Restaurant Wait Time Estimation Final Report

ESE 499: Capstone Design Project

Erandy Vennet Segovia
Systems Science and Engineering Major
erandy.segovia@wustl.edu
(760)331-8779

Anuj Rudresh Patel
Systems Science and Engineering Major
anuj.patel@wustl.edu
(574) 485-6941

Zachary William Lonneman
Systems Science and Engineering Major
zlonneman@wustl.edu
(513)600-7086

Project Advisor
Dr. Benjamin Hartlage
Adjunct Professor at Washington University in St. Louis
rhartlage@gmail.com
(210)542-7621

Abstract

This study details the methodology developed to accurately estimate restaurant wait times. We considered several industry-standard restaurant management metrics in creating a simulation model which allowed us to generate estimated wait times. To confirm the accuracy of our results, we collected real-time scenario data and compared these data that we collected to the estimate provided by our model. The results of this comparison indicate that our estimation method is accurate under normal simulation parameters. Additionally, simulation input distributions and the calculation of individual components served as feedback mechanisms used to update the initial estimate.

Introduction and background

In restaurant management, customer satisfaction is a main driver of business success. Cornell University Professor, Sheryl E. Kimes Ph.D., states: "Those that continue with seating from a first-come, first-served waitlist should consider ways to empower guests, for example, by giving accurate wait times or issuing pagers." Giving customers an estimated wait time can be risky--liberal estimates can lure some away and conservative estimates can cause issues in customer service, and negatively affect the customer's probability of return. To address this issue, we implement a simulation model that estimates a customer's wait time more precisely.

The development of our methodology led us to first consider the analysis of dine-in establishments. First, we looked at point-of-service (POS), which is an effective tool for managing issues related to data-driven decisions. POS plays a very important role in the implementation of our simulation. This tool allows us to incorporate and keep track of various occurrences in the system and gives us a first-hand look at the different trends and patterns occurring in the restaurant. Keeping track of this information allows us to generate data regarding other components in our system, which allows us to delay generating estimates until we had established a high degree of certainty in the accuracy of the estimate.

To check the results of our simulation, we looked up information regarding M/M/s queues. An M/M/s queue is a single queue served by multiple servers. For each server, service rate is represented with μ and arrival rate can be found using, λ /m where λ is the arrival rate and is measured in customers per hour. Such

systems generally begin with customers at the head of a queue and moves each customer to service when a server becomes available. From *Traffic Behavior and Queuing in a QoS Environment*, we learned that occupancy in a dine-in system can be measured using $N = \rho/(1 - \rho)$ and delay as $T = 1/(m - \lambda)$, where $\rho = \lambda/m$ (Bertsekas). These equations proved useful in allowing us to compare results from our simulation, and in the overall implementation of our estimation method.

Problem statement and objectives

Most restaurants approximate wait times using information such as occupancy and the number of available servers. When asking Tom Schmidt, owner of local St. Louis restaurant, Salt & Smoke, how wait times were estimated in his establishment, he assured us that they weren't very accurate and that hosts try to guess based on intuition. Here, our objective is to generate a method that provides a more accurate wait time estimation by accounting for forgotten or ignored variables. These variables include information specific to every table in the restaurant such as the amount and type of food ordered, and the reason for visiting (group celebration, family event, date or quick meal). Because these tasks cannot be quickly done by humans, we implement an estimation method that automates these calculations, presenting restaurants with the option to maximize wait time estimate accuracy.

Our first objective, creating a simulation of our restaurant, required an environment which considers random variables. First, we test an environment with a given number of tables, and servers, and semi-random influx of customers. The

environment consisted of a varied menu with different ranges in preparation times, and a semi-random order rate. Next, we approximated variables such as the amount of time that customers stay after they have finished their meals. From this, we observed patterns which allowed us to pick out instances that needed to be treated with specific, more unique approaches.

Methodology Development

Data Collection

After creating our simulated system we gathered information from Salt & Smoke in order to accurately replicate a realistic environment. For this project, we chose to focus on collecting data from sit-down restaurants because issues with imprecise wait-time estimations are experienced most in this setting. While at the restaurant, we focused on measuring observations such as arrival and departure times, the overall time that it takes to get a customer through the queue, time duration of individual tasks (sitting, ordering, paying, etc.), party sizes, and estimated versus actual wait times. Additionally, we observed the number of tables available, the number of servers available during wait time periods, the number of tables allotted to each server, the preparation times for each meal, and the difference in average dining time of groups based on occasion.

After accruing this data, we returned to our simulation, made comparisons, and then made necessary adjustments to our system. Taking the different variables that we gathered into consideration, we were able to compose a method to find the most accurate wait times. To better understand the current system of wait time

estimation, Schmidt explained the arrangement of tables, maximum seating, food preparation times and server allocation. When collecting data, we took into account the overall layout of the restaurant, the number of servers, and collected several other factors regarding the overall functionality of the business. The collection of this information allowed us to gain a better understanding of some of the variables used when implementing our model and even led us to notice the significance of several variables that we had not previously accounted for. From the data we collected, we calculated several parameters, which allowed us to have something to compare the results of our simulation.

