Model for Airline Ticket Counter Staffing
Using Sabre™ StaffPlan Staff Forecasting and Planning System

By Lauren Duplantis, Francisco Villagran, Hamel Husain

Table of Contents

I.	Management Summary	Pages 3-4
II.	Background and Description of Problem	Page 4
III.	Analysis of Situation	Pages 5
IV.	Technical Description of Model	Pages 6-8
V.	Analysis and Managerial Interpretation	Pages 9
VI.	Conclusions and Recommendations	Pages 9-10
VII.	Appendix	Pages 11-26

I. Management Summary

Our project involved working with Sabre Inc. on a problem that they were having with their system StaffPlan that is maintained by their Airline Solutions Division. StaffPlan is an Airline Solutions system that minimizes the number of check-in agents at an airport based on the time of day, queue, airline, destinations, and passenger arrival curve categories. The problem that they wanted us to handle dealt with the queue, which is a single line of contacts who wait to be serviced. The queue is dependent on the service time, waiting time, quality target, and contact ratio. To simplify the problem they took out the contact ratio, so we just focused on the service time, waiting time, and quality target. The service time is the time it takes for a contact to be serviced from the time he/she gets to the counter to the time he/she leaves the counter. The waiting time is the amount of time that a contact waits starting from the time that they arrive to the time that they reach the ticket counter. The quality target is the percentage of total contacts whose wait times need to be under or equal to a maximum wait time. In this case Sabre wanted the maximum wait time to be 6 minutes and the quality target to be 80%. The problem with StaffPlan is that it uses a fixed service time, so the program wasn't as realistic as it could have been. In the real world, every contact takes a different amount of time at a counter. What our client wanted was for us to put variability in the service time to see how it affected the number of check-in agents needed, which we called ticket counters in our model. Our method of analysis involved first putting together the C program and then comparing our output to StaffPlan's output. Then once our output was identical to their output we put variability into the service time by creating a function that returned a random number that had a mean of 5 and was exponentially distributed. Once we got the output from the program we compared the output to that of StaffPlan's to see what the variance was. Then we analyzed our findings and looked at other ways of making the model more realistic, such as putting in a maximum wait time for those that normally would have to wait longer than six minutes because they meet the quality target.

II. Background and Description of the Problem Situation.

The purpose of our project was to analyze the differences in check-in agents needed if service time was fixed versus stochastic. This was a solution needed by the Airlines Solutions Department at Sabre, so that they could see if their program StaffPlan could be made more realistic. We decided to make a C program that was identical to StaffPlan's model and then put in the arrival curves that gave the number of contacts that arrived at each minute interval. StaffPlan's output was in 5 minute intervals, so once we got the output we had to average the number of counters open over five minute intervals, round the averages up, and return those as the final output. We then worked on debugging the program before we put variability into the service time. Then once we got the output needed we had to decide how to analyze the output and come up with solutions for Sabre and decide if we should analyze other factors. The questions that we asked ourselves before and during the project were: "How will we set up the code most efficiently to get the output needed by Sabre and will it be usable by them in the future?", "Will the variability affect the number of check-in agents needed?", "What other factors could be tested in the program that might affect StaffPlan's output?" Also we had to take into consideration how our findings would affect the Sabre and whom it would affect even outside the company. We weren't sure what the effects would be until we got the output needed, but we figured that this could affect the check-in agents because they would either work longer or shorter hours or lose their jobs. On the flip side, this would make airlines, who were customers of Sabre, more efficient and better able to serve passengers.

III. Analysis of the Situation

We started our analysis by putting together a simulation model by hand and going through the steps that Sabre did to determine the number of check-in agents needed. We used a fixed service time of 5 minutes. We used Sabre's arrival times and the number of contacts that arrived at each minute interval. Then we went through the simulation and calculated the times when contacts were done being serviced and when new counters had to be opened. This gave us a clear idea of how we needed to design our code and make it understandable. Then we did the same simulation with a variable service time to get an idea of what the difference would be in the number of counters open when we outputted that from a program. We decided that a C program would be the most useful for this scenario because Sabre wanted a model that they could manipulate and change in the future. The most challenging part of the problem was dealing with the quality target that affected how long a contact had to wait. If a contact's service ratio was 80 percent even if he/she/they didn't get served under or equal to six minutes then he/she/they didn't have to have a wait time below or equal to six minutes then he/she could wait as long as possible. This didn't seem practical to us, so we decided to put a maximum wait time for all contacts into the code, so the neglected contacts wouldn't have to wait unfeasibly long. Our goal of doing this was to make Sabre's model as realistic as possible.

Here is an example of what our hand simulation showed with a variable service time:

Contact	Arrival Time	Service Time	Time Served	Time Finished	Wait Time (min)	Service Ratio(%)	Open Counter?	Counter Used	New Wait	New Time Finished	New Service Ratio
1	9:00	5	9:00	9:05	0	0%	Yes	1			100%
2	9:01	9	9:05	9:14	4	100.00%	No	1			
3	9:02	1	9:14	9:15	12	66.67%	Yes	2	0	9:03	100.00%
4	9:02	2	9:03	9:05	1	100.00%	No	2			
5	9:03	3	9:14	9:17	11	80.00%	No	1			
6	9:03	12	9:05	9:17	2	83.33%	No	2			
7	9:03	7	9:17	9:20	14	71.43%	Yes	3	0	9:10	85.71%
8	9:04	9	9:10	9:19	6	87.50%	No	3			
9	9:05	1	9:17	9:22	12	77.78%	Yes	4	0	9:06	88.89%
10	9:05	2	9:19	9:24	14	80.00%	No	3			

