A Queueing Theoretic Approach to Gridlock Prediction in Emergency Departments

by

Toros Caglar

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Industrial and Systems Engineering

July 1, 2005

Blacksburg, VA

Keywords: Gridlock, Emergency Departments, Predictive Modeling, Healthcare

A Queueing Theoretic Approach to Gridlock Prediction in Emergency Departments

Toros Caglar

(Abstract)

When an emergency department (ED) decides that it is not going to be able to serve any more newly arriving patients, it declares "diversion". When an ED is on diversion, it suspends arrivals that can be controlled by forcing some or all of the incoming emergency medical system (EMS) transport units to search for alternate treatment facilities for their patients. This search causes both patients and EMS crew to loose valuable time. Contrary to the general belief that suggests diversions are not very common, the results of the American Hospital Association survey present an example where one third of the studied hospitals were on diversion more than 20% of the three-day study period.

Past research indicates that the lack of critical care beds in the hospital is the primary contributor to ambulance diversion. When patients need to be transferred from the ED to the hospital with no available beds in the hospital, they continue occupying their beds (i.e. the patient is boarding). While they are boarding in the ED, the associated staff is idle, and their bed cannot be used to treat other patients. Boarders in the ED lead to gridlock, which is defined as the situation when no new patient can be accepted to the ED until a hospital bed becomes available.

In this research, we developed a predictive model to provide probabilities of entering gridlock within a time horizon, given the current state of the system. These real-time predictions are provided for a relatively short time horizon, and in order to be useful, they need to be used in conjunction with effective preventive measures that can be applied quickly. The predictive model is based on a queueing theoretic approach and encapsulated in a user-friendly Visual Basic program in order to calculate and provide gridlock probabilities. Two systems, one with low (24% - System 1), and one with high (81% - System 2) gridlock probability were simulated in conjunction with our predictive model and preventive measures. When a gridlock was found imminent, the number of ED beds was temporarily increased, attempting to prevent gridlock. With only 3 additional beds, the probability of gridlock decreased to 6% in System 1 and 58% in System 2. With 5 additional beds, gridlocks in System 1 were almost eliminated while System 2 entered gridlock only 34% of the time. Our results indicate that by temporarily increasing the number of ED beds in the event of an imminent gridlock, the proportion of time that system enters gridlock can be significantly reduced.

Acknowledgements

I would like to start by thanking my advisor, Dr. Patrick Koelling, for his countless contributions to this work, his support and encouragement, and for introducing me to the area of healthcare. I would also like to express my appreciation to my committee, Dr. Kimberly Ellis for her valuable advice, and Dr. Michael Taaffe, for the technical perspective and assistance he provided. I want to thank Dr. David Eitel and York Hospital for providing the data and information that made this research possible. I also want to mention Sid Mahapatra as an important contributor to this research with his experience and knowledge in the area of healthcare.

Many thanks to Erkin Seker, Sinem Goknur, and Devrim Demirel for letting me stay at their house and making me feel at home during the final month of my research.

I would like to express my deepest appreciation to my family, Can, Cagla and Tolga Caglar, whose support, encouragement and love were very important to me throughout my studies. I must also thank Ekin Tusalp for making this research possible with her extraordinary support and patience, and for being an invaluable source of motivation and inspiration to me over the years.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 Background

The mission of emergency departments is to provide timely emergency care to patients in need of medical attention. After attracting the public's attention more than a decade ago, overcrowding in emergency departments (ED) has resurfaced as a healthcare crisis in the past few years. Unfortunately, this time overcrowding is even more widespread, and in some places accepted as the standard of care (Henry 2001). According to surveys by the American Hospital Association (AHA), over 50% of the 28 hospitals studied describe their ED to be running at or over capacity (TheLewinGroup 2002; TheLewinGroup 2004). With patient waiting times averaging as high as 96 minutes, the EDs are rarely able to comply with the first clause of their mission, timeliness.

Patients with disparate needs and varying urgencies, randomly and without prior notice, arrive to the ED either on foot, or by an emergency medical system (EMS) transport unit. Due to the nature of the demand, peaks in the system busyness are common, and it is a challenge for the ED to respond to patient needs in a timely manner. During such chaotic periods, EDs can decrease their loads significantly through "diversion."

An ED on diversion suspends arrivals that can be controlled (i.e., non-walk-in patients) by forcing some or all of the EMS transport units to search for alternate treatment facilities for their patients (Bennett 2003). During this search, the valuable time that the patient is losing, and the unavailability of the EMS crew to answer other calls, are just two of the problems that the ED's inability to maintain availability is responsible for. An investigation in Houston, Texas, has revealed results about the relationship between diversions and trauma mortality rates, which revealed the acuity of this problem. During a two-year period, the mortality rate of severely injured trauma patients was almost twice as high (25% vs. 14.4%) on days when both of Houston's Level I trauma centers were on diversion (Brewer 2002). Unfortunately, such studies linking ambulance diversion to unsatisfactory healthcare performance are not numerous. Policymakers do not address diversion as a healthcare priority for a variety of reasons. The most obvious reason, according to Brent R. Asplin (2003), a physician at the Department of Emergency Medicine at Regions Hospital in St. Paul, Minnesota, is that the public did not ask them to do so. Even though diversion is a known problem, most people do not believe they will ever be diverted. The 2004 AHA survey results present an example contradicting this belief; during the three days the survey was conducted, one third of the 28 studied hospitals were on diversion more than 20% of the time. Lack of studies on linking poor ED performance and ambulance diversions is also a factor for the insufficient attention on ambulance diversion. When initially implemented, diversion was supposed to be a temporary mechanism that was rarely used. However, nowadays diversions have almost become standard operating procedures and need to be investigated.

1.2 Research Objectives and Approach

AHA ambulance diversion surveys show lack of critical care (CC) beds in the hospital as the major cause of the diversions (TheLewinGroup 2002; TheLewinGroup 2004). After being treated in the ED, the patient is either discharged (no further immediate treatment is necessary), or admitted to the hospital. However, admission may not be possible due to the lack of available CC beds in the hospital. In this case, the patient occupies a bed in the ED, waiting for a CC bed to become available. This condition may lead to gridlock, which may be defined as the state of an ED when (i) all beds in the ED are occupied, and (ii) there is at least one patient in one of the beds in the ED waiting to be transferred to the hospital, but the transfer is not possible due to the lack of available beds. When these conditions are reached, the idle patient that needs to be transferred is said to be on hold and is occupying a bed in the ED, preventing its use by other patients.

The objective of this research is to construct a predictive model that can provide a probability of entering gridlock within a time horizon, conditioned on the system's current state. The effects of using the predictions to employ certain preventive measures are then investigated, via a simulation evaluation of the model of the ED. A user friendly Visual Basic application is also created to allow the use of the predictive model in a real world system.

The ED is a very complex and variable system. Patients with different needs and condition severities create a mixed arrival stream. Individual service time distributions for each type of patient make the system more complex, as does the set of various resources. The capabilities and expertise of the nurses and the physicians are not all alike. Furthermore, the problem that this research will focus on requires interactions between the hospital and the ED to be examined. The system composed of the ED and the hospital is a very complex one for queueing theory to be applied thoroughly. Hence, some assumptions and simplifications are needed to make the model more manageable. These assumptions will be discussed in detail in sections to follow.

2 Literature Review

To cover the various disciplines involved in predicting and preventing gridlock, this literature review consists of three major sections. The first section concentrates on work describing and examining overcrowding in the ED and ambulance diversions. Section 2.2 concentrates on queueing theory research studying queueing networks and their applications in the healthcare area. The third portion of the literature review presents research on using simulation based methods in evaluating healthcare system models including EDs.

2.1 ED Overcrowding and Ambulance Diversions

As overcrowding in the ED and ambulance diversions are increasingly becoming serious problems that affect the public's health, they are attracting the attention of people from different backgrounds and occupations. Mark C. Henry (2001), a medical doctor from State University of New York, investigates overcrowding in EDs from a physician's point of view, searching for reasons and solutions for the problem. He describes the sole reason of overcrowding as "boarding inpatients already admitted to the hospital" (p. 188). According to Henry, "these patients are kept in ED beds, on stretchers placed in hallways, or in 'observation' areas, with little if any regard for privacy, dignity or personal hygiene" (p. 188) for hours, or even several days. These patients, while waiting to be transferred to the hospital, occupy the spaces for other patients who need "emergent evaluation or treatment" (p. 188), causing gridlock. To prevent further congestion, the ED often declares diversion, and asks ambulances to find alternate treatment facilities for their patients. As a result, "the ability of the hospital to provide emergency care to its community and serve its role in the emergency medical services (EMS) is lost" (p. 188). Henry also suggests that there are financial reasons behind boarding inpatients in the ED. To cut costs and increase hospital utilization, beds and staff are downsized. This situation does not leave any margin for fluctuations in patient volume and results in very few, if any, empty beds in the hospital at the beginning of the day. A common scenario, according to Henry, is to fill the hospital beds with scheduled, "more profitable" patients before letting the boarders in the ED fill the remaining beds. He proposes that by acknowledging the problem, taking responsibility, recognizing that healthcare is driven by profit/loss, and returning to the most basic principles of the ED, which is to serve "incoming patients with emergent needs" (p. 189), a step toward solving this problem will be taken.

Another MD, Brent R. Asplin (2003), examines the question "does ambulance diversion matter." In his work, Asplin points to the policymakers' failure to consider ambulance diversion as a problem, and searches for possible explanations for this disinterest. Asplin lists as the most obvious reason for the lack of attention on diversions is that most people do not think that it would happen to them, and are unwilling to see it as a problem. Another reason he lists is the lack of solid data and studies connecting ambulance diversions to poor healthcare service performance. According to Asplin, most information trying to link diversions and poor healthcare performance is either anecdotal, or at best, isolated and can hardly be generalized. At the end, he concludes that diversion indeed is a problem deserving proper attention.

One of the research efforts that Asplin brings up to support his point is by Shull et al. In their study, Shull et al. (2003) investigated the “relationship between physician, nursing, and patient factors on emergency department use of ambulance diversion” (p. 467). They collected data from an ED in Toronto, Ontario, Canada, and used time-series methods to determine the associations between diversions and nurse hours, physicians on duty, and boarded patients. Covariates in their study included patient volume, assessment time and boarding time. After analyzing 1,095 8-hour intervals of data, they concluded that “admitted patients in the ED are important determinants of ambulance diversion” (p. 467) whereas nurse and physician scheduling, and walk-in patients are not.

In their research, Shull et al. only considered the effects of ED related factors on ambulance diversions. There are also several studies that focus on hospital operations as well. The American Hospital Association (AHA) has conducted two surveys on 28 hospitals regarding ambulance diversions (TheLewinGroup 2002; TheLewinGroup 2004). They both concluded that the lack of critical care beds in hospitals is the major cause for ambulance diversions. Another survey conducted on 61 EDs by the Massachusetts College of Emergency Physicians also concluded that the lack of inpatient beds is the leading cause (72%) of ambulance diversions (Epstein and Slate 2001). Other causes of diversions included high acuity patients in the ED, staff shortages and overcrowded waiting rooms.

In the 2004 Society for Academic Emergency Medicine (SAEM) annual meeting, Stephen K. Epstein (2004) introduced an ED workscore to predict ambulance diversions. From this purely data based observational study, using a logistic regression model, weights for input, output, and throughput to the system were developed. In conclusion, Epstein suggested that the number of patients in the waiting room was not a factor when compared to the number of "boarders."

2.2 Queueing Networks

A queueing network is a collection of two or more nodes in which the traffic at the nodes have some level of interaction. In his work, J. R. Jackson (1957; 1963), concluded that at least for stationary models with Poisson arrivals and exponential service times, at equilibrium, the nodes of a queueing network act as if they are independent systems, allowing the joint probability distribution (2.1) of the states of these nodes to be obtained by the multiplication of the individual probability distributions of the states.

$$P(k_1, k_2, ..., k_M) = \prod_{i=1}^{M} P(k_i) \tag{2.1}$$

Several assumptions are necessary to classify a network as a Jackson Network.

(i) Inter-arrival times are random variables that are IID (identically and independently distributed) according to an exponential distribution.

(ii) Service times at each node in the network are IID random variables following an exponential distribution.

(iii) Probability of a customer moving from one node to another, denoted as the "routing probability" is a constant.

(iv) There are unlimited buffer capacities between nodes.

Due to finite buffers of the queues in our network (violation of assumption (iv) above), we cannot use the findings of Jackson directly. Such networks have been studied before

Jackson, by G. C. Hunt (1956). He investigated a network of single server queues in four separate cases.

Case 1. Infinite queues are allowed in front of each service facility.

Case 2. No queues are allowed, with the exception that the first stage may have an infinite queue.

Case 3. Finite queues are allowed in front of each stage, with the exception that the first stage may have an infinite queue.

Case 4. No queues and no vacant facilities are allowed, with the exception that the first stage may have an infinite queue; the line moves all at once, as a unit. This case may be called the "unpaced belt production line" case.

The second case (with two nodes) describes the simplest form of the ED-Hospital network that we will consider, and the two-node problem is specifically discussed in Hunt's work. He also developed exact solutions for the steady state probabilities and maximum utilization of the system.

Koizumi et al. (2002) investigated the bottlenecks in a mental health system through a queueing network model with blocking. The research focuses on blocking as a reason for congestion in the system and concludes that removal of bottlenecks in one section of the system may be the most cost-efficient way to reduce congestion in the system as a whole.

All of the research described above approaches the problem by investigating steady-state characteristics of the systems, and addresses changes in the design of the system such as staffing and resource allocation. However, by the nature of the ED (fluctuating arrival rates and long service times), a steady state may rarely be reached and maintained. The unpredictability of the ED requires an approach that will be able to respond to the short term changes. An alternative to steady state analysis is to model the network as a continuous-time Markov chain (CTMC) and form the transition probability function, given in (2.2) (Ross 1972).

$$P_{ij}(t) = P\left\{X\left(t+s\right)=j\,\middle|\,X\left(s\right)=i\right\} \tag{2.2}$$

The transition probability function denotes the probability that a system in state *i* at time *s*, will be in state *j* a time *t* later. Developing and using the transition probability function to predict the probability of gridlock occurring in a certain time given a current condition is an important element of this research.