Simulation

While working on our simulation model, we wanted to ensure we had varied data and accurately depicted a real life system. In order to achieve an accurate simulation, we wanted to use the data we collected from a real restaurant so that we would have an idea of how the restaurant flowed, or functioned. To begin the simulation process, we began using a program called Arena, a simulation software that Dr. Hartlage recommended. Our original idea for the simulation made us believe that this software would be very useful because Arena is able to create random "parties" which followed uniform normal distribution. This software automatically collects data such as queue times and also keeps track of party entries and exits. As we began to create a larger system with more variables, the trial version that we were using reached its memory capacity and we were unable to continue implementing this model. After considering several alternatives, we

created a new simulation model in Matlab which allowed us to simulate a night at the restaurant.

To implement our simulated model, we created two methods, one which initialized the "night" and another that would run through a step-by-step process of how the night would play out. Some of the steps followed in this process consisted of creating an arrival time for a party of a random size (ranging from 1 to 12, with a mean of 2), an anticipated order for the party, and the total time that the party spent in the queue. The initialization method was created to simulate a night at Salt & Smoke but adjustments in its inputs could be made to fit those of different restaurants.

The simulated data we created using these two methods were saved into an array which would be fed into the system, with each column arriving at their designated times. Another unique portion that we accurately recreated the decision process used by the host staff to seating incoming restaurant patrons who came in larger parties or required restructuring of tables to accommodate their parties. To do this, we decided that a two person party would be seated at a four person table if there were no more available two person tables. Here, we recognized that the host had an advantage over the simulation, but we figured that our algorithm was able to call on previous data. In the end, we had to make assumptions on how likely a 12 person party could be seated at the time they arrived. If there was one 6 person table, 2 available 4 person tables, and 2 available 2 person tables available, we would not seat them because it is more likely that the tables that are available would

not be around each other. Empirical experimentation eventually produced a decision process with outcomes that mimicked the human decision process we observed with sufficient accuracy to continue. As the tables filled up, we were no longer able to seat incoming parties, so we needed to create a new array to hold the data of the waiting parties. The last obstacle was deciding when to seat new arrivals versus when to seat those on the wait list and sorting between the waitlist. According to the hosts at Salt & Smoke, they try to adhere to a first come first seat rule, meaning that when possible they try to stick to the waitlist in the order that people arrived. This doesn't apply, however when a party of two arrives after a party of 10 who is waiting on tables. In real life the host can look around the restaurant and see which tables are almost finished, and set aside those tables for the larger party. In our simulation there was no way of telling which tables were surrounded by which tables. Meaning once again we would need to make some assumptions on how to seat these larger parties, without tables constantly being filled by smaller parties.

Our best solution ended up fitting the mold of a restaurant fairly well--it was to put a hold on tables, so if a restaurant was completely booked and a 10 person party arrived, we would put a hold on the first available 6 person table, and the fourth available 4 person table. The simplifying assumptions we made allowed us to capture the essential features of the restaurant queuing process without requiring unnecessary model fidelity.

Estimation Model Implementation

In order to accurately calculate an incoming party's wait time, we need data regarding the seated parties and those ahead of the incoming party in the waitlist at a restaurant. That information then must be used to estimate when a table accommodating the incoming party's size is available. To do this in a memory and time efficient manner, we chose to create five data structures that store and track information regarding the menu, the servers, the incoming parties, the tables, and the wait list throughout the simulation.

- Menu: this constant data structure contains data about the menu items served at the restaurant. For each menu item, three pieces of information are stored – a unique, non-zero numerical code that identifies each menu item, the average amount of time it takes for the item to be prepared by the cooks, and the standard deviation of this previous statistic.
- Servers: this data structure stores three pieces of data regarding the servers in the restaurant. First, there is a unique, non-zero numerical identifier called the server code. Then, the total number of parties the server has attended and the average time the parties stay at the restaurant are stored.
- Parties: this data structure stores data about the parties that come into the restaurant. The information stored includes the party code (unique, non-zero numerical identifier), the party size, their food order (slowest cooking time item is chosen), when the party entered the restaurant, and, if the party has left the restaurant, how long they stayed at the restaurant.

- Tables: this data structure stores data about the tables in the restaurant and their occupancy. The structure, using a unique, non-zero numerical code to identify each table, tracks their sizes and whether they are occupied. If the table is occupied, the party's code, the time the party was seated, and the server code.
- Waitlist: this data structure contains a list of any parties in the waitlist in order of arrival. The party code and the party's calculated wait time is stored in this structure.

In order to ensure that these structures contain accurate and up-to-date instructions, we created functions that register important events in the restaurant's operations and the customer's experience. These events are:

- The restaurant is inaugurated (initialization)
- A party enters into the restaurant
- Restaurant host checks for available seats for incoming party
- If seats are not available, party is added to the waitlist and a wait time is reported
- If seats are available, party is seated and their order taken
- The party's food order is served
- The party leaves the restaurant

While these functions input and output data commonly seen and used in POS machines in most restaurants, the one output unique to our system is the computation of a wait time. To explain how this value is estimated, we envision a

scenario in which a family of 4 comes into a restaurant. Assuming that the restaurant is packed but no wait list exists, our method will find the table of 4 that is closest to finishing their meal and leaving the restaurant. This is done through the use of the dishes' cook times and the statistical data on how long customers stay at the restaurant after the arrival of their food. For our estimation method, we initially do so using data we collected from Salt & Smoke. However, part way through the simulation, we can choose to shift to calculating the wait time using past data stored by our system. The family of 4's wait time will hence be equal to the time in which the next table of 4 becomes available.