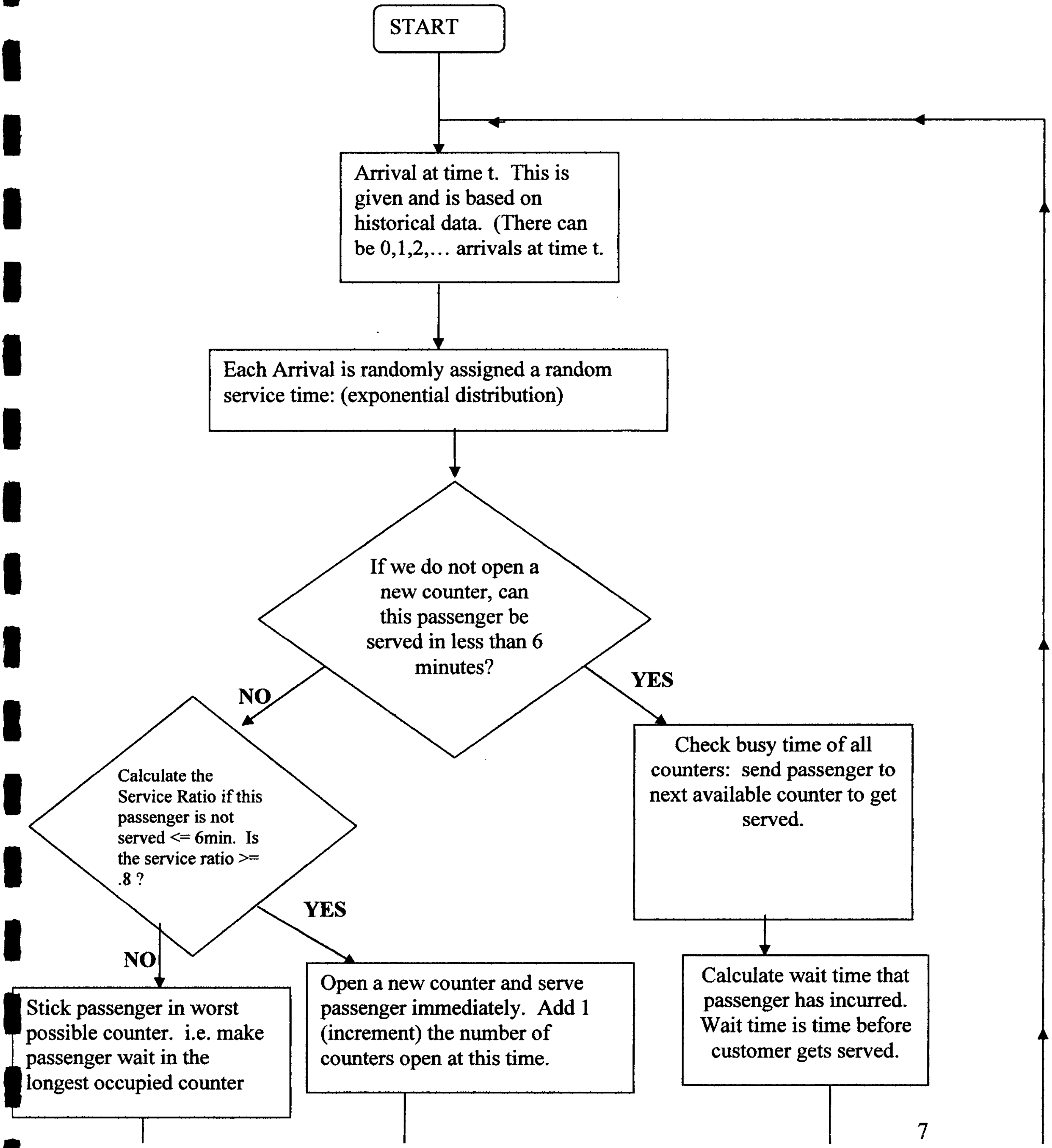
IV. Technical Description of the Model

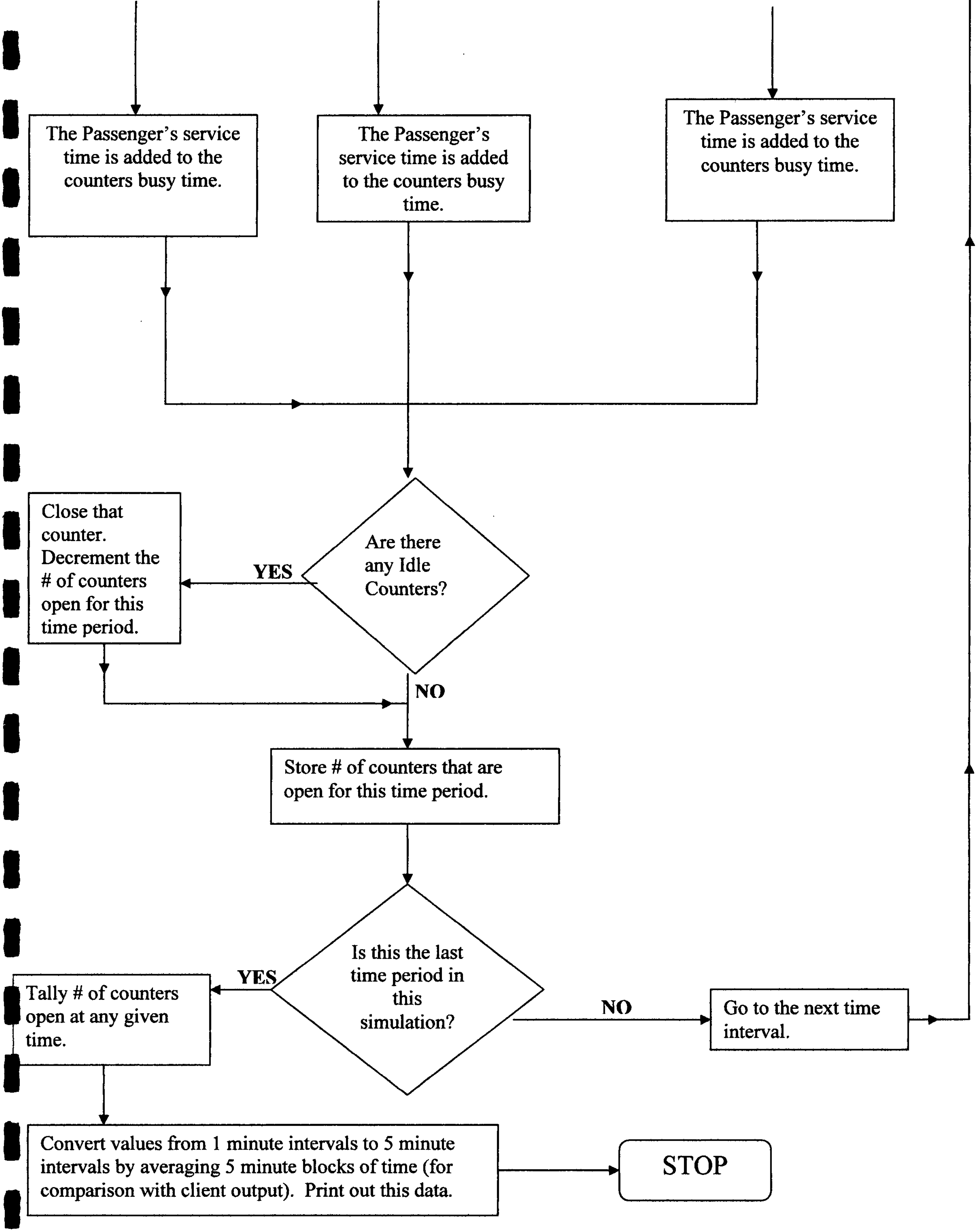
For this simulation, we decided to write a C program. First we had to accurately simulate Staffplan, Sabre's current solution and then add variability to that code. We read in the data via a file given to us by Sabre that specified arrivals per minute. To simplify this model, we analyzed six 24-hour periods of time. The five arrival curves range from small, only one flight, to large, a complete flight schedule. We ran these arrival curves in both programs, the Staffplan simulation with fixed service time, and the Staffplan simulation with variable service time. We were asked to assign each arrival an exponentially distributed random number to analyze the difference between having fixed service time and variable service time. A graph of the exponentially distributed random numbers we generated can be found in Appendix A.1. We made the variable service time program to run 1000 times so as to get a good estimate of overall variability effects. After running both programs, the output, counters open per 5-minute interval was compared for all six arrival curves. Both programs were written to also output the total wait per minute, i.e.: seventy people waited 4 minutes. This total wait time output was also compared for both programs. The findings are in the next section.