2.3 Simulation

Developing analytical tools that exhibit the complexities of healthcare systems has always been a challenging task (Harrel and Price 2000). As a result, to investigate such systems, researchers have utilized simulation-based methods, which have proven to be very useful in the healthcare industry and EDs in particular.

Rosetti et al. (1999) and Samaha et al. (2003) both used simulation to evaluate the effects of various staffing schedules on the patient length of stay (LOS). Samaha et al. also considered additional factors including the presence of a resident for training, bed side registration, and use of a dedicated nurse practitioner for routine patients. Baesler et al. (2003) used simulation and design of experiments to determine the maximum capacity in an ED. Mahapatra et al. (2003) studied the implications of various alterna care unit

(ACU) working timeframes on patient LOS by applying these scenarios to a detailed representation of the ED.

Centeno et al. (2003) did not use simulation to test previously created scenarios. They incorporated a linear programming (LP) model into a simulation, allowing the LP to create optimal staffing schedules according to the conditions coming from the simulation model, and used the simulation model to investigate the effects of these schedules. Our use of simulation will be similar to that of Centeno et al. The queueing model will gather information from the simulation, only to signal the simulation in case of an imminent gridlock, so that the simulation can react in a way to attempt to prevent gridlock.

3 Research Description

3.1 Emergency Department Overview

The system in consideration is composed of an ED and part of a hospital. In the most basic representation, the system is a network of two queues. Figure 3-1 illustrates the three main parts of the system. The patients randomly arrive to the ED. Depending on the availability of the beds and staff in the ED, the patient is either admitted for diagnosis and treatment, or sent to a waiting room. The waiting room serves as a queue for the ED beds. The patients in the waiting room are prioritized according to the severity of their condition, and they are admitted into the ED on a first come first served basis with respect to their priorities. After being treated in the ED, the patient is either discharged (no further immediate treatment is necessary), or admitted to the hospital (There is also the possibility that the patient expires during these processes.) The second queue in the system is for hospital admissions. There are only a fixed number of critical care (CC) beds in the hospital. If a patient needs to be admitted and requires a CC bed, at least one CC bed in the hospital needs to be available, or the patient will be on hold until an occupied bed is freed. In this part of the system, the ED beds serve as a queue to the CC beds, and the CC beds are "blocking" the arrival stream from the ED.

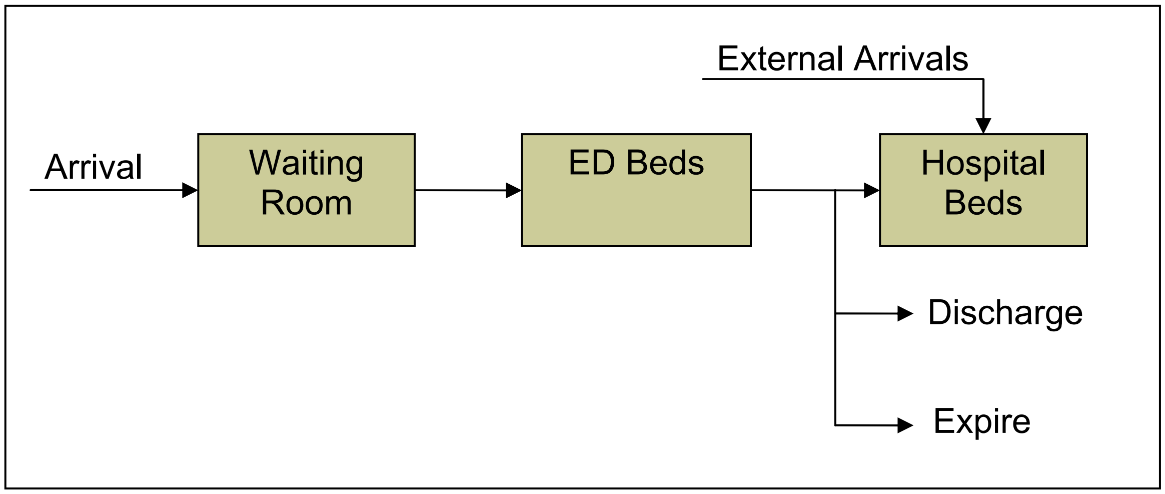


Figure 3-1 ED flowchart

The ED serves a time-dependent and multi-class arrival process. Upon entering, patients are classified according to the severity of their conditions. These classifications, called Emergency Severity Indexes (ESI), can have a number of levels depending on the healthcare provider. The most common are 3-level and 5-level ESI systems, where lower indexes denote higher severities. Different classes of patients tend to have different arrival rates that also vary according to the time of the day and day of the week.

An ED can enter gridlock by either an ED arrival, or an ED end-of-service. According to our definition, for the system to be in gridlock, the hospital and ED beds must be occupied with at least one boarding patient (waiting for a hospital bed while occupying an ED bed). An ED arrival occupying the last bed can put the system into a gridlock if the system had full hospital beds and boarding patients in the ED. An ED end-of-service can put the system in gridlock if the hospital beds are full, the ED beds are full, and the end-of-service just resulted in a boarding patient in the ED.

3.2 Research Procedure

3.2.1 Predictive Model

For the predictive model, a queueing model representing the ED-Hospital network is developed. However, the system in consideration is a very complex one to model thoroughly using queueing theory. Several assumptions were made in order to model the system.

1. *The only limiting resources in the system are the hospital and ED beds.*

   Almost all of the studies investigated in the literature agree that gridlock resulting from lack of hospital beds is the primary causes of ambulance diversions. This research intends to provide a methodology to tackle the gridlock problem under this assumption.

2. *There is only one type of bed in the ED and the hospital.*

   Many hospitals have divided their EDs into parts to serve certain ranges of ESIs. Patients in the most severe conditions (lower ESIs) are the most important sufferers and contributors to gridlocks. Thus, their arrival and service processes are isolated and investigated independent of the rest of the system.

3. *There is only one type of customer arriving at a constant arrival rate, distributed exponentially.*

   Gridlock and diversions usually occur during the busiest times of the ED, which are relatively short periods of time during the day (mostly nights and weekends). While arrival rates are most likely to fluctuate during a 24-hour period, a selected time having high arrival rates can be expected to have a constant arrival rate for a relatively short period of time.

4. *Outside arrivals to the hospital are suspended.*

   Outside arrivals to the hospital are scheduled patients, and they are not likely to be scheduled for a bed during the busiest time of the ED.

5. *Every customer leaving the ED must be admitted to the hospital.*

   Since patients with severe conditions are the main focus of this study, this research assumes the patients will need further care in a hospital bed after their treatment in the ED is complete.

During a visit, a patient can be in one of the following four sections in the system.

1. In the waiting room before the ED
2. Being treated in a bed in the ED
3. Waiting idle for a hospital bed while occupying a bed in the ED
4. Being treated in a bed in the hospital

A state space representing all these sections individually would need to be four dimensional, resulting in complex and time consuming calculations. Hunt's (1956) representation approach combines the first two positions, simplifying the state space. In this research, Hunt's (1956) approach, as shown in Table 3-1, is used to make the manipulations comparatively more manageable. The first element describing a state, n, represents the number of patients in the waiting room and the non-idle patients in the ED. The value of n ranges from 0 to n_{max} (waiting room capacity + total number of ED beds). The second element, i, denotes the number of idle patients in the ED, and the third element, h, gives the number of patients in the hospital.

Table 3-1 Description of the state naming approach

(n,i,h)	Description	Range
n	Patients in the waiting room and non-idle patients in the ED	0…n_{max} (n_{max} = # of ED beds + Waiting room capacity)
i	Idle patients occupying ED beds and waiting for a hospital bed (Patients on hold)	0…i_{max} (i_{max} = # of ED beds)
h	Patients in the hospital	0…h_{max} (h_{max} = # of hospital beds)

Not all of the states created using Hunt's naming approach are feasible. For a patient to be idle ($i > 0$), all the hospitals beds need to be full ($h = h_{max}$). Also, patients in the waiting room and the ED ($n+i$) can not be greater than the maximum capacity of the ED and the waiting room (n_{max}.) The total number of states is given in (3.1). For example a system with a 20 seat waiting room capacity, 10 ED beds and 4 critical care beds would have 410 feasible states.

$$\left(n_{\max}+1\right)\left(h_{\max}+1\right)+\sum_{k=n_{\max}-i_{\max}}^{n_{\max}-1}\left(k+1\right)=\left(n_{\max}+1\right)\left(h_{\max}+1\right)+\frac{i_{\max}\left(2n_{\max}-i_{\max}+1\right)}{2} \quad (3.1)$$

Consistent with our definition of gridlock, only the states with at least one idle ED bed ($i > 0$) and full ED beds ($n + i = i_{max}$) represent a gridlock. In the state space example provided in Figure 3-2, when there are no idle patients in the system, n reaches its maximum value (n_{max} = 7), for states with zero idle patients and any number of patients in the hospital (including full hospital). Only with a full hospital the system can have idle patients. Additionally, according to the rules of feasibility described above, ($n+i$) cannot exceed n_{max} = 7 in the rest of the state space where idle patients are present.

Waiting room capacity = 5

Number of ED beds = 2

Number of hospital beds = 2

$$n_{\max} = 7$$
$$i_{\max} = 2$$
$$h_{\max} = 2$$

$$\begin{array}{ccc} n & i & h \\ \hline \end{array}$$

$$\left.\begin{array}{l} \left.\begin{array}{ccc}0&0&0\\1&0&0\\..&..&..\\7&0&0\end{array}\right\}\text{Empty hospital }(h=0) \\ \left.\begin{array}{ccc}0&0&1\\1&0&1\\..&..&..\\7&0&1\end{array}\right\}h=1 \\ \left.\begin{array}{ccc}0&0&2\\1&0&2\\..&..&..\\7&0&2\end{array}\right\}\text{Full hospital }(h=2)\end{array}\right\}\text{No idle patients in the ED }(i=0)$$

$$\left.\begin{array}{l}\left.\begin{array}{l}\begin{array}{ccc}0&1&2\end{array}\\ \left.\begin{array}{ccc}1&1&2\\..&..&..\\6&1&2\end{array}\right\}\text{Gridlock}\end{array}\right\}i=1 \\ \left.\begin{array}{l}\begin{array}{ccc}0&2&2\end{array}\\ \left.\begin{array}{ccc}1&2&2\\..&..&..\\5&2&2\end{array}\right\}\text{Gridlock}\end{array}\right\}\text{ED beds are full and idle }(i=2)\end{array}\right\}(n+i)\text{ cannot exceed }n_{\max}$$

Figure 3-2 Example: State space

The predictive model calculates approximations to the cumulative distribution function (CDF) for the random variable T_{ij}, the first passage time to gridlock (state i) given an initial state (state *j*). T_{ij} has a continuous state space that extends from 0 to infinity. The CDF of T_{ij}, also called the transition probability function (2.2), can be interpreted as the probability of entering state *j* from state *i* in time τ. In order to calculate the first passage probabilities for a system, Kolmogorov's forward equations given in Figure 3-3 for the system are formed. Then the target state (state *j*) is transformed into an absorbing state (i.e. no departures from state *j*) by setting every element in the *j*th column to zero. Solution to this system of equations, with an initial value denoting the initial state *i*, is the CDF of T_{ij}. If we let *X(t)* denote the state of the system at time *t*, (3.2) illustrates the CDF of T_{ij}.

$$P(T_{ij} \le \tau) = P\left\{ X\left(t+s\right) = j \,\middle|\, X\left(s\right) = i, t \le \tau \right\} \tag{3.2}$$

With a generic predictive model, the system specifications are input in order to obtain gridlock probabilities. The completed predictive model, programmed in Visual Basic for Applications 6.3 (VBA), accepts values for ED and hospital arrival rates, ED and hospital service rates, ED and hospital bed capacities, and the waiting room capacity. Using the system specifications, the model generates a matrix representing the coefficients of the Kolmogorov forward equations that combine gridlock states into an absorbing super-state. The first step in the modification process is to add a state to represent the absorbing super-state. Then the rows of the gridlock states are zeroed to suspend exiting these states, making them absorbing. The rows of the absorbing super-state are then filled with the values in the respective rows of the gridlock states. The modification process is completed by zeroing the columns of the gridlock states. An example illustrating the modification process is given in Figure 3-4. Finally, this system of modified Kolmogorov forward equations, with an initial value vector representing the current state, is solved using the Runge-Kutta numerical integration algorithm. The Runge-Kutta integrator used for testing is called from a Component Object Model (COM) application, written and built in Matlab 7.0.1 (Appendix C). The details of this linkage between the programs are further discussed in section 3.2.3.