In a more advanced scenario, say the family of 4 enters a packed restaurant already with a wait list. Our method matches parties in the waitlist with occupied tables, where the parties at the front of the waitlist are matched to tables more likely to be unoccupied first. This matching isn't a locked arrangement, it is just used as a means to generate estimates; parties are still seated in order of their arrival. Therefore, for the sake of the computation only, the waitlist is divided up into separate queues, one for each table in the restaurant. However, because we do not have information on the food order of parties in the wait list, wait time estimates for parties past the first "layer" of the queue will have to rely on the average time a party spends at the restaurant. In order to ensure accurate wait time estimates, these values are constantly updated as customers leave the restaurant.

In order to merge our simulation and estimation model, we developed a simulation model that used the events randomly generated by our simulation and

ran the functions representing the appropriate simulation event in chronological order. At the end of one simulated evening, data regarding the accuracy of our wait time estimates is reported and the cooking time and eating time data is fed back into the next simulation to improve the accuracy of the next run's estimates. This process is shown in the flowchart in Appendix E.

Results

Based on the real-time sample data that we collected from Salt & Smoke, we found that once the restaurant filled up, a group of 3 was given an estimated wait time of 10-15 minutes. The actual wait time in this scenario turned out to be 4 minutes. Comparing this information to the results gathered from our simulation and the implemented method, we established the success of our experiment.

With our simulated data, our goal was to analyze the difference between the real wait times and estimated wait times of any party that was placed in the waitlist for any period of time. For a given simulated evening, approximately 80-90 parties would enter the restaurant. Out of these parties, about 10-12 parties are waitlisted at some point in their experience. Calculating and plotting the difference between these parties' real and estimated wait times shows us that, under normal dining parameters, our estimates are accurate. Appendix F shows a plot of these calculated differences from one particular simulation. In this plot, we see that the differences don't exceed approximately ±2 mins.

Discussion

Looking back into the design of our simulation and estimation method, one issue we faced was balancing the use of random numbers to create variability while ensuring accurate wait time estimates. In our initial analysis of restaurants and how they are structured, we knew that the time of completion for every process in a restaurant wasn't constant, and fell into some distribution. While we were able to ascertain that party arrivals were distributed exponentially, we didn't have the means to collect sufficient data to see what distributions food cooking time and party eating time fell under.

When it came time to create our simulation, this was a problem since we didn't know what random number generator to use. As a result, we settled on using an exponential random number generator for all variability. This decision may have added significant inaccuracy to our results. If given the chance to implement this system, further in-depth research will allow us to pinpoint the probability distributions. Additionally, our simulation only covered normal dining parameters and wasn't put through a more rigorous test of high flow scenarios. Although we decided against testing this because very few restaurants experience such situations and because of time constraints, there is a possibility that our estimation method will not be accurate in those scenarios.

When designing the estimation method, ensuring that restaurants and restaurant employees wouldn't need to make any extra effort to contribute to the success of the system was paramount. We wanted to make sure our product would integrate with existing data collection systems at restaurants. Because most modern

establishments use POS systems to manage and collect data, we designed our method to complement this. Hence, it made sense to create a real-time estimator using the seating and order information servers already input in the POS.

From a marketing point of view, this makes our product very strong and valuable. Being a moderately simple method that extends an existing system, restaurants won't need to invest large sums to obtain our product. Additionally, since servers and hosts won't have to do anything they didn't do before to ensure the success of this system, there will be no retraining or operational costs. Restaurants will also be able to reap benefits of increased customer satisfaction and additional restaurant operation data. Though these benefits aren't accurately quantifiable, they will definitely contribute to the success of an establishment.

Conclusions

As we look back on our results, and on the simulation model itself, given the right equipment, and willing restaurants, our design could become a fairly simple addition to the growing technical side of running a restaurant. The key distinction of our system being the collection and dispersion of the given inputs from hosts and waiters. The longer the system is in place the more accurate the system becomes, to the point where no longer will the host need to be a job of finding the best location for the guest. While our system runs the data on when and where the guest is likely to be seated, the host is freed up to interact with guests. While many restaurants are starting to implement a computerized system for the seating and ordering of guest, most still do this manually. It's interesting looking at this system, and knowing that

most of which is very simple, that this has yet to be put into action. While this started off as our senior design project, and will end as our senior design project, we have very little doubt that in the near future a similar program will be common in restaurants.

Bibliography

Bertsekas, Dimitri P. "Traffic Behavior and Queuing in a QoS Environment." MIT. Web. 12 Feb. 2015. <http://web.mit.edu/dimitrib/www/OPNET_Full_Presentation.ppt>.

Kimes, Sheryl E., Ph.D., and Jochen Wirtz, Ph.D. "Customer Satisfaction with Seating Policies in Casual-Dining Restaurants." Cornell Hospitality Report 7.16 (2007): n. pag. Cornell University School of Hotel Administration. Web. 1 Apr. 2015.