We also wanted to know what effect maximum wait would have on output. We ran both programs with an additional constraint of a maximum wait of 30 minutes and compared these results with the non-maximum wait programs, and those results can also be found in the next section. It was also suggested to try varying the maximum wait time, but the same trends appeared and that output has been omitted. The code for all of our programs can be found in Appendix A5-A8. We also included a program that generates the exponential service times and we show the distribution curve for a thousand random numbers (A8).

IV. Technical Description of the Model

SIMULATION FLOW CHART


START
Arrival at time t. This is given and is based on historical data. (There can be 0,1,2,… arrivals at time t.
Each Arrival is randomly assigned a random service time: (exponential distribution)
If we do not open a new counter, can this passenger be served in less than 6 minutes?
NO
YES
Calculate the Service Ratio if this passenger is not served <= 6min. Is the service ratio >= .8 ?
NO
YES
Check busy time of all counters: send passenger to next available counter to get served.
Stick passenger in worst possible counter. i.e. make passenger wait in the longest occupied counter
Open a new counter and serve passenger immediately. Add 1 (increment) the number of counters open at this time.
Calculate wait time that passenger has incurred. Wait time is time before customer gets served.


The Passenger's service time is added to the counters busy time.
The Passenger's service time is added to the counters busy time.
The Passenger's service time is added to the counters busy time.
Are there any Idle Counters?
YES
Close that counter. Decrement the # of counters open for this time period.
NO
Store # of counters that are open for this time period.
Is this the last time period in this simulation?
YES
Tally # of counters open at any given time.
NO
Go to the next time interval.
Convert values from 1 minute intervals to 5 minute intervals by averaging 5 minute blocks of time (for comparison with client output). Print out this data.
STOP

V. Analysis and Managerial Interpretation

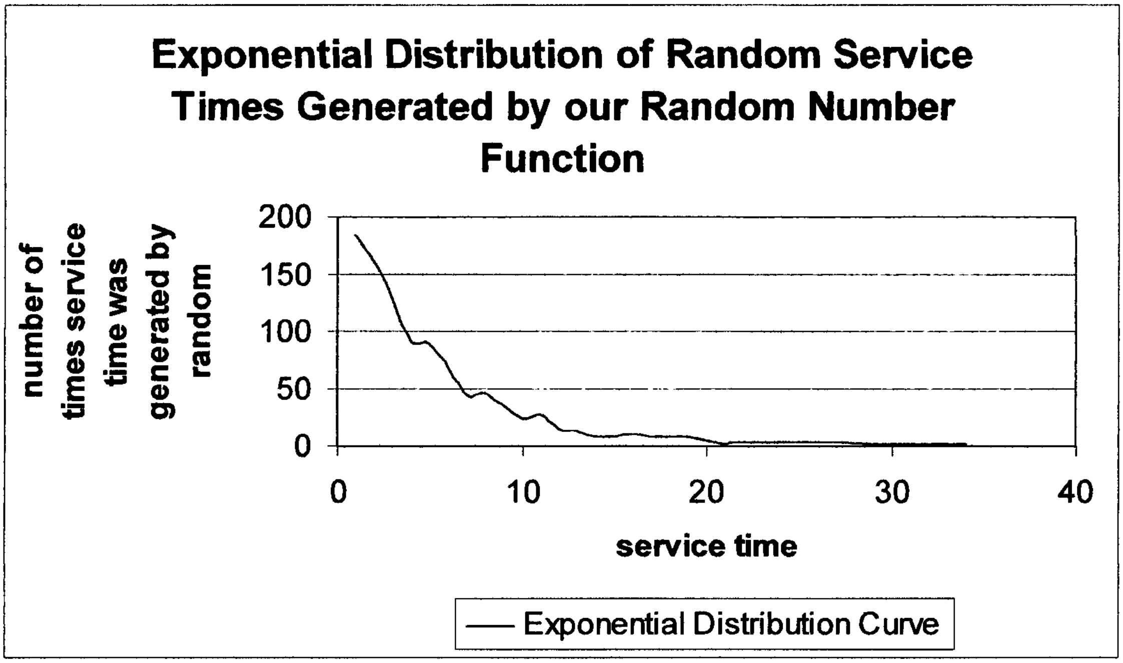
After comparing the output of the fixed and variable service time programs, we realized that as number of arrivals increase, so does the difference between the fixed and variable number of counters open, as can be clearly seen by the graphs below. In the Appendix A.3 it shows how variable service time causes more ticket counters to be opened in comparison to the model with fixed service times as bigger arrival curves are included. Therefore, this shows that as arrival curves become more dense the StaffPlan model with fixed service time becomes more unrealistic. Also, by implementing a maximum wait time to both programs we found that the number of counters opened was the same, but the distribution changed. We calculated the differences between the counters open from our model of StaffPlan and the new model with variable time and a max wait time of 30 minutes. In the graph in A.2 there is a greater difference between the fixed and variable with maximum wait time as more flights were entered into that model. The counters also needed to be opened earlier, in order that the maximum wait time constraint be met. In the graph A.4, there is also a significant difference between the number of counters open with the fixed time model and the fixed time model including a max wait time of 30 minutes as the arrival curves increased in arrival size.

