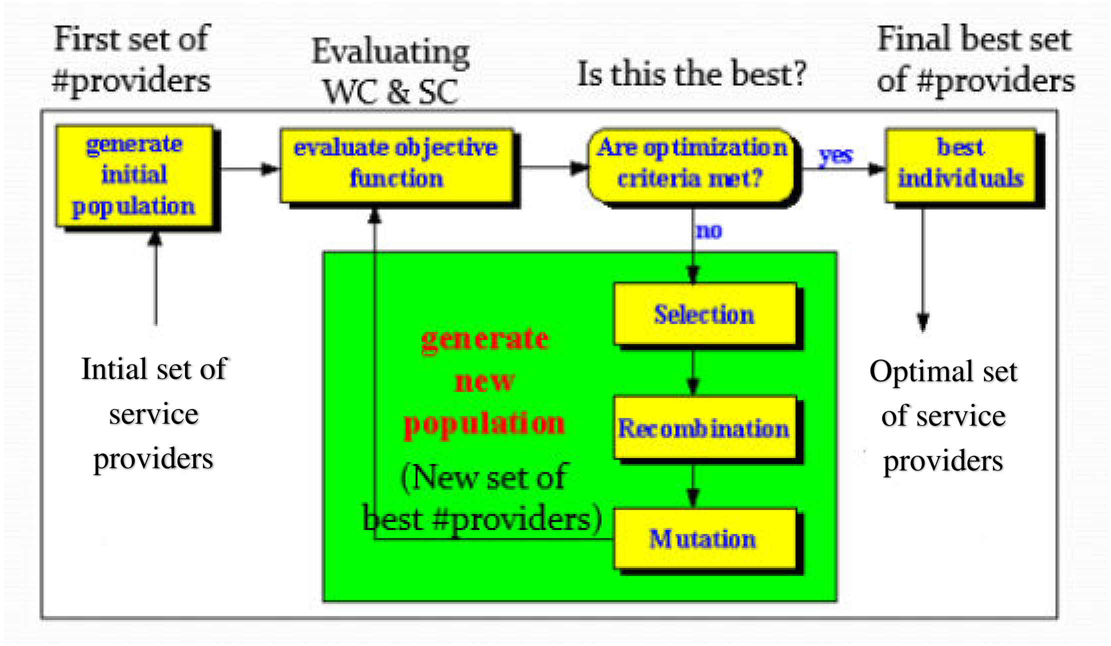
First set of #providers
Evaluating WC & SC
Is this the best?
Final best set of #providers
generate initial population
evaluate objective function
Are optimization criteria met?
yes
best individuals
no
Selection
generate new population
Recombination
(New set of best #providers)
Mutation
Intial set of service providers
Optimal set of service providers

Figure 3.3: Iterations of Genetic Algorithm

The chromosome used in this algorithm is represented below. It basically represents the number of servers at each shift of a stage and keeps them inside the given ranges for each stage.

2	2	2	3	4	2	1	3	2	2	2	1	2	2	1	2	9	10	6	10	8	7

Table I: Representation of the number of service providers as a chromosome

In this 7-stage model, each chromosome is formed from 22 genes associated with the range of service providers in the 7 stages. The initial population in this algorithm (Old Population 1), as customary, consists of many randomly generated chromosomes. In general, the number of chromosomes in a population (the population size) varies depending on the size of the search space. In our algorithm the population size is initially set to the range between 25-50. To form consecutive generations, a 2 by 2 model is utilized, meaning two parents generate two offspring.

To select the mating chromosomes, we utilize a customized roulette wheel selection algorithm which first forms a poll of contributing chromosomes and then perform the probability based selection according to the fitness measure of each chromosome. To form the poll of the participating chromosomes, this algorithm includes 20% of the fittest chromosomes from the current generation and 20% of the fittest chromosomes from the previous generation. Then it applies the classical probability based roulette wheel method to select the remaining 60% of the population from remaining chromosomes in the current and the previous generations (30% of each). After evaluating several different methodologies for chromosome selection, we found out that this modified roulette wheel selection method has the best performance in finding global optimum solutions for our problem. Additionally, after evaluating several different values, we assigned 0.5 as the probability values for crossover operation. In case of crossover, we randomly select many genes from each contributing chromosome to form two offspring. For mutation, only one gene is flipped to a different value in the acceptable range. The stopping criteria depends on the fitness stagnation condition that was provided in the algorithm.