Appendix A: arrivalGenerator.m

```
clear
clc
stop = 0;
i = 1;
x=6.0;
arrivals = [];
while x < 9.0;
    if x<6.5;
        arrivals(i) = x+rand()/15;
        x=arrivals(i);
    end;
    if x>=6.5&&x<7.0;
        arrivals(i)=x+rand()/25;
        x=arrivals(i);
    end;
    if x>=7.0&&x<7.5;
        arrivals(i)=x+rand()/20;
        x=arrivals(i);
    end;
    if x>=7.5&&x<8.0;
        arrivals(i)=x+rand()/15;
        x=arrivals(i);
    end;
    if x>=8.0&&x<9.0;
        arrivals(i)=x+rand()/10;
        x=arrivals(i);
    end;
    stop=i;
    i=i+1;
end;
i=i-1;
while i>0;
    if arrivals(1,i)<7.0;
    y = arrivals(1,i)-6.0;
    arrivals(7,i)=y*.6+6.0;
        if arrivals(7,i)>=6.595
        arrivals(7,i)= 7.00;
        end;
    elseif arrivals(1,i)<8.0;
    y = arrivals(1,i)-7.0;
    arrivals(7,i)=y*.6+7.0;
        if arrivals(7,i)>=7.595
        arrivals(7,i)= 8.00;
        end;
    else
    y = arrivals(1,i)-8.0;
    arrivals(7,i)=y*.6+8.0;
        if arrivals(7,i)>=8.595
        arrivals(7,i)= 9.00;
        end;
    end;
```

```
    arrivals(7,i)= round(arrivals(7,i)*100)/100;
    i=i-1;
end;

i=i+1;
while i<stop+1;
    order = rand();
    random = rand();
    oneThird = rand();
    dev = randn();
    if dev<0;
        dev=dev/2;
    end;
        x=.333;
    if random<.5;
        arrivals(2,i)=2;
        arrivals(5,i)=(x*15.*dev+20);
    elseif random<.75;
        arrivals(2,i)=round(rand()+3);
        arrivals(5,i)=(x*15.*dev+15);
    elseif random<.9;
        arrivals(2,i)=round(rand()+5);
        arrivals(5,i)=(x*15.*dev+25);
    elseif random<.95;
        if oneThird<.333;
            arrivals(2,i)= 7;
        elseif oneThird<.666;
            arrivals(2,i)=8;
        else
            arrivals(2,i)=9;
        end;
        arrivals(5,i)=(x*20.*dev+45);
    elseif random<.98;
        arrivals(2,i)=1;
        arrivals(5,i)=(x*15.*dev+20);
    else
        if oneThird<.333;
            arrivals(2,i)= 10;
        elseif oneThird<.666;
            arrivals(2,i)=11;
        else
            arrivals(2,i)=12;
        end;
        arrivals(5,i)=(x*20.*dev+50);
    end;
    if order<.35;
        arrivals(3,i)=1;%Deepdish Pizza
        arrivals(4,i)=(x*5.*dev+35);
    elseif order<.55;
        arrivals(3,i)=2;%Traditional Pizza
        arrivals(4,i)=(x*10.*dev+20);
    elseif order<.75;
        arrivals(3,i)=3;%Pasta
        arrivals(4,i)=(x*5.*dev+20);
```

```
    elseif order<.85;
        arrivals(3,i)=4;%Steak
        arrivals(4,i)=(x*5.*dev+15);
    elseif order<.95;
        arrivals(3,i)=5;%Chicken
        arrivals(4,i)=(x*5.*dev+15);
    else;
        arrivals(3,i)=6;%Salad
        arrivals(4,i)=(x*5.*dev+10);
    end
    arrivals(6,i)=round(arrivals(4,i)+arrivals(5,i));

    i=i+1;

end;
%Create simulation that takes away tables and counts time.
save('arrivals.mat', 'arrivals');
```