VI. Conclusions and Critiques

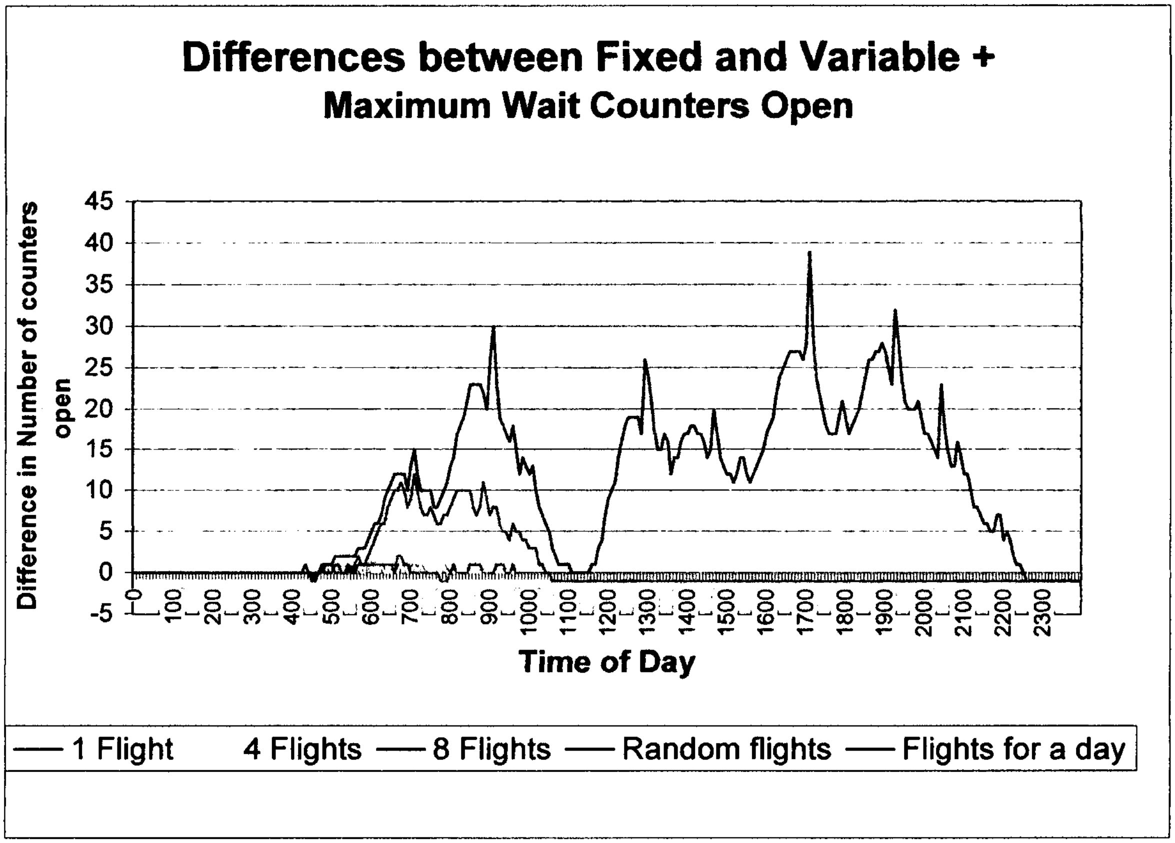
Based on the results we have obtained, we recommend that Sabre integrate variable service times into their Staffplan program, especially when dealing with large arrival curves. We feel this would make Staffplan more realistic by accounting for the fluctuation in time it takes to service different passengers. Furthermore, to make this model more realistic a fixed maximum wait time should be specified so that no passenger waits an unreasonable amount of time. The best way to handle maximum wait time is to assign each passenger their own max wait time. The reason for this is that passengers have different departure times and thus, different times by which they need to be serviced, leading to distinct maximum wait times. If our team would have had more time to work on this project, we would have observed actual ticket counter interactions at an airport. That way we could have compared our output to the actual situation to see how close our model with variable service time is to being realistic.