$$
\begin{aligned}
&(d/dt)P_{0\,0\,0} = -\lambda_{ED}P_{0\,0\,0} + \mu_H P_{0\,0\,1}\\
&(d/dt)P_{n\,0\,0} = -\left(\lambda_{ED} + \min\left(n, i_{\max}\right)\mu_{ED}\right)P_{n\,0\,0} + \mu_H P_{n\,0\,1} + \lambda_{ED}P_{n-1\,0\,1}\ ,\ 0 < n < n_{\max}\\
&(d/dt)P_{n_{\max}\,0\,0} = -\mu_{ED}P_{n_{\max}\,0\,0} + \lambda_{ED}P_{n_{\max}-1\,0\,0} + \mu_H P_{n_{\max}\,0\,1}\\
&(d/dt)P_{0\,0\,h} = -\left(\lambda_{ED} + h\mu_H\right)P_{0\,0\,h} + \mu_{ED}P_{1\,0\,h-1} + \left(h+1\right)\mu_H P_{0\,0\,h+1}\ ,\ 0 < h < h_{\max}\\
&(d/dt)P_{n\,0\,h} = -\left(\lambda_{ED} + \min\left(n, i_{\max}\right)\mu_{ED} + h\mu_H\right)P_{n\,0\,h} + \min\left(n+1, i_{\max}\right)\mu_{ED}P_{n+1\,0\,h-1}\\
&\qquad\qquad + \lambda_{ED}P_{n-1\,0\,h} + \left(h+1\right)\mu_H P_{n\,0\,h+1}\ ,\ 0 < n < n_{\max}, 0 < h < h_{\max}\\
&(d/dt)P_{n_{\max}\,0\,h} = -\left(n_{\max}\mu_{ED} + h\mu_H\right)P_{n_{\max}\,0\,h} + \lambda_{ED}P_{n_{\max}-1\,0\,h} + \left(h+1\right)\mu_H P_{n_{\max}\,0\,h+1}\\
&(d/dt)P_{0\,0\,h_{\max}} = -\left(\lambda_{ED} + h_{\max}\mu_H\right)P_{0\,0\,h_{\max}} + \mu_{ED}P_{1\,0\,h_{\max}-1} + h_{\max}\mu_H P_{0\,1\,h_{\max}}\\
&(d/dt)P_{n\,0\,h_{\max}} = -\left(\lambda_{ED} + \min\left(n, i_{\max}\right)\mu_{ED} + h_{\max}\mu_H\right)P_{n\,0\,h_{\max}} + \min\left(n+1, i_{\max}\right)\mu_{ED}P_{n+1\,0\,h_{\max}-1}\\
&\qquad\qquad + \lambda_{ED}P_{n-1\,0\,h_{\max}} + h_{\max}\mu_H P_{n\,1\,h_{\max}}\ ,\ 0 < n < n_{\max}\\
&(d/dt)P_{n_{\max}\,0\,h_{\max}} = -\left(\min\left(n_{\max}, i_{\max}\right)\mu_{ED} + h_{\max}\mu_H\right)P_{n_{\max}\,0\,h_{\max}} + \lambda_{ED}P_{n_{\max}-1\,0\,h_{\max}}\\
&\qquad\qquad + h_{\max}\mu_H P_{n_{\max}\,1\,h_{\max}}\\
&(d/dt)P_{0\,i\,h_{\max}} = -\left(\lambda_{ED} + h_{\max}\mu_H\right)P_{0\,i\,h_{\max}} + \mu_{ED}P_{1\,i-1\,h_{\max}} + h_{\max}\mu_H P_{0\,i+1\,h_{\max}}\ ,\ 0 < i < i_{\max}\\
&(d/dt)P_{n\,i\,h_{\max}} = -\left(\lambda_{ED} + \min\left(n, i_{\max} - i\right)\mu_{ED} + h_{\max}\mu_H\right)P_{n\,i\,h_{\max}} + \lambda_{ED}P_{n-1\,i\,h_{\max}}\\
&\qquad\qquad + \min\left(n+1, i_{\max} - i + 1\right)\mu_{ED}P_{n+1\,i-1\,h_{\max}} + h_{\max}\mu_H P_{n\,i+1\,h_{\max}}\ ,\ 0 < n < n_{\max}, 0 < i < i_{\max}\\
&(d/dt)P_{n_{\max}\,i\,h_{\max}} = -\left(\min\left(n_{\max}, i_{\max} - i\right)\mu_{ED} + h_{\max}\mu_H\right)P_{n_{\max}\,i\,h_{\max}} + \lambda_{ED}P_{n_{\max}-1\,i\,h_{\max}}\\
&\qquad\qquad + h_{\max}\mu_H P_{n_{\max}\,i+1\,h_{\max}}\ ,\ 0 < i < i_{\max}\\
&(d/dt)P_{0\,i_{\max}\,h_{\max}} = -\left(\lambda_{ED} + h_{\max}\mu_H\right)P_{0\,i_{\max}\,h_{\max}} + \mu_{ED}P_{1\,i_{\max}-1\,h_{\max}}\ ,\ 0 < i < i_{\max}\\
&(d/dt)P_{n\,i_{\max}\,h_{\max}} = -\left(\lambda_{ED} + h_{\max}\mu_H\right)P_{n\,i_{\max}\,h_{\max}} + \lambda_{ED}P_{n-1\,i_{\max}\,h_{\max}} + \mu_{ED}P_{n+1\,i_{\max}-1\,h_{\max}}\ ,\ 0 < n < n_{\max}\\
&(d/dt)P_{n_{\max}\,i_{\max}\,h_{\max}} = -\left(h_{\max}\mu_H\right)P_{n_{\max}\,i_{\max}\,h_{\max}} + \lambda_{ED}P_{n_{\max}-1\,i_{\max}\,h_{\max}}
\end{aligned}
$$

Figure 3-3 - Kolmogorov Forward Equations

The table below shows a 10-state M/M/1/10 queueing system with arrival rate λ and service rate μ and system capacity 10. Suppose states 5 and 9 are two states out of which we want to create an absorbing super-state (similar to our gridlock states).

	0	1	2	3	4	**5**	6	7	8	**9**	10
0	-λ	λ	0	0	0	**0**	0	0	0	**0**	0
1	μ	-(λ+μ)	λ	0	0	**0**	0	0	0	**0**	0
2	0	μ	-(λ+μ)	λ	0	**0**	0	0	0	**0**	0
3	0	0	μ	-(λ+μ)	λ	**0**	0	0	0	**0**	0
4	0	0	0	μ	-(λ+μ)	**λ**	0	0	0	**0**	0
5	**0**	**0**	**0**	**0**	**μ**	**-(λ+μ)**	**λ**	**0**	**0**	**0**	**0**
6	0	0	0	0	0	**μ**	-(λ+μ)	λ	0	**0**	0
7	0	0	0	0	0	**0**	μ	-(λ+μ)	λ	**0**	0
8	0	0	0	0	0	**0**	0	μ	-(λ+μ)	**λ**	0
9	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**μ**	**-(λ+μ)**	**λ**
10	0	0	0	0	0	**0**	0	0	0	**μ**	-μ

We first add a super-state (state 11), then make the rows of state 5 and 9 zeros. Next we add all the elements in state 5 and 9, and copy them to their respective rows in the super-state column. Finally, we make the columns of states 5 and 9 zeros.

	0	1	2	3	4	5	6	7	8	9	10	**11**
0	-λ	λ	0	0	0	0	0	0	0	0	0	**0**
1	μ	-(λ+μ)	λ	0	0	0	0	0	0	0	0	**0**
2	0	μ	-(λ+μ)	λ	0	0	0	0	0	0	0	**0**
3	0	0	μ	-(λ+μ)	λ	0	0	0	0	0	0	**0**
4	0	0	0	μ	-(λ+μ)	0	0	0	0	0	0	**λ**
5	0	0	0	0	0	0	0	0	0	0	0	**0**
6	0	0	0	0	0	0	-(λ+μ)	λ	0	0	0	**μ**
7	0	0	0	0	0	0	μ	-(λ+μ)	λ	0	0	**0**
8	0	0	0	0	0	0	0	μ	-(λ+μ)	0	0	**λ**
9	0	0	0	0	0	0	0	0	0	0	0	**0**
10	0	0	0	0	0	0	0	0	0	0	-μ	**μ**
11	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**

Figure 3-4 Example: Modification of Kolmogorov forward equation coefficients to include the absorbing super-state

3.2.2 Simulation Model

There are two purposes of the simulation model in this research; (i) to determine the quality of the predictions made by the predictive model, and (ii) to observe the effects of the preventive measures on gridlock probabilities. The simulation model shown in

Figure 3-5 is developed with the same specifications as the predictive model. Customers arrive to the ED and, at every arrival, a gridlock probability is calculated in the first VBA block. If a gridlock is imminent, the replication number is written to a text file and preventive measures are taken. Then the simulation model checks to see if the system is really in gridlock, and writes to the same text file in the case that it is. From then on, the patient is routed to the ED, waits or starts service, and is then transferred to the hospital if there is an available bed. If a gridlock is not imminent, the patient keeps the ED bed, waiting for a hospital bed to become available. The patient releases the ED bed only when a hospital bed becomes available. After being treated in the hospital, the patient is discharged.

The model simulates 8-hour segments of the day in which the arrival and service rates remain constant. Since a replication's role is to report whether a gridlock has occurred or not, in the event of a gridlock it is terminated. The predictive model in the first VBA block also assigns the system parameters and they are set from within the VBA code. The details of the linkage between the models and the testing procedures will be discussed in sections 3.2.3 and 3.2.5, respectively.

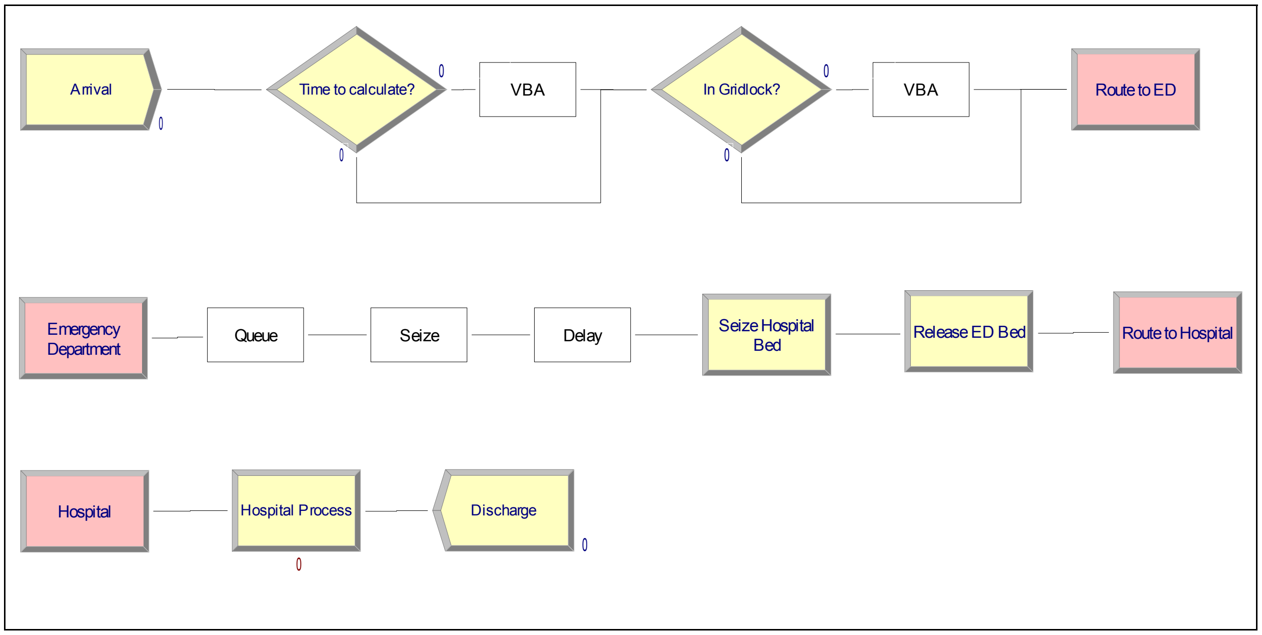


Figure 3-5 Arena 7.0 simulation model

3.2.3 Linking the Predictive Model to the Simulation

Figure 3-6 illustrates the linkage between the predictive model, the simulation and the Runge-Kutta integrator in Matlab. System specifications set in the predictive model are passed to the simulation during the beginning of a replication. As the simulation runs, after every arrival, it sends the current state of the system to the predictive model. The predictive model then generates the Kolmogorov forward equations and, along with the current state and, calls the Matlab COM application for a solution. The COM application solves the system using the 4-step Runge-Kutta integrator, sending the solution back to the predictive model, where the probability of gridlock is carried onto the simulation. If there is enough reason to believe that a gridlock is imminent, the simulation model modifies the system, trying to prevent the gridlock. During this process, each prediction and gridlock is recorded in a text file for analysis. Details of the testing are discussed in section 3.2.5.

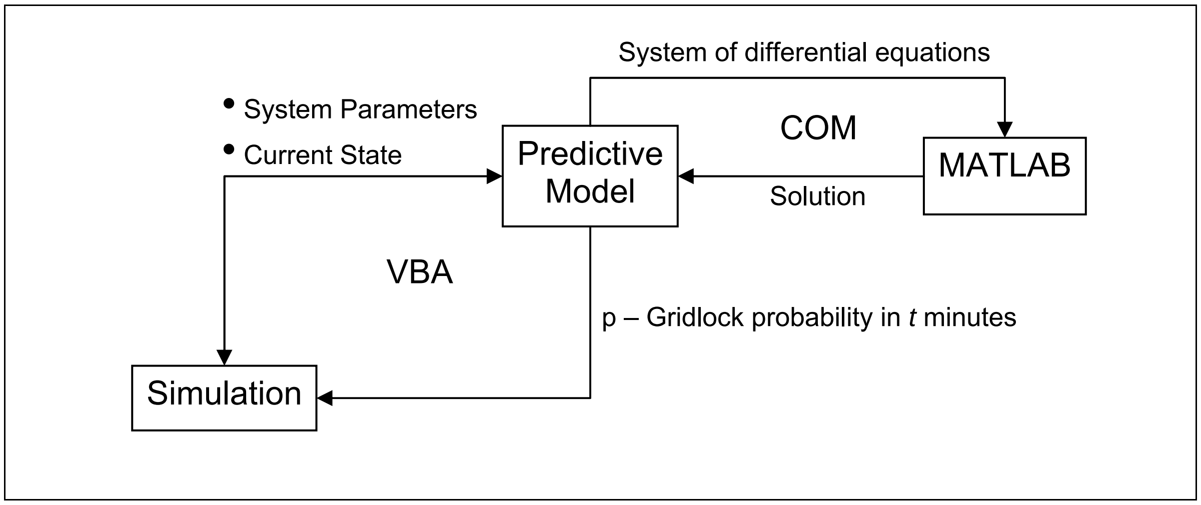


Figure 3-6 Arena-VBA-Matlab integration

3.2.4 Model Verification

To ensure that the predictive model is working properly, several attributes of the outputs were verified during the initial construction phase. The negative values in a row in the Kolmogorov forward equation coefficient matrix denote the total rate at which a particular state is left. The positive values in the same row denote the rates the system enters other states from the current state. Consequently, row sums of the generated coefficient matrices must be zero. The row sums must remain zero even after the addition of the absorbing super-state. The matrices created during the verification process were checked for this property after every generation and modification until we were confident that the generation procedures operate as expected. Communication between the Matlab module and the predictive model, and the accuracy of the probabilities generated by the predictive model were verified by comparing the model outputs with solutions obtained by manually entering the equations into a numerical integrator. Since the numerical integration we employ approximates the value of a CDF at a given time, the solutions must be bounded by 0 and 1 for positive time horizons, converging to 1 as the time horizon is increased. These bounds were also verified by graphing the solution and observing the convergence to 1.