Appendix B: openTable.m

```
function [i,twotable,fourtable,sixtable,wait,queue,tables] =
opentable(i,twotable,fourtable,sixtable,wait,queue,tables)
load('arrivals.mat');
if arrivals(2,i)<=2;
    if twotable>0&&twotable<=12
        tables(1,1)=arrivals(6,i);
        twotable=twotable-1;
    elseif fourtable>0&&twotable<=12;
        tables(2,1)=arrivals(6,i);
        fourtable=fourtable-1;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=4;
    if f function [i,twotable,fourtable,sixtable,wait,queue,tables] =
opentable(i,twotable,fourtable,sixtable,wait,queue,tables)
load('arrivals.mat');
if arrivals(2,i)<=2;
    if twotable>0&&twotable<=12
        tables(1,1)=arrivals(6,i);
        twotable=twotable-1;
    elseif fourtable>0&&twotable<=12;
        tables(2,1)=arrivals(6,i);
        fourtable=fourtable-1;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=4;
    if fourtable>0 && fourtable<=12
        tables(2,1)=arrivals(6,i);
        fourtable=fourtable-1;
```

```
    elseif twotable>4 && twotable<=12
        tables(1,1)=arrivals(6,i);
        tables(1,2)=arrivals(6,i);
        twotable=twotable-2;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;

    end;
elseif arrivals(2,i)<=6;
    if sixtable>0 && sixtable<=2
        tables(3,1)=arrivals(6,i);
        sixtable=sixtable-1;
    elseif fourtable>5 && fourtable<=12;
        if twotable>5 && twotable<12;
            tables(2,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            fourtable=fourtable-1;
            twotable=twotable-1;
        else
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
            fourtable=fourtable-2;
        end;
    elseif twotable>5 && twotable<=12;
        tables(1,1)=arrivals(6,i);
        tables(1,2)=arrivals(6,i);
        tables(1,3)=arrivals(6,i);
        twotable=twotable-3;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=8;
    if fourtable>=6 && fourtable<=12;
        tables(2,1)=arrivals(6,i);
        tables(2,2)=arrivals(6,i);
        fourtable=fourtable-2;
    elseif sixtable>0 && sixtable<=2;
        if twotable>=5 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            twotable=twotable-1;
            sixtable=sixtable-1;
        elseif fourtable>=5 && fourtable<=12;
            tables(3,1)=arrivals(6,i);
            tables(2,1)=arrivals(6,i);
            fourtable=fourtable-1;
            sixtable=sixtable-1;
        else
            tables(3,1)=arrivals(6,i);
            tables(3,2)=arrivals(6,i);
            sixtable=sixtable-2;
        end;
```

```
    elseif fourtable>=5 && fourtable<=12;
        if twotable>=5 && twotable<=12;
            tables(2,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            fourtable=fourtable-1;
            twotable=twotable-1;
        end;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=10;
    if sixtable>0;
        if fourtable>4 && fourtable<=12;
            tables(3,1)=arrivals(6,i);
            tables(2,1)=arrivals(6,i);
            fourtable=fourtable-1;
            sixtable=sixtable-1;
        elseif twotable>5 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
            twotable=twotable-2;
            sixtable=sixtable-1;
        else
            tables(3,1)=arrivals(6,i);
            tables(3,2)=arrivals(6,i);
            sixtable=sixtable-2;
        end;
    elseif fourtable>4 && fourtable<=12;
        if fourtable>5 && twotable>4 && fourtable<=12 && twotable<=12;
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            fourtable=fourtable-2;
            twotable=twotable-1;
        elseif twotable>6 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
            tables(1,3)=arrivals(6,i);
            twotable=twotable-3;
            sixtable=sixtable-1;
        end;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=12;
    if sixtable>0;
        if sixtable>1;
            tables(3,1)=arrivals(6,i);
            tables(3,2)=arrivals(6,i);
            sixtable=sixtable-2;
```

```
        elseif fourtable>4 && fourtable<=12;
            if twotable>4 && twotable<=11;
                tables(3,1)=arrivals(6,i);
                tables(2,1)=arrivals(6,i);
                tables(1,1)=arrivals(6,i);
                twotable=twotable-1;
                fourtable=fourtable-1;
                sixtable=sixtable-1;
            elseif fourtable>5 && fourtable<=12;
                tables(3,1)=arrivals(6,i);
                tables(2,1)=arrivals(6,i);
                tables(2,2)=arrivals(6,i);
                fourtable=fourtable-2;
                sixtable=sixtable-1;
            end
        elseif twotable>7 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
            tables(1,3)=arrivals(6,i);
            twotable=twotable-3;
            sixtable=sixtable-1;
        end;
    elseif fourtable>5 && fourtable<=12;
        if fourtable>6 && fourtable<=12;
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
            tables(2,3)=arrivals(6,i);
            fourtable=fourtable-3;
        elseif twotable>5 && twotable<=12;
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
            fourtable=fourtable-2;
            twotable=twotable-2;
        end;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
end;

i=i+1;
count=1;
if i<=(length(arrivals));
    while count<=12
        if i<=(length(arrivals)); %This is always true per while loop
criteria.
            if i==(length(arrivals))
                time=0;
            else
                time = round((arrivals(1,i+1)-arrivals(1,i))*60); %Change
to minutes before subtracting.
```

```
            end
            if tables(1,count)>0;
                tables(1,count)=tables(1,count)-time;
                if tables(1,count)<=0;
                    tables(1,count)=0;
                    twotable=twotable+1;
                end;
            end;
            if tables(2,count)>0;
                tables(2,count)=tables(2,count)-time;
                if tables(2,count)<=0;
                    tables(2,count)=0;
                    fourtable=fourtable+1;
                end;
            end;
            if tables(3,count)>0;
                tables(3,count)=tables(3,count)-time;
                if tables(3,count)<=0;
                    tables(3,count)=0;
                    sixtable=sixtable+1;
                end;
            end;
            count = count+1;
        end;
        if(i==88 && count==12)
            print=5;
        end
    end
end;
ourtable>0 && fourtable<=12
        tables(2,1)=arrivals(6,i);
        fourtable=fourtable-1;
    elseif twotable>4 && twotable<=12
        tables(1,1)=arrivals(6,i);
        tables(1,2)=arrivals(6,i);
        twotable=twotable-2;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;

    end;
elseif arrivals(2,i)<=6;
    if sixtable>0 && sixtable<=2
        tables(3,1)=arrivals(6,i);
        sixtable=sixtable-1;
    elseif fourtable>5 && fourtable<=12;
        if twotable>5 && twotable<12;
            tables(2,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            fourtable=fourtable-1;
            twotable=twotable-1;
        else
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
```

```
            fourtable=fourtable-2;
        end;
    elseif twotable>5 && twotable<=12;
        tables(1,1)=arrivals(6,i);
        tables(1,2)=arrivals(6,i);
        tables(1,3)=arrivals(6,i);
        twotable=twotable-3;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=8;
    if fourtable>=6 && fourtable<=12;
        tables(2,1)=arrivals(6,i);
        tables(2,2)=arrivals(6,i);
        fourtable=fourtable-2;
    elseif sixtable>0 && sixtable<=2;
        if twotable>=5 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            twotable=twotable-1;
            sixtable=sixtable-1;
        elseif fourtable>=5 && fourtable<=12;
            tables(3,1)=arrivals(6,i);
            tables(2,1)=arrivals(6,i);
            fourtable=fourtable-1;
            sixtable=sixtable-1;
        else
            tables(3,1)=arrivals(6,i);
            tables(3,2)=arrivals(6,i);
            sixtable=sixtable-2;
        end;
    elseif fourtable>=5 && fourtable<=12;
        if twotable>=5 && twotable<=12;
            tables(2,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            fourtable=fourtable-1;
            twotable=twotable-1;
        end;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=10;
    if sixtable>0;
        if fourtable>4 && fourtable<=12;
            tables(3,1)=arrivals(6,i);
            tables(2,1)=arrivals(6,i);
            fourtable=fourtable-1;
            sixtable=sixtable-1;
        elseif twotable>5 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
```

```
            twotable=twotable-2;
            sixtable=sixtable-1;
        else
            tables(3,1)=arrivals(6,i);
            tables(3,2)=arrivals(6,i);
            sixtable=sixtable-2;
        end;
    elseif fourtable>4 && fourtable<=12;
        if fourtable>5 && twotable>4 && fourtable<=12 && twotable<=12;
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            fourtable=fourtable-2;
            twotable=twotable-1;
        elseif twotable>6 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
            tables(1,3)=arrivals(6,i);
            twotable=twotable-3;
            sixtable=sixtable-1;
        end;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
elseif arrivals(2,i)<=12;
    if sixtable>0;
        if sixtable>1;
            tables(3,1)=arrivals(6,i);
            tables(3,2)=arrivals(6,i);
            sixtable=sixtable-2;
        elseif fourtable>4 && fourtable<=12;
            if twotable>4 && twotable<=11;
                tables(3,1)=arrivals(6,i);
                tables(2,1)=arrivals(6,i);
                tables(1,1)=arrivals(6,i);
                twotable=twotable-1;
                fourtable=fourtable-1;
                sixtable=sixtable-1;
            elseif fourtable>5 && fourtable<=12;
                tables(3,1)=arrivals(6,i);
                tables(2,1)=arrivals(6,i);
                tables(2,2)=arrivals(6,i);
                fourtable=fourtable-2;
                sixtable=sixtable-1;
            end
        elseif twotable>7 && twotable<=12;
            tables(3,1)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
            tables(1,3)=arrivals(6,i);
            twotable=twotable-3;
            sixtable=sixtable-1;
```

```
        end;
    elseif fourtable>5 && fourtable<=12;
        if fourtable>6 && fourtable<=12;
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
            tables(2,3)=arrivals(6,i);
            fourtable=fourtable-3;
        elseif twotable>5 && twotable<=12;
            tables(2,1)=arrivals(6,i);
            tables(2,2)=arrivals(6,i);
            tables(1,1)=arrivals(6,i);
            tables(1,2)=arrivals(6,i);
            fourtable=fourtable-2;
            twotable=twotable-2;
        end;
    else
        wait(:,queue)=arrivals(:,i);
        queue=queue+1;
    end;
end;

i=i+1;
count=1;
if i<=(length(arrivals));
    while count<=12
        if i<=(length(arrivals)); %This is always true per while loop
criteria.
            if i==(length(arrivals))
                time=0;
            else
                time = round((arrivals(1,i+1)-arrivals(1,i))*60); %Change
to minutes before subtracting.
            end
            if tables(1,count)>0;
                tables(1,count)=tables(1,count)-time;
                if tables(1,count)<=0;
                    tables(1,count)=0;
                    twotable=twotable+1;
                end;
            end;
            if tables(2,count)>0;
                tables(2,count)=tables(2,count)-time;
                if tables(2,count)<=0;
                    tables(2,count)=0;
                    fourtable=fourtable+1;
                end;
            end;
            if tables(3,count)>0;
                tables(3,count)=tables(3,count)-time;
                if tables(3,count)<=0;
                    tables(3,count)=0;
                    sixtable=sixtable+1;
                end;
            end;
```

```
            count = count+1;
        end;
        if(i==88 && count==12)
            print=5;
        end
    end
end;
```