Appendix

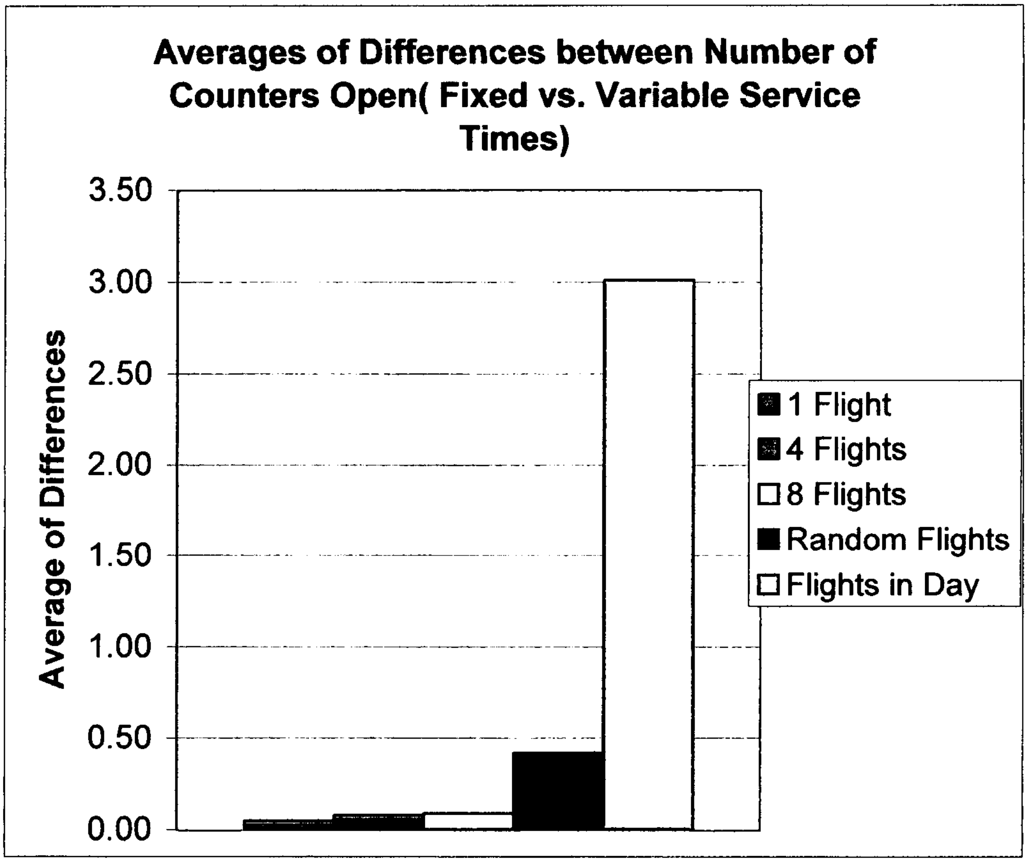
A.1


Exponential Distribution of Random Service Times Generated by our Random Number Function
number of times service time was generated by random
200
150
100
50
0
0
10
20
30
40
service time
Exponential Distribution Curve

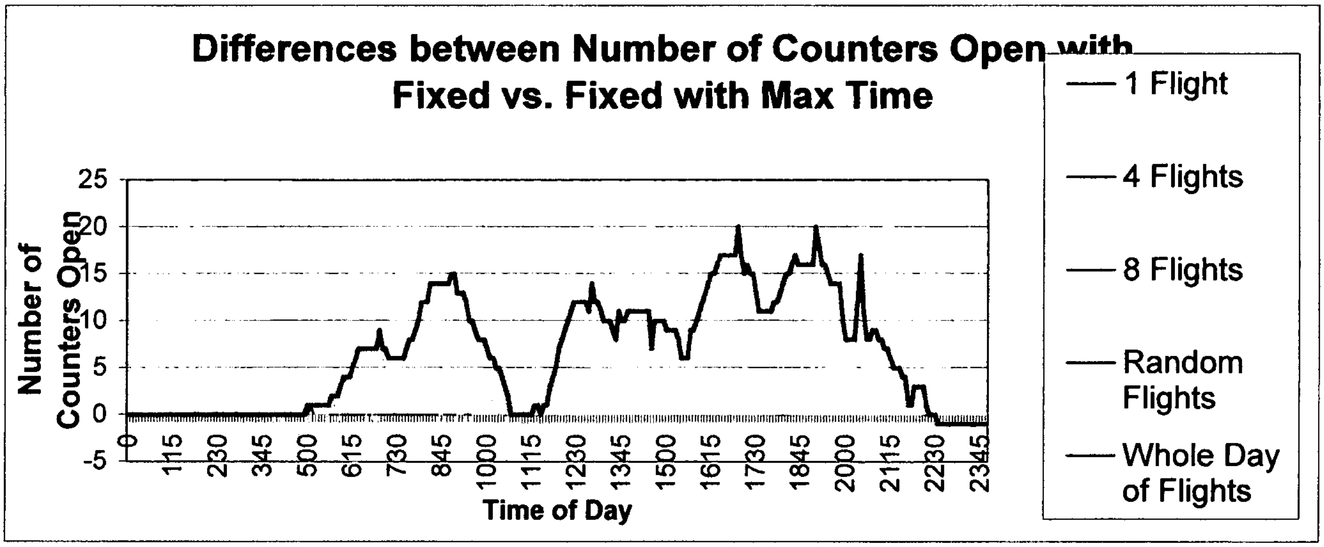
A.2


Differences between Fixed and Variable +
Maximum Wait Counters Open
Difference in Number of counters open
45
40
35
30
25
20
15
10
5
0
-5
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
1700
1800
1900
2000
2100
2200
2300
Time of Day
1 Flight
4 Flights
8 Flights
Random flights
Flights for a day

A.3


Averages of Differences between Number of Counters Open(Fixed vs. Variable Service Times)
Average of Differences
3.50
3.00
2.50
2.00
1.50
1.00
0.50
0.00
■ 1 Flight
■ 4 Flights
□ 8 Flights
■ Random Flights
□ Flights in Day

A.4


Differences between Number of Counters Open with Fixed vs. Fixed with Max Time
Number of Counters Open
25
20
15
10
5
0
-5
0
115
230
345
500
615
730
845
1000
1115
1230
1345
1500
1615
1730
1845
2000
2115
2230
2345
Time of Day
1 Flight
4 Flights
8 Flights
Random Flights
Whole Day of Flights