3.2.5 Model Validation

To complete a thorough validation of the predictive and the simulation models, a resource providing historical data and insight would be necessary. Due to the lack of such a resource, we were only able to conduct face validity tests by observing the outputs from the models and comparing them to anecdotal information obtained from the literature. With a more detailed validation process, historical data with known gridlock instances can be simulated and effects of using the predictive model and preventive measures on real gridlocks and diversions can be investigated.

4 Predictive Model Performance Evaluation and Results

In this section, the testing procedure conducted to evaluate the predictive model and the preventive measures, and their results are discussed. First, the performance measures of interest are described. Then the systems used in testing the models are detailed, along with the testing procedures and results. Finally a short discussion of the results and their implications are presented.

4.1 Performance Measures

The purpose of the predictive model is to calculate the probability of a gridlock within a pre-determined time period. Then, according to the prediction, the simulation model increases the number of ED beds for the remainder of the replication in order to prevent the gridlock. Obviously, increasing the number of beds in the ED, even for a short period of time, would be a costly procedure. Hence, it would not be desirable to try preventing a gridlock that was wrongfully predicted. On the other hand, not being able to prevent a gridlock because of a poor prediction is not preferred. So the predictive model must be tested for both false alarms, and false "clears".

The performance of the preventive measures depends highly on the quality of the predictions. Therefore, rather than concentrating on the preventive measures alone, the performance of the prediction/prevention collaboration must be investigated. The major performance measure of interest is the effect of the preventive measures on gridlock probabilities.

4.2 Parameter Selection

Gridlock probability calculation involves selecting the values of two very crucial parameters; the time horizon (τ) that the probability is calculated for, and the alarm probability (p_a; the threshold probability, when exceeded, a gridlock alarm is given). Combinations of these variables affect the sensitivity of the predictive model. A highly sensitive predictor (large τ and low p_a) will foresee most of the gridlocks whereas a less sensitive predictor (small τ and high p_a) will result in more false "clears". Selection of these values will also affect the time a system has to react and try to prevent gridlock. For a flexible system that can take drastic measures in order to prevent gridlock, short time horizons will be sufficient, resulting in fewer false alarms. On the other hand, a system that requires longer times to react or cannot go to drastic changes will need a longer time horizon. We did not base the models on a real system. Additionally, deciding on how to prevent gridlocks is outside the scope of this research. The preventive measures are included in the investigation in order to test whether the short-term predictions made by the predictive model can be useful in preventing gridlock. For our testing purposes, p_a and τ were chosen through observation of pre-testing model runs. The only property that was sought in these values was that they employed the preventive measures one or two steps prior to a gridlock. Arbitrarily selected values, $p_a = 0.80$ and $\tau = 120$ minutes, provided acceptable predictive quality and were used in all of the testing.

York Hospital in York, Pennsylvania has been involved in academic research at Virginia Tech previously. A large database containing a detailed ED time study was used in a simulation study by Mahapatra et al. in 2003. The same dataset was used in this research to obtain patient arrival and service rates. The inter-arrival and service time data were analyzed using Arena's Input Analyzer. An exponential distribution with a mean of 49.2 was a very good fit for the inter-arrival times (Figure 4-1) of ESI 1 and 2 patients. For the service times, a Weibull(215, 1.56) distribution provided a very good fit (Figure 4-2), but for simplifying reasons, an exponential distribution with a mean 212 minutes is used (Figure 4-3). Since data regarding the hospital were not readily available, all other parameters in the tests were determined to resemble anecdotal information. The system parameters used in testing are summarized in Table 4-1.

Two systems with different gridlock probabilities[1] were used in testing the models. The system with the lower gridlock probability (~20%) had a mean inter-arrival time higher than that of the system with the higher gridlock probability (~%80).

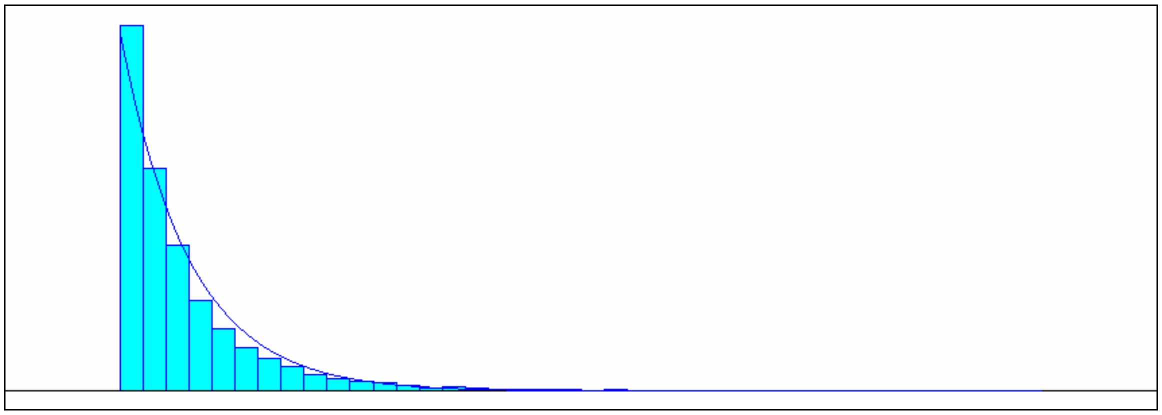


Figure 4-1 Inter-arrival times distribution fit – Expo(49.2), Chi-squared corresponding p-value < 0.005

[1] Proportion of time the system enters gridlock in consecutive 8-hour periods

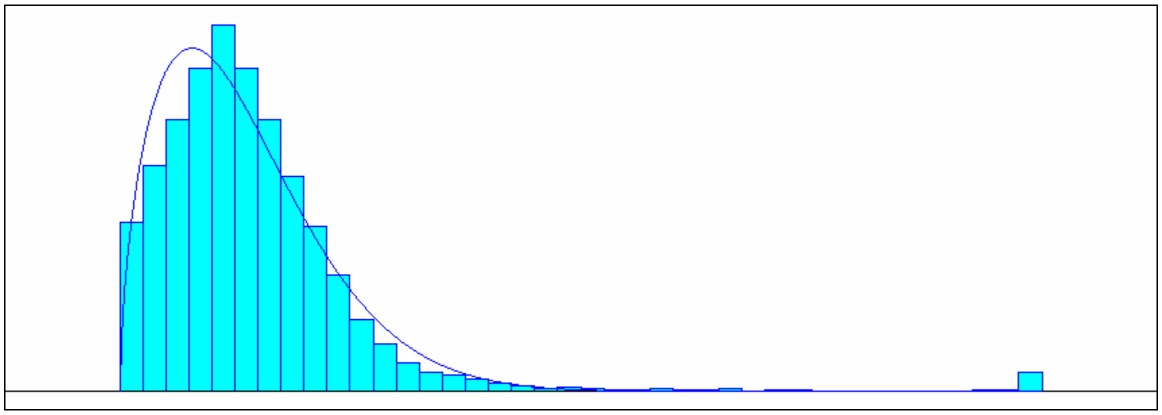


Figure 4-2 Service time distribution fit **-** Weibull(215, 1.56), Chi-squared corresponding p-value < 0.005

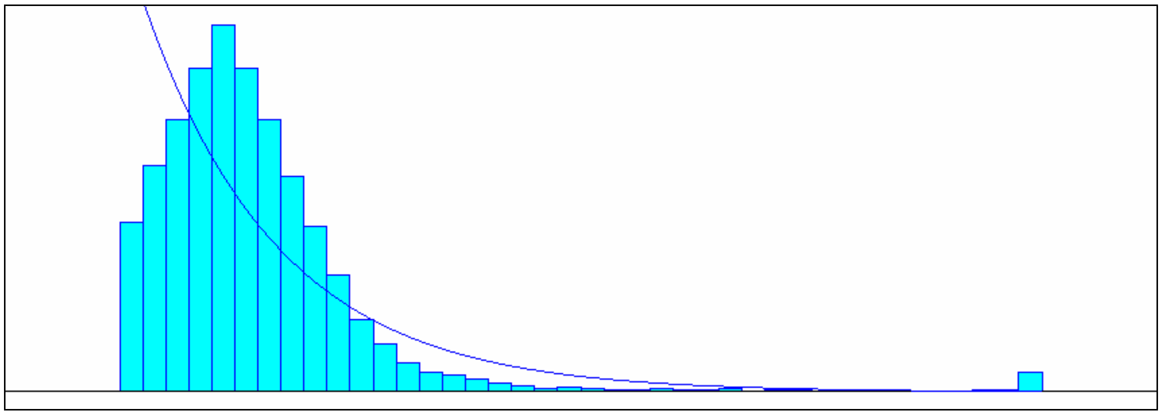


Figure 4-3 Service time distribution fit - Expo(212), Chi-squared corresponding p-value < 0.005

Table 4-1 System Parameters (all times are in minutes)

	System 1	**System 2**
Waiting Room Capacity	20	20
ED Capacity	10	10
Hospital Capacity	2	2
ED Mean Time Between Arrivals	49.2	30
ED Mean Service Time	212	212
Hospital Mean Service Time	1440	1440

Due to the increased rate of arrival in System 2, the ED beds fill up much quicker than System 1, resulting in gridlocks occurring more quickly. This eventually increases the probability of reaching a gridlock within the 8-hour replication period. Outputs from simulations running these two systems indicate whether or not a gridlock has occurred in a replication, and therefore can be represented by Bernoulli random variables shown in (3.3).

$$G_{s,r} = \begin{cases} 0, \text{ if system s did not enter gridlock in replication r} & s = \{1, 2\} \\ 1, \text{ if system s entered gridlock in replication r} & r = \{1, 2, \ldots, 1500\} \end{cases} \quad (3.3)$$

where $P\{G_{s,r} = 1\} = p_s^{(G)}$

Summing up n of these Bernoulli($p_s^{(G)}$) trials for each system will result in two Binomial($p_s^{(G)}, n$) trials. If n is sufficiently large, we can form a confidence interval for the binomial parameter $p_s^{(G)}$ for each of the systems, obtaining a confidence interval for the gridlock probability, by using the normal approximation in (3.4) (Agresti and Coull 1998).

$$\hat{p}_s^{(G)} \pm z_{\alpha/2}\sqrt{\hat{p}_s^{(G)}\left(1-\hat{p}_s^{(G)}\right)/n} \quad (3.4)$$

There are several suggestions on the value of n, the number of replications, in order to successfully apply the normal approximation. A skewed binomial distribution will converge to normal much slower than a symmetric one. A rather conservative suggestion for n is to choose one that satisfies $np(1-p) > 10$. Both our systems comply with this suggestion as can be seen from Table 4-2. These confidence intervals also create a base scenario that the improvements can be compared against in section 4.4.

Table 4-2 Confidence intervals for gridlock probabilities

	System 1	**System 2**
np(1-p)	270	229
95% Confidence Interval	0.235 ± 0.021	0.812 ± 0.020

4.3 Testing Procedures

The tests described in sections 4.3.1 and 4.3.2 are conducted for two systems (System 1 and System 2), with parameters given in Table 4-1.

4.3.1 Prediction Quality

To test the predictive model for false alarms, the simulation model is run for 1500 replications in conjunction with the predictive model without employing any preventive measures. At every ED arrival, the predictive model is called by the first VBA module in the simulation model. The VBA code for the predictive model is given in appendices A and B. If a gridlock is imminent ($P\{Time\ until\ gridlock \leq \tau\} > p_a$), the replication number is recorded into a text file. The same arrival then triggers a gridlock check, in which the system without that arrival is checked whether the conditions for gridlock are satisfied. If they are, the word “Gridlock” and the replication number are outputted in to the same text file as the predictions. Then the predicted gridlocks are compared to actual gridlocks, and a “1” is recorded for every correct prediction, while a “0” is recorded for every false one. To test the proportion of gridlocks that the predictive model failed to foresee, the same output is used. Only this time, at every gridlock that was predicted, a “1”, and at every gridlock that was not predicted, a “0” was recorded. The outputs from these tests are analyzed using the normal approximation to binomial as described in section 4.2, and 95% confidence intervals were formed to quantify the significance of our findings.

4.3.2 Preventive Measures

Three levels of preventive measures were applied to both systems. In the case of a gridlock probability exceeding the threshold, extra ED beds were made available for the remaining of the replication. Having a lower patient arrival rate, System 1 was tested for 1, 3 and 5 extra beds. For System 2, since adding 1 bed did not result in a significant difference, 3, 5 and 7 extra bed scenarios were created. Outputs collected from these scenarios were used to investigate the effectiveness of the preventive measures when used in conjunction with the predictive model. Each of the 1500 replications in the six scenarios created (3 per system) once again yielded a "1" if a gridlock occurred during the 8-hour run, and a "0" otherwise. For each scenario, 1500 Bernoulli trials constituted a Binomial experiment, to which the normal approximation to the binomial was applied, and 95% confidence intervals around the underlying gridlock probabilities were formed.

4.4 Summary and Discussion of Results

The predictive model was able to predict all of the gridlocks in both System 1 and System 2. However, almost 30% of predictions in System 1, and 12% of predictions in System 2 were false (detailed in Table 4-3). The values for *np(1-p)* were again well over 10, allowing normal approximation to be used.

Table 4-3 Confidence intervals for false alarm probability

	System 1	**System 2**
np(1-p)	101	147
95% Confidence Interval	0.285 ± 0.040	0.121 ± 0.017

The results of the simulation run testing the various preventive measures for both systems were also analyzed using a normal approximation. The gridlock probabilities obtained from these models with preventive measures are summarized in Table 4-4 and Table 4-5, and illustrated in Figure 4-4. The first row in the tables below restates the probability of gridlock with no prediction or prevention for comparison.