Appendix C: tableCounter.m

```
clear
clc
load('arrivals.mat');
tables=zeros(3,12);
twotable=12;
fourtable=12;
sixtable=2;
wait=[];
queue=1;
call=1;
i=1;
while i<((length(arrivals)+1)) || isempty(wait)==0;
    if isempty(wait)==0;
        if wait(2,1)==0;
            wait(:,1)=[];
            circshift(wait, [0,-1])
            continue
        end;
        print = wait(2,1)
        if 0<wait(2,1)&& wait(2,1)<=2;
            if twotable>0&&twotable<=12
                tables(1,1)=wait(6,1);
                twotable=twotable-1;
                wait(:,1)=[];
                circshift(wait, [0,-1])
            elseif fourtable>0&&twotable<=12;
                tables(2,1)=wait(6,1);
                fourtable=fourtable-1;
                wait(:,1)=[];
                circshift(wait, [0,-1])
           end;
        elseif 2<wait(2,1)&& wait(2,1)<=4;
            if fourtable>0 && fourtable<=12
                tables(2,1)=wait(6,1);
                fourtable=fourtable-1;
                wait(:,1)=[];
                circshift(wait, [0,-1])
            elseif twotable>4 && twotable<=12
                tables(1,1)=wait(6,1);
                tables(1,2)=wait(6,1);
                twotable=twotable-2;
                wait(:,1)=[];
                circshift(wait, [0,-1])
```

```
    end;
elseif 4<wait(2,1)&& wait(2,1)<=6;
    if sixtable>0 && sixtable<=2
        tables(3,1)=wait(6,1);
        sixtable=sixtable-1;
        wait(:,1)=[];
        circshift(wait, [0,-1])
    elseif fourtable>5 && fourtable<=12;
        if twotable>5 && twotable<12;
            tables(2,1)=wait(6,1);
            tables(1,1)=wait(6,1);
            fourtable=fourtable-1;
            twotable=twotable-1;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        else
            tables(2,1)=wait(6,1);
            tables(2,2)=wait(6,1);
            fourtable=fourtable-2;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        end;
    elseif twotable>5 && twotable<=12;
        tables(1,1)=wait(6,1);
        tables(1,2)=wait(6,1);
        tables(1,3)=wait(6,1);
        twotable=twotable-3;
        wait(:,1)=[];
        circshift(wait, [0,-1])
    end;
elseif 6<wait(2,1)&& wait(2,1)<=8;
    if fourtable>=6 && fourtable<=12;
        tables(2,1)=wait(6,1);
        tables(2,2)=wait(6,1);
        fourtable=fourtable-2;
        wait(:,1)=[];
        circshift(wait, [0,-1])
    elseif sixtable>0 && sixtable<=2;
        if twotable>=5 && twotable<=12;
            tables(3,1)=wait(6,1);
            tables(1,1)=wait(6,1);
            twotable=twotable-1;
            sixtable=sixtable-1;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        elseif fourtable>=5 && fourtable<=12;
            tables(3,1)=wait(6,1);
            tables(2,1)=wait(6,1);
            fourtable=fourtable-1;
            sixtable=sixtable-1;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        else
            tables(3,1)=wait(6,1);
```

```
                    tables(3,2)=wait(6,1);
                    sixtable=sixtable-2;
                    wait(:,1)=[];
                    circshift(wait, [0,-1])
                end;
            elseif fourtable>=5 && fourtable<=12;
                if twotable>=5 && twotable<=12;
                    tables(2,1)=wait(6,1);
                    tables(1,1)=wait(6,1);
                    fourtable=fourtable-1;
                    twotable=twotable-1;
                    wait(:,1)=[];
                    circshift(wait, [0,-1])
                end;
            end;
        elseif 8<wait(2,1)&& wait(2,1)<=10;
            if sixtable>0;
                if fourtable>4 && fourtable<=12;
                    tables(3,1)=wait(6,1);
                    tables(2,1)=wait(6,1);
                    fourtable=fourtable-1;
                    sixtable=sixtable-1;
                    wait(:,1)=[];
                    circshift(wait, [0,-1])
                elseif twotable>5 && twotable<=12;
                    tables(3,1)=wait(6,1);
                    tables(1,1)=wait(6,1);
                    tables(1,2)=wait(6,1);
                    twotable=twotable-2;
                    sixtable=sixtable-1;
                    wait(:,1)=[];
                    circshift(wait, [0,-1])
                else
                    tables(3,1)=wait(6,1);
                    tables(3,2)=wait(6,1);
                    sixtable=sixtable-2;
                    wait(:,1)=[];
                    circshift(wait, [0,-1])
                end;
            elseif fourtable>4 && fourtable<=12;
                if fourtable>5 && twotable>4 && fourtable<=12 &&
twotable<=12;
                    tables(2,1)=wait(6,1);
                    tables(2,2)=wait(6,1);
                    tables(1,1)=wait(6,1);
                    fourtable=fourtable-2;
                    twotable=twotable-1;
                    wait(:,1)=[];
                    circshift(wait, [0,-1])
                elseif twotable>6 && twotable<=12;
                    tables(3,1)=wait(6,1);
                    tables(1,1)=wait(6,1);
                    tables(1,2)=wait(6,1);
                    tables(1,3)=wait(6,1);
```

```
            twotable=twotable-3;
            sixtable=sixtable-1;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        end;
    end;
elseif 10<wait(2,1)&& wait(2,1)<=12;
    if sixtable>0;
        if sixtable>1;
            tables(3,1)=wait(6,1);
            tables(3,2)=wait(6,1);
            sixtable=sixtable-2;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        elseif fourtable>4 && fourtable<=12;
            if twotable>4 && twotable<=11;
                tables(3,1)=wait(6,1);
                tables(2,1)=wait(6,1);
                tables(1,1)=wait(6,1);
                twotable=twotable-1;
                fourtable=fourtable-1;
                sixtable=sixtable-1;
                wait(:,1)=[];
                circshift(wait, [0,-1])
            elseif fourtable>5 && fourtable<=12;
                tables(3,1)=wait(6,1);
                tables(2,1)=wait(6,1);
                tables(2,2)=wait(6,1);
                fourtable=fourtable-2;
                sixtable=sixtable-1;
                wait(:,1)=[];
                circshift(wait, [0,-1])
            end
        elseif twotable>7 && twotable<=12;
            tables(3,1)=wait(6,1);
            tables(1,1)=wait(6,1);
            tables(1,2)=wait(6,1);
            tables(1,3)=wait(6,1);
            twotable=twotable-3;
            sixtable=sixtable-1;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        end;
    elseif fourtable>5 && fourtable<=12;
        if fourtable>6 && fourtable<=12;
            tables(2,1)=wait(6,1);
            tables(2,2)=wait(6,1);
            tables(2,3)=wait(6,1);
            fourtable=fourtable-3;
            wait(:,1)=[];
            circshift(wait, [0,-1])
        elseif twotable>5 && twotable<=12;
            tables(2,1)=wait(6,1);
            tables(2,2)=wait(6,1);
```