A.5

```
/* fixed.c */
/* Simulating the StaffPlan program */
/* Lauren Duplantis, Francisco Villagran and Hamel Husain */

#include <stdio.h>
#include <math.h>

int time=1440;             /* length of time to be looked at, in minutes*/
int maxC=40;               /* maximum available counters*/
float qualitytarget=0.8;/* quality target % */
int maxwait=6;         /* quality target wait time*/
int servicetime=5;     /* must make funtion for this*/


main()
{
        int arrivals[time][2], countersopen[time];
        int counters[time][maxC];
        int i,j,k,arvati,totalarv,servicedarv,queue,GRANDTOTAL;
        FILE *fparrivals,*fpoutput, *fpwait, *fpmatrix;
        float targetratio,totalarrivals;
        double average;

        /* open input and output files */

        fparrivals = fopen("ato8.log","r");
        if (fparrivals == NULL){
          printf("Error opening source file!\n");
          exit(1);
        }
    fpoutput = fopen("output.txt","w");
    if (fpoutput == NULL){
      printf("Error opening output file!\n");
    }
        fpwait = fopen("wait.txt","w");
        if (fpoutput == NULL){
          printf("Error opening wait output file!\n");
        }
        fpmatrix = fopen("matrix.txt","w");
    if (fpmatrix == NULL){
      printf("Error opening matrix file!\n");
          exit(1);
    }

    /* input arrival curve */
    /*input file should be in format: time, arrivals */

        GRANDTOTAL=0;
        for (i=0;i<time;i++){
     fscanf(fparrivals,"%d,%d",&arrivals[i][0],&arrivals[i][1]);
     arrivals[i][0]=(i/60)*100+(i % 60);
      /* sets time from 0000 to 2359 */
         GRANDTOTAL+=arrivals[i][1];/*counts # of arrivals*/
        }
        fclose(fparrivals);
     for (i=0;i<time;i++){
      for (j=0;j<maxC;j++){
       counters[i][j]=0;
      }
     }

// SIMULATION

         int g=0,waittime[GRANDTOTAL];
         float ratios[GRANDTOTAL];
     totalarv=0;
         totalarrivals=totalarv;
     servicedarv=0;
         targetratio=0.0;

     for (i=0;i<time;i++){
       arvati=arrivals[i][1];

       /* as minutes pass, reduce time busy*/
           if (i>0){
       for (j=0;j<maxC;j++){
                  if (counters[i-1][j]>0){
                   counters[i][j]=counters[i-1][j]-1;
                  }
       }}

           while (arvati>0){
        totalarv++;
                  totalarrivals=totalarv;
                  targetratio=servicedarv/totalarrivals;
            if (targetratio<qualitytarget){

//THIS IS THAT FINDING COUNTERS THING!!!

     queue=0;
     for (j=1;j<maxC;j++){
          if (counters[i][j]>0){
               if (counters[i][j]<counters[i][queue]){
                    queue=j;
               }
          }
     }

     if (counters[i][queue]>maxwait){
     queue=0;
     for (j=1;j<maxC;j++){
          if (counters[i][j]<counters[i][queue]){
               queue=j;
          }
     }
     }

// THIS IS END OF THAT FINDING COUNTERS THING!!!!

                  waittime[g]=counters[i][queue];
                  ratios[g]=targetratio;
                  g++;
```

```
                    counters[i][queue]+=servicetime;
                    servicedarv++;
              } else {
                    /* target ratio met, make arrival wait */
                    /* as long aspossible */
           queue=0;
           for (j=1;j<maxC;j++){
             if (counters[i][queue]<counters[i][j]) queue=j;
           }
//           if (counters[i][queue]<=maxwait) servicedarv++;
           waittime[g]=counters[i][queue];
                    ratios[g]=targetratio;
                    g++;
           counters[i][queue]+=servicetime;
              }
         arvati–;
        }
      }
          printf("total arrivals:%2d\n",totalarv);
          fprintf(fpmatrix,"time - arrivals - counter busy time\n");

          /* Display busy time per counter for all times */

          for (i=0;i<time;i++){
            fprintf(fpmatrix,"%4d - %2d -",arrivals[i][0],arrivals[i][1]);
            for (j=0;j<maxC;j++){
             if (counters[i][j]!=0) fprintf(fpmatrix," %2d",counters[i][j]);
            }
            fprintf(fpmatrix,"\n");
          }

          for (i=0;i<time;i++){
            countersopen[i]=0;
          }

//        fprintf(fpoutput,"Arrivals serviced within %2d minutes: %2d\n",
//                  maxwait,servicedarv);
//        fprintf(fpoutput,"Total arrivals: %2d\n",totalarv);
          totalarrivals=totalarv;
          targetratio=servicedarv/totalarrivals;
          targetratio=targetratio*100;
          printf("service ratio: %2.4f\n",targetratio);

//    Commented out, counters needed per time, not averaged yet
//        fprintf(fpoutput,"\nTime, Counters needed\n");
          for (i=0;i<time;i++){
            for (j=0;j<maxC;j++){
             if (counters[i][j]>0) countersopen[i]++;
            }
//          if (countersopen[i]!=0)fprintf(fpoutput,"%4d,%3d\n",
//          arrivals[i][0],countersopen[i]);
          }

          int waittotals[maxwait+2];
          for (i=0;i<maxwait+2;i++){
            waittotals[i]=0;
          }
          /* Average number of counters opened, and round up */
//        fprintf(fpoutput,"\n\nTime,Counters Needed\n");
          j=time/5;
          int ACO[j][2],roundedaverage=0;
      for(i=0;i<j;i++){
       ACO[i][0]=arrivals[i*5][0];
            average=(countersopen[i*5]+countersopen[i*5+1]+
                    countersopen[i*5+2]+countersopen[i*5+3]+
                    countersopen[i*5+4])/5.0;
            roundedaverage=ceil(average);
            ACO[i][1]=roundedaverage;
       fprintf(fpoutput,"%4d, %4d\n",ACO[i][0],ACO[i][1]);

      }

          /* waiting time calculations */
          fprintf(fpwait,"Arrival - Wait time - Target Ratio\n");
          for (i=0;i<GRANDTOTAL;i++){
            fprintf(fpwait,"%4d - %4d    -  %2.4f\n"
                 ,i+1,waittime[i],ratios[i]);
          }

          for (i=0;i<GRANDTOTAL;i++){
            j=maxwait+1;
            k=maxwait+1;
            while (k>=0){
                    if (waittime[i]<j)j–;
                    k–;
            }
            waittotals[j]++;
          }
          fprintf(fpwait,"\n\n\nWaiting Time totals\n\n");
          fprintf(fpwait,"min, total\n");
          for (i=0;i<maxwait+2;i++){
            fprintf(fpwait,"%3d, %3d\n",i,waittotals[i]);
          }
          fprintf(fpwait,"Note: The last value is sum of all wait times ");
          fprintf(fpwait,"greater than %d minutes",maxwait);

          fclose(fpwait);
          fclose(fpoutput);
          fclose(fpmatrix);
}
```