Table 4-4 System 1 gridlock probabilities with preventive scenarios

# of Extra ED Beds	np(1-p)	95% Confidence Interval
0	270	0.235 ± 0.021
1	202	0.161 ± 0.019
3	86	0.061 ± 0.012
5	33	0.022 ± 0.008

Table 4-5 System 2 gridlock probabilities with preventive scenarios

# of Extra ED Beds	np(1-p)	95% Confidence Interval
0	229	0.812 ± 0.020
3	364	0.587 ± 0.025
5	336	0.340 ± 0.024
7	224	0.183 ± 0.020

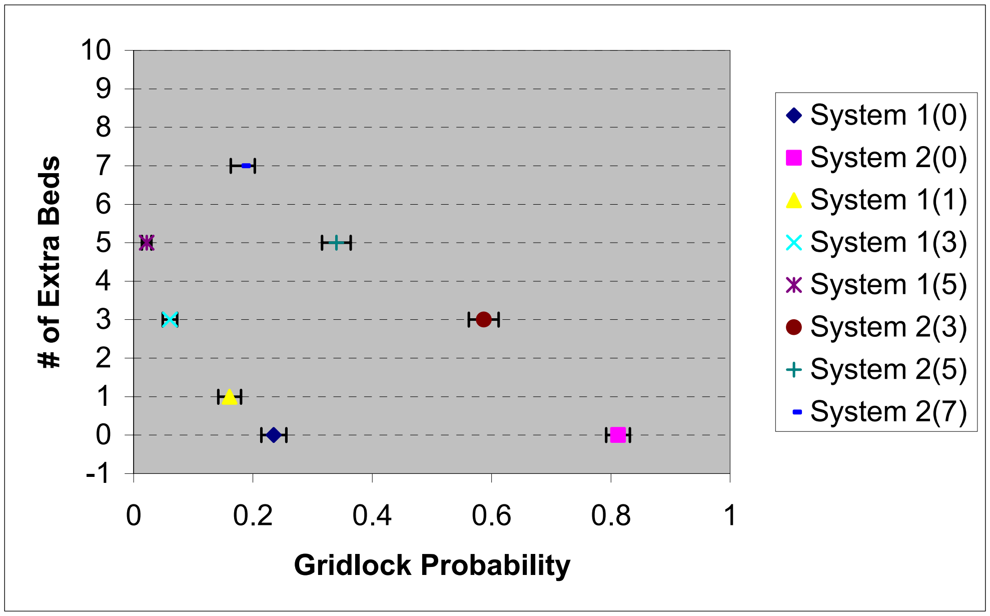


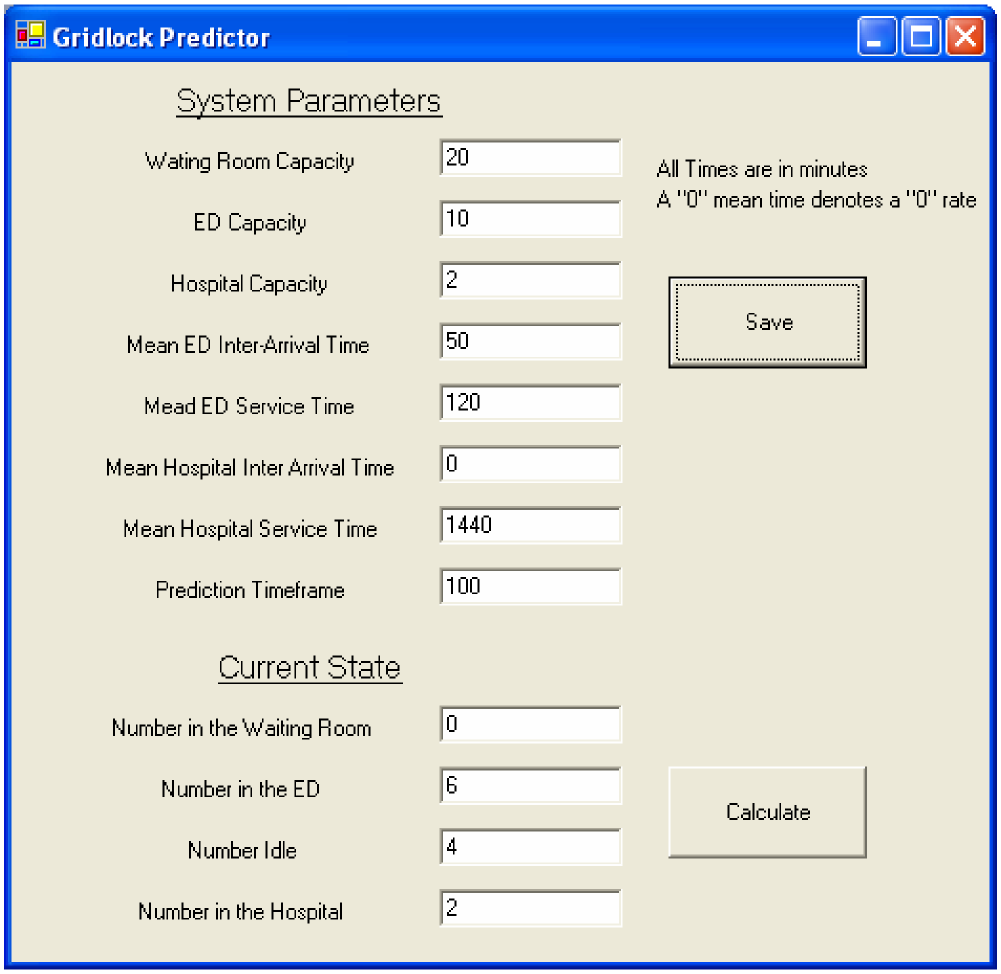
Figure 4-4 - Summary of gridlock probabilities. The error bars show 95% confidence intervals

While these improvements seem promising, enforcing timely preventive measures may be complicated. Staffing of the newly added beds may be challenging. When a gridlock alarm is given and a number of beds suddenly become available, the system somehow needs to find the necessary doctors and nurses to serve the newly opened beds. This may not be a major problem if there is idle staff due to boarding ED patients, however, if all of the current staff is fully utilized, the ED will need to find additional personnel in a timely fashion. Proper usage of the extra beds may also be difficult to impose. One can ask why the extra beds are not scheduled for use without prior prediction. To answer this question, we would need to compare the costs associated with pre-scheduling the extra bed usage and opening the beds according to a real-time prediction. Further investigation of these issues require additional communication and collaboration with a hospital, which along with other future work recommendations, will be discussed in section 6.

Although there may be challenges in utilizing our model in a real ED-hospital system due to our assumptions, our results show that, the percentage of gridlocks may be decreased if the model is successfully implemented, leading to less diversion-related service and financial performance inadequacies.

5 Visual Basic Tool

The final objective of this research is to create a tool so that the predictive model can be easily applied to any system complying with the assumptions in 3.2.1. The code for the tool created is given in appendices D and E. Once compiled, this tool can be used independent of any additional software. Ideally, such a tool can be incorporated in a compatible ED-hospital system, where system parameters and the current state can be entered in order to obtain gridlock probabilities. In our tests, the system increases the number of ED beds temporarily and tries to prevent the gridlock, whereas in the real world, even the existence of such information may be helpful.



System Parameters

Wating Room Capacity	20
ED Capacity	10
Hospital Capacity	2
Mean ED Inter-Arrival Time	50
Mead ED Service Time	120
Mean Hospital Inter Arrival Time	0
Mean Hospital Service Time	1440
Prediction Timeframe	100

All Times are in minutes
A "0" mean time denotes a "0" rate

Current State

Number in the Waiting Room	0
Number in the ED	6
Number Idle	4
Number in the Hospital	2

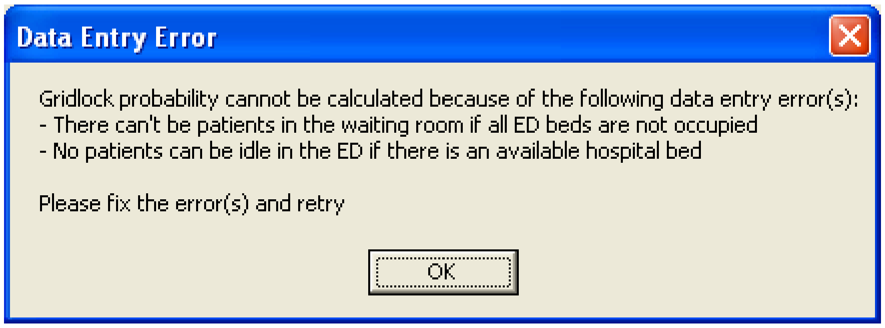
Figure 5-1 Screenshot of the gridlock prediction tool

Figure 5-1 shows a screenshot of the tool created in Visual Basic .Net. The values listed under “System Parameters” contain the parameters that describe the ED-hospital system, and the time horizon that the gridlock probability will be calculated for. All the mean times need to be entered in minutes, and a “0” mean inter-arrival time denotes no arrivals.

The first time the executable is run, it looks for a data file “systemparameters.dat” in the same directory as itself. If the data file is not found, one is created with default values. The user now must enter the parameters that describe the system in order to use the tool. Once the system parameters are entered, the user may choose to save them by clicking the “Save” button, so that they can be recalled when the program is run again. Otherwise, the last saved values will remain in the data file. Now the user must enter the current state of the system. There are certain rules that the user must abide in order to enter feasible states. These rules (listed below) are checked once the user clicks the “Calculate” button and, they keep the entered parameters within the boundaries of the predictive model’s assumptions discussed in section 3.2.1.

1. *There can’t be patients in the waiting room if all ED beds are not occupied.* Empty ED beds would be immediately filled with the patients in the waiting room.
2. *No patients can be idle in the ED if there is an available hospital bed.* In the event of an available hospital bed, the idle patients would immediately be admitted.
3. *Number of patients in the Waiting Room, ED, or the Hospital cannot exceed the pre-set capacities.*

The model currently does not check for negative or non-numeric values. They would be fairly easy to include, however, priority was given to logical errors that are harder to be noticed by the user. If the user enters an infeasible combination of values (e.g. exceeding bed capacities), the gridlock probability is not calculated, and a message similar to that in Figure 5-2 that describes the errors is shown.

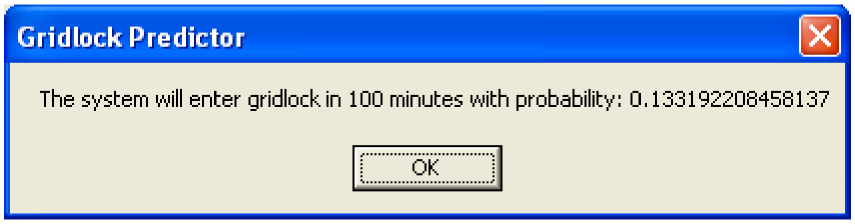

Data Entry Error

Gridlock probability cannot be calculated because of the following data entry error(s):
- There can't be patients in the waiting room if all ED beds are not occupied
- No patients can be idle in the ED if there is an available hospital bed

Please fix the error(s) and retry

Figure 5-2 Example: Infeasible state error

Upon successful data entry, the predictor creates the Kolmogorov forward equations describing the system, creates the initial state depending on the current state values, and sends them to the Runge-Kutta integrator in the Matlab COM application, returning the gridlock probability (Figure 5-3).


Gridlock Predictor

The system will enter gridlock in 100 minutes with probability: 0.133192208458137

Figure 5-3 Example: Gridlock prediction result

6 Conclusions, Recommendations and Future Work

When an emergency department (ED) decides that it is not going to be able to serve any more newly arriving patients, it declares a “diversion.” During a diversion, all or some of the controllable EMS transport units are denied and required to find alternative treatment facilities for their patients. This search causes both patients and EMS crew to lose valuable time. Past research suggests that the lack of critical care beds in the hospital is the primary contributor to ambulance diversion. When patients who need to be transferred from the ED to the hospital can not be moved due to the lack of an available bed in the hospital (i.e. the patient is boarding), they continue occupying their beds. While they are boarding in the ED, the associated staff is idle, and their bed cannot be used to treat other patients. Boarders in the ED lead to a gridlock state, which is defined as the situation when no new patient can be accepted to the ED until a hospital bed becomes available.

In this research, a predictive model that can calculate gridlock probabilities of a given system was developed. The model was combined with an Arena simulation and tested for prediction quality. Effects of using a predictive model and enforcing temporary preventive measures were investigated. Finally, the model was encapsulated into an executable computer program to enable its use as a prediction tool in a real world setting.

Our results indicate that through appropriate modeling, short-term gridlock predictions can be made using queueing theoretic approaches. These predictions provide probabilities of entering gridlock within a preset time horizon. The system is assumed to have constant arrival and service rates during 8-hour “high arrival rate” periods. The purpose of the predictions is to trigger preventive measures in order to try to exit the high arrival rate period without entering gridlock. To be effective, these preventive measures must be implemented quickly and temporarily. Preventive measures that are slower to implement will not be able to react to imminent gridlocks while permanent modifications in the system may be costly. In this research, the preventive measures we simulated were temporary increases in the number of ED beds. These measures took effect immediately after a gridlock alarm is issued and they were very effective in preventing gridlock. Two systems, one with low (~20% - System 1), and one with high (~80% - System 2) gridlock probability were simulated in conjunction with our predictive model and preventive measures. With only 3 additional beds, the probability of gridlock decreased to 6% in System 1 and 58% in System 2. With 5 additional beds, gridlocks in System 1 were almost eliminated while System 2 entered gridlock only 34% of the time. The addition of more beds would certainly eliminate gridlock. However, resource limitations which weren’t in this research need to be considered.

The most important limitation of this research was the unavailability of a real world system that the predictive model and the simulation could be based on. ED arrival and service rates were extracted from a database that was previously constructed for a simulation study on York Hospital in Pennsylvania. The next step following this research should include tighter collaborations with a hospital to gain more information about hospital procedures and policies. Historical data from a hospital would also allow more accurate modeling, validation using past system behavior, and a base scenario for comparison of alternatives. The predictive model can also be improved to accommodate multiple classes of patients and time dependent arrival rates, so that the model can be applied to systems without ED sections dedicated to ESI ranges. Following these improvements, the prediction quality and preventive measures can be tested in a real ED-hospital system.