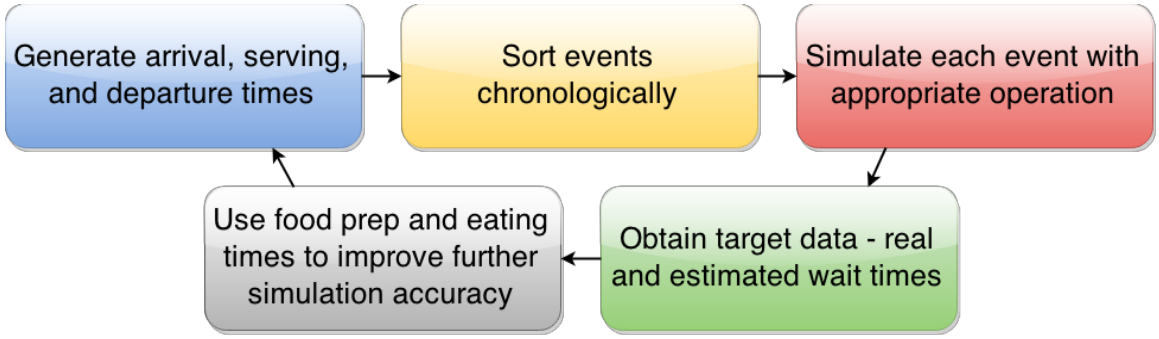
```
                    tables(1,1)=wait(6,1);
                    tables(1,2)=wait(6,1);
                    fourtable=fourtable-2;
                    twotable=twotable-2;
                    wait(:,1)=[];
                    circshift(wait, [0,-1])
                end;
            end;
        end;
       count = 1;
        while count<=12
            if tables(1,count)>0;
                tables(1,count)=tables(1,count)-1;
                if tables(1,count)<=0;
                    tables(1,count)=0;
                    twotable=twotable+1;
                end;
            end;
            if tables(2,count)>0;
                tables(2,count)=tables(2,count)-1;
                if tables(2,count)<=0;
                    tables(2,count)=0;
                    fourtable=fourtable+1;
                end;
            end;
            if tables(3,count)>0;
                tables(3,count)=tables(3,count)-1;
                if tables(3,count)<=0;
                    tables(3,count)=0;
                    sixtable=sixtable+1;
                end;
            end;
            count = count+1;
        end;
    else
        [i, twotable, fourtable, sixtable, wait, queue,
tables]=opentable(i, twotable, fourtable, sixtable, wait, queue, tables);
    end;
    a = sort(tables(1,:));
    b = sort(tables(2,:));
    c = sort(tables(3,:));
    tables = [a;b;c]
end;
```