A.6

```
/* project.c */
/* Staffplan + variable service time Simulated 1000 times */
/* Adding variability to the StaffPlan simulation program */
/* Lauren Duplantis, Francisco Villagran and Hamel Husain */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int time=1440;               /* length of time to be looked at, in minutes*/
int maxC=30;                /* maximum available counters*/
float qualitytarget=0.8;/* quality target % */
int maxwait=6;        /* quality target wait time*/
int servicetime=5;
int runs=1000;              /* number of runs in simulation */
double lambda=5.0;          /* mean of service time*/

double expon(double x);

main()
{
        int arrivals[time][2], countersopen[time];
        int counters[time][maxC],run;
        int i,j,k,arvati,totalarv,servicedarv,queue,GRANDTOTAL;
        FILE *fparrivals,*fpoutput, *fpwait, *fpmatrix;
        float targetratio,totalarrivals,fruns=runs;
        double average;
        j=time/5;
        float TACO[j][2]; /* contains total average counters open*/

        /* open input and output files */

        fparrivals = fopen("ato2.log","r");
        if (fparrivals == NULL){
          printf("Error opening source file!\n");
          exit(1);
        }
    fpoutput = fopen("projectoutput.txt","w");
    if (fpoutput == NULL){
      printf("Error opening output file!\n");
          exit(1);
    }
        fpwait = fopen("projectmaxwait.txt","w");
        if (fpoutput == NULL){
          printf("Error opening wait output file!\n");
        }
//      fpmatrix = fopen("matrix.txt","w");
//  if (fparrivals == NULL){
//    printf("Error opening matrix file!\n");
//        exit(1);
//  }

    /* input arrival curve */
    /*input file should be in format: time, arrivals */

        GRANDTOTAL=0;
        for (i=0;i<time;i++){
     fscanf(fparrivals,"%d,%d",&arrivals[i][0],&arrivals[i][1]);
     arrivals[i][0]=(i/60)*100+(i % 60);
      /* sets time from 0000 to 2359 */
         GRANDTOTAL+=arrivals[i][1];/*counts # of arrivals*/
        }
        fclose(fparrivals);

        for (i=0;i<j;i++){
          TACO[i][0]=arrivals[i*5][0];
          TACO[i][1]=0.0;
        }
     int waittotals[maxwait+2];
     for (i=0;i<maxwait+2;i++){
       waittotals[i]=0;
     }


// SIMULATION

  for (run=0;run<runs;run++){

     for (i=0;i<time;i++){
      for (j=0;j<maxC;j++){
       counters[i][j]=0;
      }
     }

         int g=0,waittime[GRANDTOTAL];
     totalarv=0;
         totalarrivals=totalarv;
     servicedarv=0;
         targetratio=0.0;

     for (i=0;i<time;i++){
       arvati=arrivals[i][1];

       /* as minutes pass, reduce time busy*/
           if (i>0){
       for (j=0;j<maxC;j++){
                  if (counters[i-1][j]>0){
                   counters[i][j]=counters[i-1][j]-1;
                  }
       }}

           while (arvati>0){
        totalarv++;
                  totalarrivals=totalarv;
                  targetratio=servicedarv/totalarrivals;
            if (targetratio<qualitytarget){

//THIS IS THAT FINDING COUNTERS THING!!!
     queue=0;
     for (j=1;j<maxC;j++){
          if (counters[i][j]>0){
               if (counters[i][j]<counters[i][queue]){
                    queue=j;
               }
          }
     }

     if (counters[i][queue]>maxwait){
     queue=0;
     for (j=1;j<maxC;j++){
          if (counters[i][j]<counters[i][queue]){
               queue=j;
          }
     }
     }
// THIS IS END OF THAT FINDING COUNTERS THING!!!!


                             waittime[g]=counters[i][queue];
                             g++;
                             counters[i][queue]+=expon(lambda);
                             servicedarv++;
                    } else {
                             /* target ratio met, make arrival wait */
                             /* as long aspossible */
                 queue=maxC;
                for (j=0;j<maxC;j++){
                             if (counters[i][queue]<counters[i][j]) queue=j;
                }
//                if (counters[i][queue]<=maxwait) servicedarv++;
                waittime[g]=counters[i][queue];
                             g++;
                counters[i][queue]+=expon(lambda);
                    }
             arvati–;
           }
        }

              for (i=0;i<time;i++){
                 countersopen[i]=0;
              }

              for (i=0;i<time;i++){
                 for (j=0;j<maxC;j++){
                   if (counters[i][j]>0) countersopen[i]++;
                 }
              }

              /* Average number of counters opened, and round up */
//            fprintf(fpoutput,"\n\nTime,Counters Needed\n");
              j=time/5;
              int ACO[j][2],roundedaverage=0;
        for(i=0;i<j;i++){
          ACO[i][0]=arrivals[i*5][0];
                average=(countersopen[i*5]+countersopen[i*5+1]+
                             countersopen[i*5+2]+countersopen[i*5+3]+
                             countersopen[i*5+4])/5.0;
                roundedaverage=ceil(average);
                ACO[i][1]=roundedaverage;
                TACO[i][1]+=ACO[i][1];
        }

              for (i=0;i<GRANDTOTAL;i++){
                 j=maxwait+1;
                 k=maxwait+1;
                 while (k>=0){
                             if (waittime[i]<j)j–;
                             k--;
                 }
                 waittotals[j]++;
              }
```

```
// END OF SIMULATION
  }
//        fprintf(fpoutput,"After %4d simulations,",runs);
//        fprintf(fpoutput,"here are the average counters open:\n");
//        fprintf(fpoutput,"\ntime - Average counters open\n");
          k=0;
          j=time/5;
          for (i=0;i<j;i++){
            TACO[i][1]=TACO[i][1]/fruns;
            k=rint(TACO[i][1]);
            fprintf(fpoutput,"%4.0f, %2d\n",TACO[i][0],k);
          }

          float waittotal;
      fprintf(fpwait,"\n\n\nWaiting Time totals\n\n");
      fprintf(fpwait,"min, total\n");
      for (i=0;i<maxwait+2;i++){
            waittotal=waittotals[i]/fruns;
        fprintf(fpwait,"%3d, %3.2f\n",i,waittotal);
      }
      fprintf(fpwait,"Note: The last value is sum of all wait times ");
      fprintf(fpwait,"greater than %d minutes",maxwait);

          fclose(fpwait);
          fclose(fpoutput);
}

double expon(double x)
{
 double z;                  // Uniform random number (0 < z < 1)
 double expvalue;              // Computed exponential value to b
 int rounded;

 do
 {
  z = ((double) rand() / RAND_MAX);
 }
 while ((z == 0) || (z == 1));

 // Compute exponential random variable using inversion method
 expvalue = -x * log(z);
 rounded=ceil(expvalue);
 return(rounded);
}
```