This research focused primarily on predicting gridlock, the major reason for ambulance diversions. In other related studies to follow, the predictive model may be expanded to include other factors that cause diversions such as staff scheduling, and focus on predicting and preventing ambulance diversions rather than one particular cause. Declaring diversion in one hospital immediately increases the load on neighboring hospitals. Trying to expand this methodology to a network of hospitals aiming to decrease diversions in an area is yet another potential extension of our research.

7 References

Agresti, A., and Coull, B. A. (1998). "Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions." *The American Statistician*, 52(2), 119-126.

Asplin, B. R. (2003). "Does Ambulance Diversion Matter." *Annals of Emergency Medicine*, 41(4), 477-480.

Baesler, F. F., Jahnsen, H. E., and DaCosta, M. "The Use of Simulation and Design of Experiments for Estimating Maximum Capacity in an Emergency Room." *Winter Simulation Conference*, New Orleans, LA.

Bennett, R. M. (2003). "State of Emergency." Industrial Engineer, 50-55.

Brewer, S. (2002). "Study: clogged trauma care leads to deaths." Houston Chronicle, Houston, A27.

Centeno, M. A., Giachetti, R., Linn, R., and Ismail, A. M. "A Simulation-ILP Based Tool for Scheduling ER Staff." *Winter Simulation Conference*, New Orleans, LA.

Epstein, S. K. (2004). "Development of an Emergency Department Workscore to Predict Ambulance Diversion." *Academic Emergency Medicine*, 11(5), 484.

Epstein, S. K., and Slate, D. H. (2001). "The Massachusetts College of Emergency Physicians Ambulance Diversion Survey." *Academic Emergency Medicine*, 8(5), 526-527.

Harrel, C. R., and Price, R. N. "Healthcare Simulation Modeling and Optimization Using Medmodel." *Winter Simulation Conference*, Orlando, FL.

Henry, M. C. (2001). "Overcrowding in America's Emergency Departments: Inpatient Wards Replace Emergency Care." *Academic Emergency Medicine*, 8(2), 151-155.

Hunt, G. C. (1956). "Sequential Arrays of Waiting Lines." *Operations Research*, 4(6), 674-683.

Jackson, J. R. (1957). "Networks of Waiting Lines." *Operations Research*, 5(4), 518-521.

Jackson, J. R. (1963). "Jobshop-like Queueing Systems." *Management Science*, 10(1), 131-142.

Koizumi, N. (2002). "A Queueing Network Model with Blocking: Analysis of Congested Patients Flows in Mental Health Systems." University of Pensylvania.

Mahapatra, S., Koelling, C. P., Patvivatsiri, L., Fraticelli, B., Eitel, D., and Grove, L. "Pairing Emergency Severity Index 5-Level Triage Data with Computer Aided System Design to Improve Emergency Department Access and Throughput." *Winter Simulation Conference*, New Orleans, LA.

Ross, S. M. (1972). *Introduction to Probability Models*, Academic Press, San Diego.

Rossetti, M. D., Trzcinski, G. F., and Syverud, S. A. "Emergency Department Simulation and Determination of Optimal Attending Physician Staffing Schedules." *Winter Simulation Conference*, Phoenix, AZ.

Samaha, S., Armel, W. S., and Starks, D. W. "The Use of Simulation to Reduce the Length of Stay in Emergency Department." *Winter Simulation Conference*, New Orleans, LA.

Schull, M. J., Lazier, K., Vermeulen, M., Mawhinney, S., and Morrison, L. J. (2003). "Emergency Department Contributors to Ambulance Diversion: A Quantitative Analysis." *Annals of Emergency Medicine*, 41(4), 467-476.

TheLewinGroup. (2002). "Emergency Department Overload: A Growing Crisis Survey." AHA Survey Results, American Health Association.

TheLewinGroup. (2004). "Hospital Capacity and Emergency Department Diversion: Four Community Case Studies." American Health Association.

Appendix A
VBA Controller Code behind the Arena Model

```
Public NumberOfStates As Integer

Private Sub VBA_Block_2_Fire()

    Dim currentstate As Integer
    Dim probability As Double
    Dim NumberOfStates As Integer
    Dim finalstate As Integer

    Dim state_index() As Integer

    Dim s As SIMAN
    Set s = ThisDocument.Model.SIMAN

    'system specifications
    Dim N, I, H As Integer
    Dim lambda_ED, lambda_H, mu_ED, mu_H As Double
    Dim N_address, lambda_ED_address, lambda_H_address, mu_ED_address,
      mu_H_address As Integer
    Dim delta_t, forecast_time As Double
    '*********************************************
    'System parameters:
    '*********************************************
    forecast_time = 120


    N_address = s.SymbolNumber("ED Queue Capacity_N")
    I_address = s.SymbolNumber("ED Beds")
    H_address = s.SymbolNumber("Hospital Beds")
    lambda_ED_address = s.SymbolNumber("ED Arrival Rate_lambdaED")
    lambda_H_address = s.SymbolNumber("Hospital Arrival Rate_lambdaH")
    mu_ED_address = s.SymbolNumber("ED Service Rate_muED")
    mu_H_address = s.SymbolNumber("Hospital Service Rate_muH")
    ED_Queue_address = s.SymbolNumber("ED Bed Queue")
    H_Queue_address = s.SymbolNumber("Seize Hospital Bed.Queue")

    If s.RunCurrentTime = 0 Then
        numberofgridlocks = 0
        numberofalerts = 0

        N = 20
        I = 10
        H = 2
        lambda_ED = 1 / 30
        mu_ED = 1 / 212
        lambda_H = 0
        mu_H = 1 / 1440

        s.VariableArrayValue(N_address) = N
        s.ResourceCapacity(I_address) = I
        s.VariableArrayValue(lambda_ED_address) = lambda_ED
        s.VariableArrayValue(mu_ED_address) = mu_ED
        s.ResourceCapacity(H_address) = H
        s.VariableArrayValue(lambda_H_address) = lambda_H
        s.VariableArrayValue(mu_H_address) = mu_H

    End If

    N = s.VariableArrayValue(N_address)
    I = s.ResourceCapacity(I_address)
    lambda_ED = s.VariableArrayValue(lambda_ED_address)
    mu_ED = s.VariableArrayValue(mu_ED_address)
    H = s.ResourceCapacity(H_address)
    lambda_H = s.VariableArrayValue(lambda_H_address)
    mu_H = s.VariableArrayValue(mu_H_address)


    Dim system1 As New SystemOfEquations

    system1.SetSystemSpecs(N, I, H, lambda_ED, lambda_H, mu_ED) = mu_H

    'calculate current state
    If I <= s.ResourceNumberBusy(I_address) And _
        s.QueueNumberOfEntities(H_Queue_address) > 0 And
s.ResourceNumberBusy(H_address) = H Then

        currentstate = system1.numstates - 1
    Else
        currentstate =
system1.getIndex(s.QueueNumberOfEntities(ED_Queue_address) +
s.ResourceNumberBusy(I_address) -
s.QueueNumberOfEntities(H_Queue_address), _
                        s.QueueNumberOfEntities(H_Queue_address),
s.ResourceNumberBusy(H_address))
    End If

    Call system1.Final_Generate

    NumberOfStates = system1.numstates()

    Dim InitialValuesArray() As Double
    ReDim InitialValuesArray(NumberOfStates - 1)
    InitialValuesArray(currentstate) = 1


    finalstate = system1.numstates

    Dim Dummy As Variant
    If s.RunCurrentReplication >= 0 Then
       probability = system1.MatlabSolveDE(0, forecast_time,
InitialValuesArray)
    End If

    If probability > 0.8 Then
        'Increase ED bed capacity
        s.ResourceCapacity(s.SymbolNumber("ED Beds")) =
s.ResourceCapacity(s.SymbolNumber("ED Beds")) + 7
        s.VariableArrayValue(s.SymbolNumber("Gridlock Alarm")) = 1
       Open "Gridlocks_Preventive.txt" For Append As #1
        Print #1, s.RunCurrentReplication; " ";
s.VariableArrayValue(s.SymbolNumber("Gridlock Alarm")); " ";
s.RunCurrentTime
        Close #1
    End If


   Call system1.EraseSystem

End Sub

Private Sub VBA_Block_3_Fire()
    Dim s As SIMAN
    Set s = ThisDocument.Model.SIMAN
    Open "Gridlocks_Preventive.txt" For Append As #1    ' Open file for
output.
    Print #1, "Gridlock"; " "; s.RunCurrentReplication; " ";
s.RunCurrentTime
    Close #1


    s.VariableArrayValue(s.SymbolNumber("Gridlock Indicator")) = 1

End Sub
```