Appendix D: statisticalData.m

```
%Normal Distribution Plot,
k = length(arrivals(7,:));
hours = (arrivals(7,89)-arrivals(7,1));
lambda = (k/hours); % Average arrival rate (customers/hour)
x = [-arrivals(7,89):1:arrivals(7,89)];
norm = normpdf(x,0,1);
figure;
```

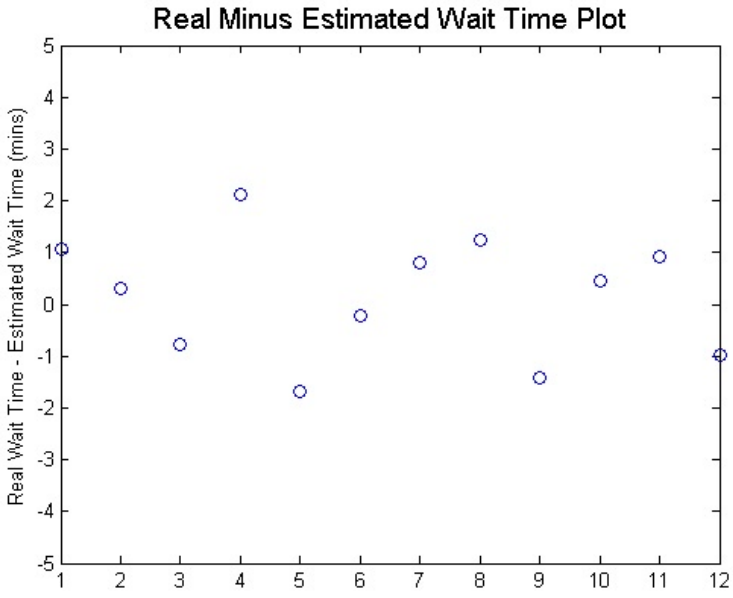
```
plot(x,norm);

%Mean and standard deviation for eating times
eatingTimes = arrivals(5,:);
meanEating = mean(arrivals(5,:));
stDevEating = std(arrivals(5,:));
```

Appendix E: Simulation Model Flowchart



This flowchart depicts the simulation model used to evaluate the accuracy of our wait time estimation method. This model takes randomly generated restaurant events as inputs.

Appendix F: Real Minus Estimated Wait Times Plot



This plot depicts the results of one simulation. The y-axis depicts the difference between a party's experienced wait time and their estimated wait time. The x-axis depicts each individual party that was in the waitlist at some point during the simulation.