For a bigger problem such as the one with seven stages, we may not be able to accurately predict a boundary for the number of iterations. Instead, in addition to presetting the number of iterations to a relatively high value, we compare the average fitness of the chromosomes as well as the fittest chromosome of the last 15 generations and stop if there is no improvement made. This comparison is only made when the preset number of iterations has passed. At the end, when the stopping criterion is met, the fittest chromosome will be reported as the solution.

To explain the cost equation, a multi-stage multi-server queue was used to formulate the overall cost of ED which was used as the fitness function for the genetic algorithm. However, in real world, EDs are much more complex. For example, at an ED triage stage, the patients who are arriving are classified to five different triage levels according to the severity of their condition. Patients with different triage level may go through different tracks and sequence of stations. Moreover, different patients may require different treatment and therefore different stations. Additionally, patients' behavior varies, so we must consider different waiting tolerances for different patients and the fact that in different stages we may have patients who would leave without being seen. In this section, we present a DES simulation model that considers most of the complex properties in an emergency department. Like the simple case, the complex case has two major components which are the simulation model and the optimization algorithm. The optimization algorithm is very similar to genetic algorithm discussed for the simple case, while the simulation model is quite different. To implement the complex DES simulation model, the server has utilized Matlab Simulink. We created the simulation using timed-based blocks in Simulink called the Web API in the genetic algorithm using HttpRequest and HtttpResponse developed in C# for optimization. It is obvious that for this model, the fitness function utilized by the genetic algorithm would be the representation of the total cost of complex model.

3.2.1 Optimization Cycle

The goal of the GA optimization algorithm is to find the best set of numbers for service providers in different stations, as discussed earlier. For this purpose, the C# GA algorithm takes the control of the optimization and simulation. First, GA runs the simulation by sending the initial state of the emergency department, which is the first chromosome of the current generation, to the Simulink simulator created according to the properties of the specific ED being evaluated. Then the simulator calculates the fitness value based on the average wait time and the service cost which is returned to the GA. So, for optimization purpose, the simulator functions as the fitness calculator for the genetic algorithm. The genetic algorithm then runs for several iterations until it meets the stopping criteria (stated under the simple case) and returns the fittest chromosome as the result.

3.2.2 Genetic algorithm libraries (GeneticSharp)

GeneticSharp is an open-source Genetic Algorithm library for C#, developed by Giacomelli and released under the MIT license. It has an extensible interface that allows for most, if not all, functionality to be implemented from scratch via interfaces or leveraged by extending base classes. Classes and interfaces also use the same terminology that has already been established, which makes the translation from theory to implementation much clearer. Since the GitHub repository has an explanation of included features, and contains some example usages, there is a low requirement to reiterate it here.

CHAPTER 4
EXPERIMENTAL ANALYSIS

4.1 A Demonstration of ED with Seven Stages

A complex emergency department with 7 stages is considered in this work. Each stage could have a possible range of the number of providers. Service cost and waiting cost are also required data that need to be entered to the system according to the hospital information. This information is present in the JSON string which we give as input. The genetic algorithm meets the stopping criteria (59 interactions) within few minutes with a normal core i5 CPU (2.40 GHz). The algorithm successfully finds the minimum point in the search space, the fittest chromosome.

The range of the number of the servers in each stage for this example is as shown in the table below.

Stage	1	2	3	4	5	6	7
Number of Shifts	3	5	2	3	3	3	3
Range of providers	1-2	1-4	1-2	1-2	1-2	5-10	5-10

Table II: Stage, Number of shifts and Range of Providers in the Emergency Department

The same formula for the fitness function is utilized here, but naturally, it has become more complex with a higher calculation cost. In this example, the number of generations were limited to 59 and the cost was being observed by first maintain the population size between 25-50 and then increasing it to the range 50-100 and then to 100-150. As expected, larger population size was slowing down the computation greatly. It takes several minutes for the genetic algorithm to reach a solution considering more complex fitness calculation and larger number of chromosomes. The following solution is reached after 59 iterations for the 7 stages. The following table displays the optimal number of service providers required at every shift of a particular stage.