Appendix B
VBA Code for the SystemOfEquations Class

```
'Class properties

Private NumberOfStates As Integer
Private CoefficientMatrix() As Double
Private stateIndex() As Integer
Private N, I, H As Integer
Private lambda_ED, lambda_H, mu_ED, mu_H As Double

'Set system specifications
Public Property Let SetSystemSpecs(ByVal set_N As Integer, ByVal set_I
As Integer, ByVal set_H As Integer, ByVal set_lambda_ED As Double,
ByVal set_lambda_H As Double, ByVal set_mu_ED As Double, ByVal set_mu_H
As Double)

    N = set_N
    I = set_I
    H = set_H
    lambda_ED = set_lambda_ED
    lambda_H = set_lambda_H
    mu_ED = set_mu_ED
    mu_H = set_mu_H

    'calculate number of states
    NumberOfStates = (N + I + 1) * (H + 1)
    idle_count = 1
    Do While idle_count <= I
        NumberOfStates = NumberOfStates + ((N + I) - idle_count + 1)
        idle_count = idle_count + 1
    Loop

    NumberOfStates = NumberOfStates + 1
    ReDim CoefficientMatrix(NumberOfStates - 1, NumberOfStates - 1)
    ReDim RungeKuttaSolution(NumberOfStates - 1)
    ReDim stateIndex(NumberOfStates - 2, 3)

    'create stateIndex
    state_count = 0
    n_count = 0
    i_count = 0
    h_count = 0

    Do While h_count <= H
        Do While n_count <= N + I
            stateIndex(state_count, 0) = state_count
            stateIndex(state_count, 1) = n_count
            stateIndex(state_count, 2) = i_count
            stateIndex(state_count, 3) = h_count

            n_count = n_count + 1
            state_count = state_count + 1
        Loop
        n_count = 0
        h_count = h_count + 1
    Loop

    i_count = i_count + 1
    n_count = 0
    h_count = H
    Do While i_count <= I
```

```
        Do While (n_count + i_count) <= N + I
            stateIndex(state_count, 0) = state_count
            stateIndex(state_count, 1) = n_count
            stateIndex(state_count, 2) = i_count
            stateIndex(state_count, 3) = h_count

            n_count = n_count + 1
            state_count = state_count + 1
        Loop

        n_count = 0
        i_count = i_count + 1
    Loop

End Property

Public Property Get numstates() As Integer
    numstates = NumberOfStates
End Property


'returns the minimum of the two values that are passed in
Public Function min(ByVal first_value As Double, ByVal second_value As
Double)
    If first_value < second_value Then
        min = first_value
    Else
        min = second_value
    End If
End Function
'Generates the final model coefficient matrix
Public Function Final_Generate()

    'Initialize the coefficientmatrix
    eqn_counter = 0
    coef_counter = 0

    Do While eqn_counter < NumberOfStates
        Do While coef_counter < NumberOfStates
            CoefficientMatrix(eqn_counter, coef_counter) = 0
            coef_counter = coef_counter + 1
        Loop
        eqn_counter = eqn_counter + 1
    Loop


    'Fill the row for state (0,0,0)
    CoefficientMatrix(0, 0) = (lambda_ED + lambda_H) * (-1)
    CoefficientMatrix(0, getIndex(0, 0, 1)) = mu_H

    'Fill the rows for states (n,i,h) where,
    '0<n<N, i=0, 0<h<H

    For h_counter = 1 To H - 1
        For n_counter = 1 To (N + I - 1)
            this_state = (h_counter * (N + I + 1)) + n_counter
            CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter, 0, h_counter)) = (lambda_ED + lambda_H + min(I,
n_counter) * mu_ED + h_counter * mu_H) * (-1)
            CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter - 1, 0, h_counter)) = lambda_ED
            CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter, 0, h_counter - 1)) = lambda_H
            CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter, 0, h_counter + 1)) = (h_counter + 1) * mu_H
```

```
            CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter + 1, 0, h_counter - 1)) = min(I, n_counter + 1) *
mu_ED
        Next n_counter

        'n=0 i=0 0<h<H
        CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(0, 0,
h_counter)) = (-1) * (lambda_ED + lambda_H + h_counter * mu_H)
        CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(0, 0,
h_counter - 1)) = lambda_H
        CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(0, 0,
h_counter + 1)) = (h_counter + 1) * mu_H
        CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(1, 0,
h_counter - 1)) = mu_ED

        'n=N+I i=0 0<h<H
        CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N +
I, 0, h_counter)) = (-1) * (I * mu_ED + lambda_H + h_counter * mu_H)
        CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N + I
- 1, 0, h_counter)) = lambda_ED
        CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N +
I, 0, h_counter - 1)) = lambda_H
        CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N +
I, 0, h_counter + 1)) = (h_counter + 1) * mu_H



    Next h_counter

    '0<n<N+I

    For n_counter = 1 To N + I - 1
        '0<n<N+I, i=0, h=0
        CoefficientMatrix(getIndex(n_counter, 0, 0),
getIndex(n_counter, 0, 0)) = (-1) * (lambda_ED + lambda_H + min(I,
n_counter) * mu_ED)
        CoefficientMatrix(getIndex(n_counter, 0, 0),
getIndex(n_counter, 0, 1)) = mu_H
        CoefficientMatrix(getIndex(n_counter, 0, 0), getIndex(n_counter
- 1, 0, 0)) = lambda_ED

        '0<n<N+I, i=0, h=H
        CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter, 0, H)) = (-1) * (lambda_ED + min(I, n_counter) *
mu_ED + H * mu_H)
        CoefficientMatrix(getIndex(n_counter, 0, H), getIndex(n_counter
- 1, 0, H)) = lambda_ED
        CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter, 0, H - 1)) = lambda_H
        CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter, 1, H)) = H * mu_H
        CoefficientMatrix(getIndex(n_counter, 0, H), getIndex(n_counter
+ 1, 0, H - 1)) = min(I, n_counter + 1) * mu_ED


    Next n_counter

    'n=N+I, i=0, h=H

    CoefficientMatrix(getIndex(N + I, 0, H), getIndex(N + I, 0, H)) = -
(I * mu_ED + H * mu_H)
    CoefficientMatrix(getIndex(N + I, 0, H), getIndex(N + I - 1, 0, H))
= lambda_ED
    CoefficientMatrix(getIndex(N + I, 0, H), getIndex(N + I, 0, H - 1))
= lambda_H
```

```
    'n=N+I, i=0, h=0

    CoefficientMatrix(getIndex(N + I, 0, 0), getIndex(N + I, 0, 0)) = -
(lambda_H + I * mu_ED)
    CoefficientMatrix(getIndex(N + I, 0, 0), getIndex(N + I - 1, 0, 0))
= lambda_ED
    CoefficientMatrix(getIndex(N + I, 0, 0), getIndex(N + I, 0, 1)) =
mu_H


    'n=0, i=0, h=H

    CoefficientMatrix(getIndex(0, 0, H), getIndex(0, 0, H)) = (-1) *
(lambda_ED + H * mu_H)
    CoefficientMatrix(getIndex(0, 0, H), getIndex(1, 0, H - 1)) = mu_ED
    CoefficientMatrix(getIndex(0, 0, H), getIndex(0, 0, H - 1)) =
lambda_H
    CoefficientMatrix(getIndex(0, 0, H), getIndex(0, 1, H)) = H * mu_H

    'n=N+I, 0<i<=I, h=H

    For i_counter = 1 To I - 1

        '0<n<N+I-i, 0<i<=I, h=H
        For n_counter = 1 To (N + I - i_counter)

            If i_counter > 0 And I <= n_counter + i_counter Then

                CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter - 1, i_counter, H)) = _
                CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter - 1, i_counter, H)) + _
                lambda_ED

                CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter + 1, i_counter - 1, H)) = _
                CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter + 1, i_counter - 1, H)) + _
                min(I - (i_counter - 1), n_counter + 1) * mu_ED

                If (n_counter + i_counter) < N + I Then
                    CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter, i_counter + 1, H)) = _
                    CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter, i_counter + 1, H)) + _
                    H * mu_H
                End If

            Else
                CoefficientMatrix(getIndex(n_counter, i_counter, H),
getIndex(n_counter, i_counter, H)) = -(lambda_ED + H * mu_H + min(I -
i_counter, n_counter) * mu_ED)
                CoefficientMatrix(getIndex(n_counter, i_counter, H),
getIndex(n_counter - 1, i_counter, H)) = lambda_ED
                CoefficientMatrix(getIndex(n_counter, i_counter, H),
getIndex(n_counter + 1, i_counter - 1, H)) = min(I - (i_counter - 1),
n_counter + 1) * mu_ED
                If (n_counter + i_counter) < N + I Then
                    CoefficientMatrix(getIndex(n_counter, i_counter,
H), getIndex(n_counter, i_counter + 1, H)) = H * mu_H
                End If
            End If
        Next n_counter
        'n=N+I, 0<i<=I, h=H

        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) = _
        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) + _
        lambda_ED

        'n=0, 0<i<=I, H

        CoefficientMatrix(getIndex(0, i_counter, H), getIndex(0,
i_counter, H)) = -1 * (lambda_ED + H * mu_H)
        CoefficientMatrix(getIndex(0, i_counter, H), getIndex(1,
i_counter - 1, H)) = mu_ED
        CoefficientMatrix(getIndex(0, i_counter, H), getIndex(0,
i_counter + 1, H)) = H * mu_H
    Next i_counter

    i_counter = I
    For n_counter = 1 To (N + I - I)
            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter -
1, i_counter, H)) = _
            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter -
1, i_counter, H)) + _
            lambda_ED

            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter +
1, i_counter - 1, H)) = _
            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter +
1, i_counter - 1, H)) + _
            min(I - (i_counter - 1), n_counter + 1) * mu_ED
    Next n_counter

        'n=N+I, i=I, h=H

        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) = _
        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) + _
        lambda_ED

        'n=0, i=I, H

        CoefficientMatrix(NumberOfStates - 1, getIndex(1, i_counter -
1, H)) = _
        CoefficientMatrix(NumberOfStates - 1, getIndex(1, i_counter -
1, H)) + _
        mu_ED


End Function
Public Function CheckCoefficients()
    sum = 0
    For col = 0 To NumberOfStates - 1
        For row = 0 To NumberOfStates - 1
           sum = sum + CoefficientMatrix(row, col)
        Next row
        sum = 0
    Next col

End Function

'Returns the index corresponding to the state with n,i and h
Public Function getIndex(ByVal get_n As Integer, ByVal get_i As
Integer, ByVal get_h As Integer)
```

```
    Dim Found As Boolean
    Found = False

    state_count = 0


    Do While (state_count < NumberOfStates - 1) And Not (Found)
        If stateIndex(state_count, 2) = get_i Then
            Do While (state_count < NumberOfStates - 1) And Not (Found)
                If stateIndex(state_count, 3) = get_h Then
                    Do While (state_count < NumberOfStates - 1) And Not
(Found)
                        If stateIndex(state_count, 1) = get_n Then
                            Found = True
                        Else
                            state_count = state_count + 1
                        End If

                    Loop

                Else
                    state_count = state_count + 1
                End If
            Loop
        Else
            state_count = state_count + 1
        End If
    Loop

    If state_count = NumberOfStates - 1 And Not (Found) Then
        MsgBox "State n = " & get_n & " i = " & get_i & " h = " & get_h
& " is not found"
    End If


getIndex = state_count

End Function
Public Function MatlabSolveDE(ByVal t0 As Double, ByVal tf As Double,
ByRef initialvalues() As Double) As Double

    Dim Solver As New matlabSolutionclass
    Dim Dummy As Variant
    Dim solution As Variant


    Call Solver.setCoefficients(1, Dummy, CoefficientMatrix)
    Call Solver.odesolver(1, solution, t0, tf, initialvalues)

    MatlabSolveDE = solution

    Call Solver.clearmatlab(1, Dummy)

End Function


Public Function EraseSystem()
    Erase CoefficientMatrix
    Erase stateIndex
End Function
```

Appendix C
Matlab Numerical Integration Functions

```
%ODESolver function odesolver.m
%Solves the system of differential equations and returns the gridlock
probability
function solution = odesolver(t0, tf, initialvalues)
global global_coefficients;
[t, Y] = ode45('coefcalculate', [t0 tf], initialvalues);
solution = Y(size(Y,1), size(Y,2));

%SetCoefficients function setCoefficients.m
%Takes in a 2 dimensional array and stores it in a global matrix
function set = setCoefficients(toSet)
global global_coefficients;
global_coefficients = toSet;
clear toSet
set = 1

%CoefCalculate function coefcalculate.m
%Calculates the value of a function at a given point
function solution = coefcalculate(t , values)
global global_coefficients
solution = global_coefficients * values;
clear values

%ClearMatlab function clearmatlab.m
%Clears memory allocated by Matlab functions
function cleared = clearmatlab()
global global_coefficients
clear all
cleared = 1;
```

Appendix D
Controller Code for the VB.Net Gridlock Predictor Tool

```
Imports System
Imports System.IO
Public Class GridlockPredictor
Inherits System.Windows.Forms.Form

Dim WRCap, EDCap, HCap As Integer
    Dim EDArr, EDSrv, HArr, HSrv, TmFrm As Double

    Private Sub GridlockPredictor_Load(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles MyBase.Load

        Try
            Dim sr As StreamReader = New
StreamReader("systemparameters.dat")
            ' Read and display the lines from the file
            WRCap = sr.ReadLine()
            EDCap = sr.ReadLine()
            HCap = sr.ReadLine()
            EDArr = sr.ReadLine()
            EDSrv = sr.ReadLine()
            HArr = sr.ReadLine()
            HSrv = sr.ReadLine()
            TmFrm = sr.ReadLine()
            sr.Close()
        Catch ex As Exception
            ' Let the user know what went wrong.
            MsgBox(ex.Message & Chr(13) & "The data file will be
created and default values will be used")
            ' Create an instance of StreamWriter to write text to a
file.
            Dim sw As StreamWriter = New
StreamWriter("systemparameters.dat")
            ' Add some text to the file.
            Dim i As Integer
            For i = 1 To 7
                sw.WriteLine("0")
            Next
            sw.WriteLine("120")
            sw.Close()
            WRCap = 0
            EDCap = 0
            HCap = 0
            EDArr = 0
            EDSrv = 0
            HArr = 0
            HSrv = 0
            TmFrm = 120
        End Try

        'Set the values in the form to values from the data file
        WaitingRoomCapacity.Text = WRCap
        EDCapacity.Text = EDCap
        HospitalCapacity.Text = HCap
        MeanEDArrival.Text = EDArr
        MeanEDService.Text = EDSrv
        MeanHospitalArrival.Text = HArr
        MeanHospitalService.Text = HSrv
        PredictionTimeframe.Text = TmFrm
        NumberWaiting.Text = 0
        NumberED.Text = 0
        NumberIdle.Text = 0
        NumberHospital.Text = 0

    End Sub

    Private Sub SaveButton_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles SaveButton.Click
        Dim sw As StreamWriter = New
StreamWriter("systemparameters.dat")
        sw.WriteLine(WaitingRoomCapacity.Text)
        sw.WriteLine(EDCapacity.Text)
        sw.WriteLine(HospitalCapacity.Text)
        sw.WriteLine(MeanEDArrival.Text)
        sw.WriteLine(MeanEDService.Text)
        sw.WriteLine(MeanHospitalArrival.Text)
        sw.WriteLine(MeanHospitalService.Text)
        sw.WriteLine(PredictionTimeframe.Text)

        sw.Close()
    End Sub

    Private Sub CalculateButton_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles CalculateButton.Click
        Dim gridprob As Double
        Dim EqnSystem As New SystemOfEquations
        Dim N, I, H, NumberInWR, NumberInED, NumberInIdle, NumberInH,
currentstate As Integer
        Dim lambdaED, lambdaH, muED, muH, timeframe As Double
        N = System.Convert.ToInt16(WaitingRoomCapacity.Text) +
System.Convert.ToInt16(EDCapacity.Text)
        I = System.Convert.ToInt16(EDCapacity.Text)
        H = System.Convert.ToInt16(HospitalCapacity.Text)
        If System.Convert.ToDouble(MeanEDArrival.Text) > 0 Then
            lambdaED = 1 / System.Convert.ToDouble(MeanEDArrival.Text)
        Else
            lambdaED = 0
        End If

        If System.Convert.ToDouble(MeanEDService.Text) > 0 Then
            muED = 1 / System.Convert.ToDouble(MeanEDService.Text)
        Else
            muED = 0
        End If

        If System.Convert.ToDouble(MeanHospitalArrival.Text) > 0 Then
            lambdaH = 1 /
System.Convert.ToDouble(MeanHospitalArrival.Text)
        Else
            lambdaH = 0
        End If

        If System.Convert.ToDouble(MeanHospitalService.Text) > 0 Then
            muH = 1 / System.Convert.ToDouble(MeanHospitalService.Text)
        Else
            muH = 0
        End If

        timeframe = System.Convert.ToDouble(PredictionTimeframe.Text)

        NumberInWR = System.Convert.ToInt16(NumberWaiting.Text)
        NumberInED = System.Convert.ToInt16(NumberED.Text)
        NumberInIdle = System.Convert.ToInt16(NumberIdle.Text)
        NumberInH = System.Convert.ToInt16(NumberHospital.Text)
        'Check if current state is feasible
        Dim Valid As Boolean
        Valid = True

        Dim errormessage As String



        If NumberInED < I And NumberInWR > 0 Then
            Valid = False
            errormessage = errormessage & "- There can't be patients in
the waiting room if all ED beds are not occupied" & Chr(13)
        End If

        If NumberInED > I Or NumberInIdle > I Then
            Valid = False
            errormessage = errormessage & "- Number of patients in ED
is greater that ED capacity" & Chr(13)
        End If

        If NumberInWR > N - I Then
            Valid = False
            errormessage = errormessage & "- Number of patients in the
waiting room is greater than waiting room capacity" & Chr(13)
        End If

        If NumberInIdle > 0 And NumberInH < H Then
            Valid = False
            errormessage = errormessage & "- No patients can be idle in
the ED if there is an available hospital bed" & Chr(13)
        End If

        If NumberInH > H Then
            Valid = False
            errormessage = errormessage & "- Number of patients in the
hospital is greater than the hospital capacity" & Chr(13)

        End If


        If Valid Then
            Call EqnSystem.SetSystemSpecs(N, I, H, lambdaED, lambdaH,
muED, muH)
            Call EqnSystem.Final_Generate()



            'calculate current state
            If I <= NumberInED And _
                NumberInIdle > 0 And NumberInH = H Then

                currentstate = EqnSystem.numstates - 1
            Else
                currentstate = EqnSystem.getIndex(NumberInWR +
NumberInED - NumberInIdle, _
                                NumberInIdle, NumberInH)
            End If

            Dim InitialValuesArray() As Double
            ReDim InitialValuesArray(EqnSystem.numstates - 1)
            InitialValuesArray(currentstate) = 1
            gridprob = EqnSystem.MatlabSolveDE(0, timeframe,
InitialValuesArray)
```

```
            MsgBox("The system will enter gridlock in " & timeframe & "
minutes with probability: " & gridprob)
        Else
            MsgBox("Gridlock probability cannot be calculated because
of the following data entry error(s):" & Chr(13) & errormessage & _
            Chr(13) & "Please fix the error(s) and retry", , "Data
Entry Error")
        End If
End Sub
```

Appendix E
Updated SystemOfEquations Class for VB.Net

```
Public Class SystemOfEquations
    'Class Properties
    Private NumberOfStates, NumberOfStatesSS As Integer
    Private CoefficientMatrix(1, 1) As Double
    Private stateIndex(1, 1) As Integer
    Private RungeKuttaSolution() As Double
    Private N, I, H As Integer
    Private lambda_ED, lambda_H, mu_ED, mu_H As Double

    Public Function SetSystemSpecs(ByVal set_N As Integer, ByVal set_I
As Integer, ByVal set_H As Integer, ByVal set_lambda_ED As Double,
ByVal set_lambda_H As Double, ByVal set_mu_ED As Double, ByVal set_mu_H
As Double)

        N = set_N
        I = set_I
        H = set_H
        lambda_ED = set_lambda_ED
        lambda_H = set_lambda_H
        mu_ED = set_mu_ED
        mu_H = set_mu_H

        'calculate number of states
        NumberOfStates = (N + I + 1) * (H + 1)
        Dim idle_count As Integer
        For idle_count = 1 To I
            NumberOfStates = NumberOfStates + ((N + I) - idle_count +
1)
        Next idle_count

        NumberOfStates = NumberOfStates + 1
        ReDim CoefficientMatrix(NumberOfStates - 1, NumberOfStates - 1)
        ReDim RungeKuttaSolution(NumberOfStates - 1)
        ReDim stateIndex(NumberOfStates - 2, 3)

        'create stateIndex
        Dim n_count, i_count, h_count, state_count As Integer
        state_count = 0

        For h_count = 0 To H
            For n_count = 0 To N + I
                stateIndex(state_count, 0) = state_count
                stateIndex(state_count, 1) = n_count
                stateIndex(state_count, 2) = i_count
                stateIndex(state_count, 3) = h_count
                state_count = state_count + 1
            Next n_count
        Next h_count

        n_count = 0
        h_count = H
        For i_count = 1 To I
            Do While (n_count + i_count) <= N + I
                stateIndex(state_count, 0) = state_count
                stateIndex(state_count, 1) = n_count
                stateIndex(state_count, 2) = i_count
                stateIndex(state_count, 3) = h_count

                n_count = n_count + 1
                state_count = state_count + 1
            Loop
            n_count = 0
        Next i_count

    End Function

    Public Function numstates() As Integer
        Return NumberOfStates
    End Function
    Public Function min(ByVal first_value As Double, ByVal second_value
As Double)
        If first_value < second_value Then
            Return first_value
        Else
            Return second_value
        End If
    End Function
    Public Function Final_Generate()

        'Initialize the coefficientmatrix
        Dim eqn_counter, coef_counter As Integer

        coef_counter = 0

        For eqn_counter = 0 To NumberOfStates - 1
            For coef_counter = 0 To NumberOfStates - 1
                CoefficientMatrix(eqn_counter, coef_counter) = 0
            Next coef_counter
        Next eqn_counter


        'Fill the row for state (0,0,0)
        CoefficientMatrix(0, 0) = (lambda_ED + lambda_H) * (-1)
        CoefficientMatrix(0, getIndex(0, 0, 1)) = mu_H

        'Fill the rows for states (n,i,h) where,
        '0<n<N, i=0, 0<h<H

        Dim h_counter, n_counter, i_counter As Integer

        For h_counter = 1 To H - 1
            For n_counter = 1 To (N + I - 1)
                'this_state = (h_counter * (N + I + 1)) + n_counter
                CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter, 0, h_counter)) = (lambda_ED + lambda_H + min(I,
n_counter) * mu_ED + h_counter * mu_H) * (-1)
                CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter - 1, 0, h_counter)) = lambda_ED
                CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter, 0, h_counter - 1)) = lambda_H
                CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter, 0, h_counter + 1)) = (h_counter + 1) * mu_H
                CoefficientMatrix(getIndex(n_counter, 0, h_counter),
getIndex(n_counter + 1, 0, h_counter - 1)) = min(I, n_counter + 1) *
mu_ED
            Next n_counter

            'n=0 i=0 0<h<H
            CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(0, 0,
h_counter)) = (-1) * (lambda_ED + lambda_H + h_counter * mu_H)
            CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(0, 0,
h_counter - 1)) = lambda_H
            CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(0, 0,
h_counter + 1)) = (h_counter + 1) * mu_H
```

```
            CoefficientMatrix(getIndex(0, 0, h_counter), getIndex(1, 0,
h_counter - 1)) = mu_ED

            'n=N+I i=0 0<h<H
            CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N
+ I, 0, h_counter)) = (-1) * (I * mu_ED + lambda_H + h_counter * mu_H)
            CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N
+ I - 1, 0, h_counter)) = lambda_ED
            CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N
+ I, 0, h_counter - 1)) = lambda_H
            CoefficientMatrix(getIndex(N + I, 0, h_counter), getIndex(N
+ I, 0, h_counter + 1)) = (h_counter + 1) * mu_H



        Next h_counter

        '0<n<N+I

        For n_counter = 1 To N + I - 1
            '0<n<N+I, i=0, h=0
            CoefficientMatrix(getIndex(n_counter, 0, 0),
getIndex(n_counter, 0, 0)) = (-1) * (lambda_ED + lambda_H + min(I,
n_counter) * mu_ED)
            CoefficientMatrix(getIndex(n_counter, 0, 0),
getIndex(n_counter, 0, 1)) = mu_H
            CoefficientMatrix(getIndex(n_counter, 0, 0),
getIndex(n_counter - 1, 0, 0)) = lambda_ED

            '0<n<N+I, i=0, h=H
            CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter, 0, H)) = (-1) * (lambda_ED + min(I, n_counter) *
mu_ED + H * mu_H)
            CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter - 1, 0, H)) = lambda_ED
            CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter, 0, H - 1)) = lambda_H
            CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter, 1, H)) = H * mu_H
            CoefficientMatrix(getIndex(n_counter, 0, H),
getIndex(n_counter + 1, 0, H - 1)) = min(I, n_counter + 1) * mu_ED


        Next n_counter

        'n=N+I, i=0, h=H

        CoefficientMatrix(getIndex(N + I, 0, H), getIndex(N + I, 0, H))
= -(I * mu_ED + H * mu_H)
        CoefficientMatrix(getIndex(N + I, 0, H), getIndex(N + I - 1, 0,
H)) = lambda_ED
        CoefficientMatrix(getIndex(N + I, 0, H), getIndex(N + I, 0, H -
1)) = lambda_H
        '    CoefficientMatrix(getIndex(N + I, 0, H), getIndex(N + I,
1, H)) = H * mu_H

        'n=N+I, i=0, h=0

        CoefficientMatrix(getIndex(N + I, 0, 0), getIndex(N + I, 0, 0))
= -(lambda_H + I * mu_ED)
        CoefficientMatrix(getIndex(N + I, 0, 0), getIndex(N + I - 1, 0,
0)) = lambda_ED
        CoefficientMatrix(getIndex(N + I, 0, 0), getIndex(N + I, 0, 1))
= mu_H
```

```
        'n=0, i=0, h=H

        CoefficientMatrix(getIndex(0, 0, H), getIndex(0, 0, H)) = (-1)
* (lambda_ED + H * mu_H)
        CoefficientMatrix(getIndex(0, 0, H), getIndex(1, 0, H - 1)) =
mu_ED
        CoefficientMatrix(getIndex(0, 0, H), getIndex(0, 0, H - 1)) =
lambda_H
        CoefficientMatrix(getIndex(0, 0, H), getIndex(0, 1, H)) = H *
mu_H

        'n=N+I, 0<i<=I, h=H

        For i_counter = 1 To I - 1

            '0<n<N+I-i, 0<i<=I, h=H
            For n_counter = 1 To (N + I - i_counter)

                If i_counter > 0 And I <= n_counter + i_counter Then
                    '                CoefficientMatrix(NumberOfStates -
1, getIndex(n_counter, i_counter, H)) = _
                    '                CoefficientMatrix(NumberOfStates -
1, getIndex(n_counter, i_counter, H)) + _
                    '                -(lambda_ED + H * mu_H + min(I -
i_counter, n_counter) * mu_ED)

                    CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter - 1, i_counter, H)) = _
                    CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter - 1, i_counter, H)) + _
                    lambda_ED

                    CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter + 1, i_counter - 1, H)) = _
                    CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter + 1, i_counter - 1, H)) + _
                    min(I - (i_counter - 1), n_counter + 1) * mu_ED

                    If (n_counter + i_counter) < N + I Then
                        CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter, i_counter + 1, H)) = _
                        CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter, i_counter + 1, H)) + _
                        H * mu_H
                    End If

                Else
                    CoefficientMatrix(getIndex(n_counter, i_counter,
H), getIndex(n_counter, i_counter, H)) = -(lambda_ED + H * mu_H + min(I
- i_counter, n_counter) * mu_ED)
                    CoefficientMatrix(getIndex(n_counter, i_counter,
H), getIndex(n_counter - 1, i_counter, H)) = lambda_ED
                    CoefficientMatrix(getIndex(n_counter, i_counter,
H), getIndex(n_counter + 1, i_counter - 1, H)) = min(I - (i_counter -
1), n_counter + 1) * mu_ED
                    If (n_counter + i_counter) < N + I Then
                        CoefficientMatrix(getIndex(n_counter,
i_counter, H), getIndex(n_counter, i_counter + 1, H)) = H * mu_H
                    End If
                End If
            Next n_counter

            'n=N+I, 0<i<=I, h=H
            '        CoefficientMatrix(NumberOfStates - 1, getIndex(N +
I - i_counter, i_counter, H)) = _
            '        CoefficientMatrix(NumberOfStates - 1, getIndex(N +
I - i_counter, i_counter, H)) + _
            '        -((I - i_counter) * mu_ED + H * mu_H)

            CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) = _
            CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) + _
            lambda_ED

            'n=0, 0<i<=I, H

            CoefficientMatrix(getIndex(0, i_counter, H), getIndex(0,
i_counter, H)) = -1 * (lambda_ED + H * mu_H)
            CoefficientMatrix(getIndex(0, i_counter, H), getIndex(1,
i_counter - 1, H)) = mu_ED
            CoefficientMatrix(getIndex(0, i_counter, H), getIndex(0,
i_counter + 1, H)) = H * mu_H
        Next i_counter

        i_counter = I
        For n_counter = 1 To (N + I - I)
            '            CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter, i_counter, H)) = _
            '            CoefficientMatrix(NumberOfStates - 1,
getIndex(n_counter, i_counter, H)) + _
            '            -(lambda_ED + H * mu_H + min(I - i_counter, N)
* mu_ED)

            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter -
1, i_counter, H)) = _
            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter -
1, i_counter, H)) + _
            lambda_ED

            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter +
1, i_counter - 1, H)) = _
            CoefficientMatrix(NumberOfStates - 1, getIndex(n_counter +
1, i_counter - 1, H)) + _
            min(I - (i_counter - 1), n_counter + 1) * mu_ED
        Next n_counter

        'n=N+I, i=I, h=H

        '        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter, i_counter, H)) = _
        '        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter, i_counter, H)) + _
        '        -((I - i_counter) * mu_ED + H * mu_H)

        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) = _
        CoefficientMatrix(NumberOfStates - 1, getIndex(N + I -
i_counter - 1, i_counter, H)) + _
        lambda_ED

        'n=0, i=I, H

        '        CoefficientMatrix(NumberOfStates - 1, getIndex(0,
i_counter, H)) = _
        '        CoefficientMatrix(NumberOfStates - 1, getIndex(0,
i_counter, H)) + _
        '        -1 * (lambda_ED + H * mu_H)
        CoefficientMatrix(NumberOfStates - 1, getIndex(1, i_counter -
1, H)) = _
        CoefficientMatrix(NumberOfStates - 1, getIndex(1, i_counter -
1, H)) + _
        mu_ED


    End Function
    'Returns the index corresponding to the state with n,i and h
    Public Function getIndex(ByVal get_n As Integer, ByVal get_i As
Integer, ByVal get_h As Integer)
        Dim Found As Boolean
        Found = False

        Dim state_count As Integer
        state_count = 0


        Do While (state_count < NumberOfStates - 1) And Not (Found)
            If stateIndex(state_count, 2) = get_i Then
                Do While (state_count < NumberOfStates - 1) And Not
(Found)
                    If stateIndex(state_count, 3) = get_h Then
                        Do While (state_count < NumberOfStates - 1) And
Not (Found)
                            If stateIndex(state_count, 1) = get_n Then
                                Found = True
                            Else
                                state_count = state_count + 1
                            End If

                        Loop

                    Else
                        state_count = state_count + 1
                    End If
                Loop
            Else
                state_count = state_count + 1
            End If
        Loop

        If state_count = NumberOfStates - 1 And Not (Found) Then
            MsgBox("State n = " & get_n & " i = " & get_i & " h = " &
get_h & " is not found")
        End If


        Return state_count

    End Function
    Public Function MatlabSolveDE(ByVal t0 As Double, ByVal tf As
Double, ByRef initialvalues() As Double) As Double

        Dim Solver As New matlabSolution.matlabSolutionclassClass
        Dim Dummy As Object
        Dim solution As Object


        Call Solver.setCoefficients(1, Dummy, CoefficientMatrix)
        Call Solver.odesolver(1, solution, t0, tf, initialvalues)

        Return solution
        Call Solver.clearmatlab(1, Dummy)

    End Function
    Public Function EraseSystem()
        Erase CoefficientMatrix
        Erase stateIndex
    End Function
End Class
```

Vita

Toros Caglar was born in Turkey in 1982. In 1999, he attended Virginia Tech for his undergraduate studies in the department of Industrial and Systems Engineering. In 2003, after receiving his B.S. degree, he continued his studies at Virginia Tech towards a masters degree in the area of Operations Research in the same department. During these two years, he developed interest in applications of his education in the area of healthcare management. He is currently attending George Washington University for a doctoral degree in the department of Management Sciences where he hopes to continue working in the area of healthcare.