Stage number	**Stage ID**	**Stage Name**	**Number of shifts**	**Set of providers**
1	41	Registration	3	2,2,2
2	42	Nurse	5	3,4,2,1,3
3	43	Physician	2	2,2
4	44	Imaging	3	2,1,2
5	45	Lab	3	2,2,2
6	46	Admit Approval	3	9,10,6
7	47	Transfer Approval	3	10,10,8

Table III: Optimal number of service providers at each stage.

4.1.1 Performance Evaluation

The stop condition of the genetic algorithm was set to 59 iterations. During several runs on different number of stages with different population sizes, it was observed that the algorithm successfully returned either the optimum or a close to optimum solution. However, the more the number of the stages and their shifts, the longer the GA would take to return a solution. One way to improve the performance for large EDs is to modify the stopping criteria. It happens less frequently that earlier generations include the final length of stay of the patient.

The graphs below demonstrate how the genetic algorithm was able to minimize the length of stay of the patients over many generations by optimally allocating service providers at each shift of a stage.

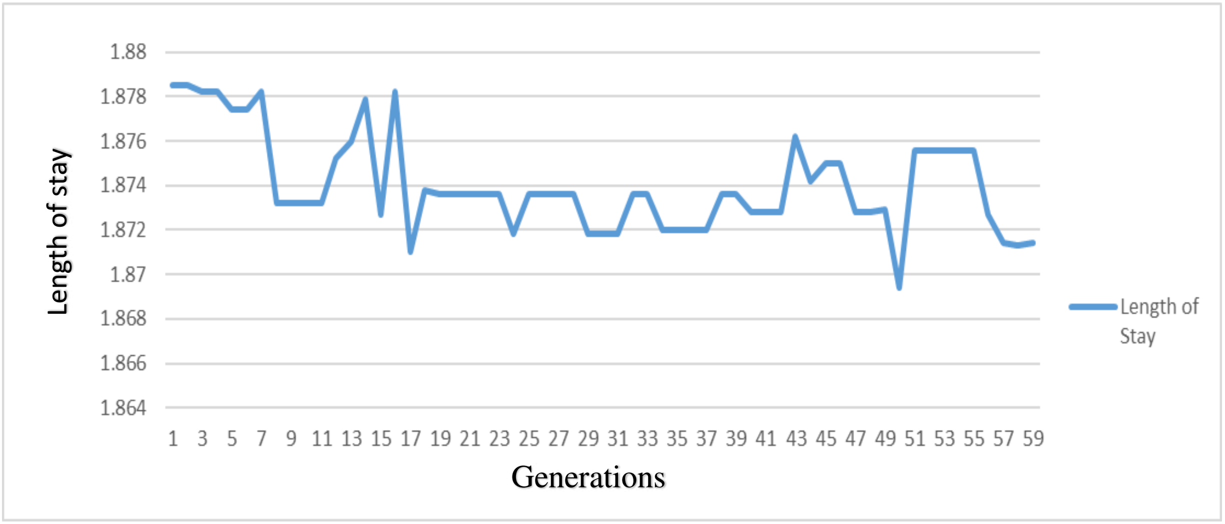

Figure 4.1: Length of Stay for a population size of 25-50

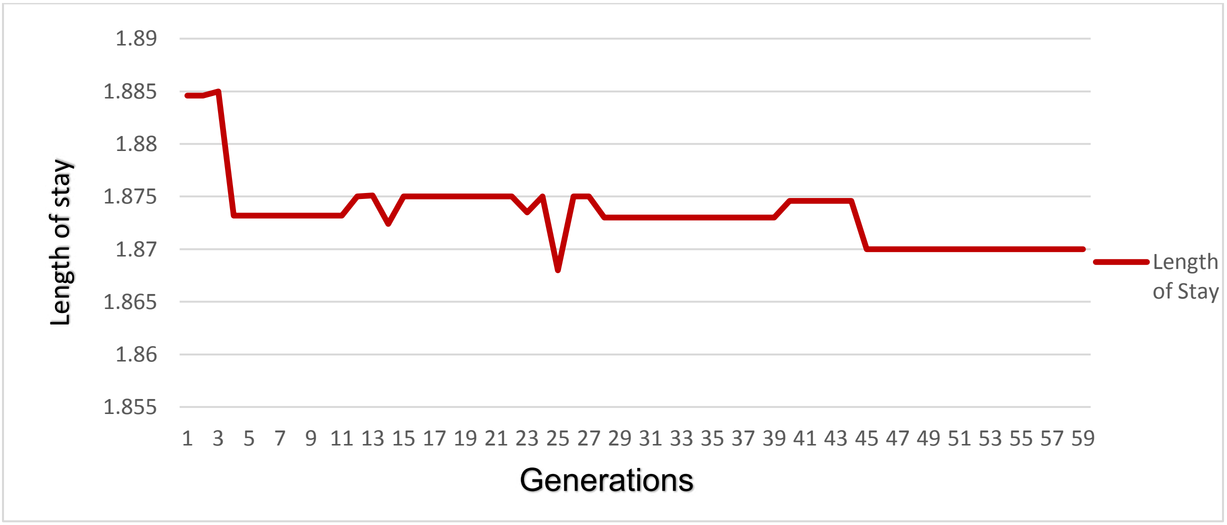

Figure 4.2: Length of Stay for a population size of 50-100

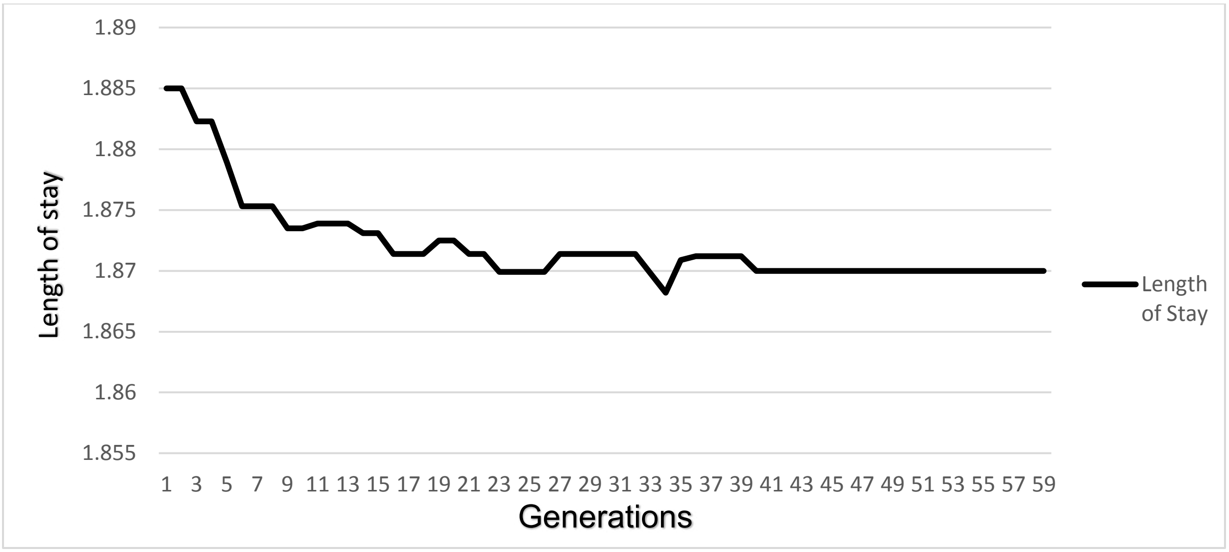

Figure 4.3: Length of Stay for a population size of 100-150

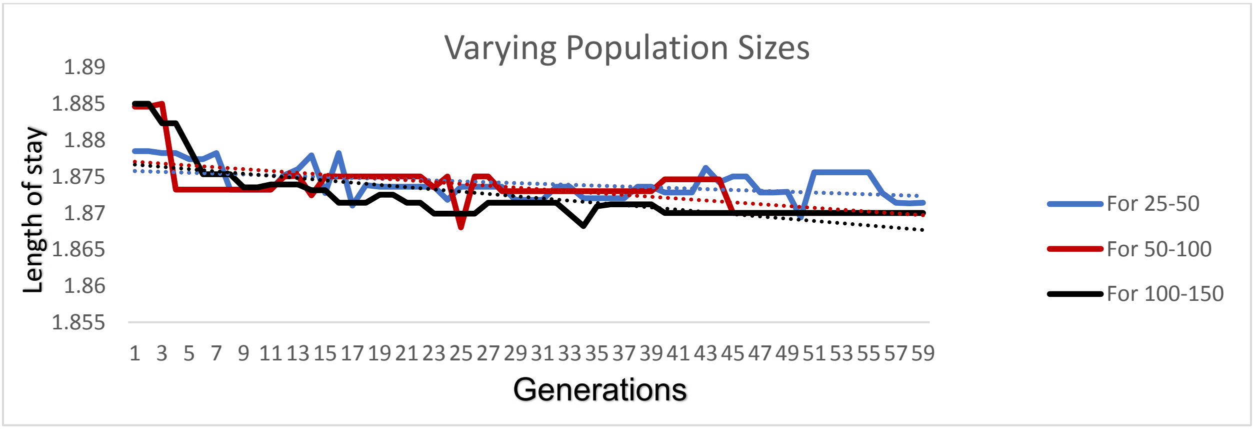

Figure 4.4: Graph showing the trend line of varying population sizes

As observed in the above graph, the algorithm reaches to a minimum value faster if we increase the population size. This is because it is efficient to select a best value among a very large set of chromosomes.

CONCLUSION

In this project, we developed a genetic algorithm that utilizes the discrete event simulation model for the optimal allocation of service providers at each and every stage of the Emergency Departments (ED). The simple EDs were first modeled and evaluated to explain about the fitness function that we use for the Genetic Algorithm. For the simulated EDs, many detailed information such as the waiting time of each and every patient in different stages could be calculated using the number of minimum and maximum providers allocated at each and every stage. The genetic algorithm was developed in C# using GeneticSharp with appropriate fitness functions calculated from DES model to find the optimum service provider arrangement for minimizing the length of stay of the patient. In this work, it was illustrated that the DES model has better performance than the simple model for complex EDs since it is difficult to model many characteristics of large EDs in queuing models. A JSON string which contains the range of service providers to be allocated at every stage is given as an input to the DES simulation by calling a Web API.

REFERENCES

[1] Jacobson, S. H., Hall, S. N., & Swisher, J. R. (2006). Discrete-Event Simulation of Health Care Systems. In R. Hall, Patient Flow: Reducing Delay in Healthcare Delivery (pp. 211-252). New York: Swisher.'

[2] Vlasis K. Koumousis and Christos P. Katsaras," A Saw-Tooth Genetic Algorithm Combining the Effects of Variable Population Size and Reinitialization to Enhance Performance", IEEE Transactions on Evolutionary Computation, Vol. 10, No. 1, February 2006.

[3] Phillip David Power, "Non Linear Multi-Layer Perceptron Channel Equalisation", Chapter 4 „Genetic Algorithm Optimisation", in IEEE Transactions, The Queen University of Belfast, April 2001.

[4] Apache and the JSON license on LWN.net by Jake Edge (November 30, 2016)

[5] Wiler, J. L., Griffey, R. T., & Olsen, T. (2011). Review of modeling approaches for emergency department patient flow and crowding research. Academic Emergency Medicine, 18(12), 1371-1379.

[6] Akbari, Ziarati (2010). "A multilevel evolutionary algorithm for optimizing numerical functions" IJIEC 2 (2011): 419–430.

[7] Green, L.V., Giulio, J., Green, R., and Soares, J., 2005, Using queueing theory to increase the effectiveness of physician staffing in the emergency department, Academic Emergency Medicine, to appear.