A.7

```
/* maxwaitfinal.c */
/* Simulates StaffPlan program + a global maxwait of 30 minutes */
/* Lauren Duplantis, Francisco Villagran and Hamel Husain */
```

```
#include <stdio.h>
#include <math.h>

int time=1440;              /* length of time to be looked at, in minutes*/
int maxC=40;                /* maximum available counters*/
float qualitytarget=0.8;/* quality target % */
int maxwait=6;         /* quality target wait time*/
int servicetime=5;
int feasiblemaxwait=30;  /* nobody can wait more than this */

main()
{
        int arrivals[time][2], countersopen[time];
        int counters[time][maxC];
        int i,j,k,arvati,totalarv,servicedarv,queue,GRANDTOTAL;
        FILE *fparrivals,*fpoutput, *fpwait, *fpmatrix;
        float targetratio,totalarrivals;
        double average;

        /* open input and output files */

        fparrivals = fopen("ato8.log","r");
        if (fparrivals == NULL){
          printf("Error opening source file!\n");
          exit(1);
        }
    fpoutput = fopen("maxoutput.txt","w");
    if (fpoutput == NULL){
      printf("Error opening output file!\n");
    }
        fpwait = fopen("maxwait.txt","w");
        if (fpoutput == NULL){
          printf("Error opening wait output file!\n");
        }
        fpmatrix = fopen("maxmatrix.txt","w");
    if (fpmatrix == NULL){
      printf("Error opening matrix file!\n");
          exit(1);
    }

    /* input arrival curve */
    /*input file should be in format: time, arrivals */

        GRANDTOTAL=0;
        for (i=0;i<time;i++){
     fscanf(fparrivals,"%d,%d",&arrivals[i][0],&arrivals[i][1]);
     arrivals[i][0]=(i/60)*100+(i % 60);
      /* sets time from 0000 to 2359 */
         GRANDTOTAL+=arrivals[i][1];/*counts # of arrivals*/
        }
        fclose(fparrivals);

    for (i=0;i<time;i++){
     for (j=0;j<maxC;j++){
      counters[i][j]=0;
     }
    }
```

```
// SIMULATION

        int g=0,waittime[GRANDTOTAL];
        float ratios[GRANDTOTAL];
    totalarv=0;
        totalarrivals=totalarv;
    servicedarv=0;
        targetratio=0.0;

    for (i=0;i<time;i++){
      arvati=arrivals[i][1];

      /* as minutes pass, reduce time busy*/
          if (i>0){
      for (j=0;j<maxC;j++){
                if (counters[i-1][j]>0){
                  counters[i][j]=counters[i-1][j]-1;
                }
      }}

          while (arvati>0){
       totalarv++;
                totalarrivals=totalarv;
                targetratio=servicedarv/totalarrivals;
           if (targetratio<qualitytarget){

             queue=0;
        for (j=1;j<maxC;j++){
                if (counters[i][j]>0){
                if (counters[i][j]<counters[i][queue]){
                        queue=j;
                }
                }
        }

        if (counters[i][queue]>maxwait){
             queue=0;
                for (j=1;j<maxC;j++){
                        if (counters[i][j]<counters[i][queue]){
                        queue=j;
                        }
                }
        }

                waittime[g]=counters[i][queue];
                ratios[g]=targetratio;
                g++;
                counters[i][queue]+=servicetime;
                servicedarv++;
           } else {
                /* target ratio met, make arrival wait */
                /* as long as possible, but not more than 30 */
         queue=maxC;
         for (j=0;j<maxC;j++){
                        if (counters[i][j]<feasiblemaxwait){
                if (counters[i][queue]<counters[i][j]) queue=j;
                        }
```

```
            }
//            if (counters[i][queue]<=maxwait) servicedarv++;
           waittime[g]=counters[i][queue];
                   ratios[g]=targetratio;
                   g++;
           counters[i][queue]+=servicetime;
             }
         arvati--;
        }
      }
         printf("total arrivals:%2d\n",totalarv);
         fprintf(fpmatrix,"time - arrivals - counter busy time\n");

         /* Display busy time per counter for all times */

         for (i=0;i<time;i++){
           fprintf(fpmatrix,"%4d - %2d -",arrivals[i][0],arrivals[i][1]);
           for (j=0;j<maxC;j++){
            if (counters[i][j]!=0) fprintf(fpmatrix," %2d",counters[i][j]);
           }
           fprintf(fpmatrix,"\n");
         }

         for (i=0;i<time;i++){
           countersopen[i]=0;
         }

//       fprintf(fpoutput,"Arrivals serviced within %2d minutes: %2d\n",
//                 maxwait,servicedarv);
//       fprintf(fpoutput,"Total arrivals: %2d\n",totalarv);
         totalarrivals=totalarv;
         targetratio=servicedarv/totalarrivals;
         targetratio=targetratio*100;
//       printf("service ratio: %2.4f\n",targetratio);

//     Commented out, counters needed per time, not averaged yet
//       fprintf(fpoutput,"\nTime, Counters needed\n");
         for (i=0;i<time;i++){
           for (j=0;j<maxC;j++){
             if (counters[i][j]>0) countersopen[i]++;
           }
//         if (countersopen[i]!=0)fprintf(fpoutput,"%4d,%3d\n",
//         arrivals[i][0],countersopen[i]);
         }

         int waittotals[maxwait+2];
         for (i=0;i<maxwait+2;i++){
           waittotals[i]=0;
         }

         /* Average number of counters opened, and round up */
//       fprintf(fpoutput,"\n\nTime,Counters Needed\n");
         j=time/5;
         int ACO[j][2],roundedaverage=0;
     for(i=0;i<j;i++){
       ACO[i][0]=arrivals[i*5][0];
           average=(countersopen[i*5]+countersopen[i*5+1]+
                   countersopen[i*5+2]+countersopen[i*5+3]+
              countersopen[i*5+4])/5.0;
        roundedaverage=ceil(average);
        ACO[i][1]=roundedaverage;
     fprintf(fpoutput,"%4d, %4d\n",ACO[i][0],ACO[i][1]);

    }

       /* waiting time calculations */
       fprintf(fpwait,"Arrival - Wait time - Target Ratio\n");
       for (i=0;i<GRANDTOTAL;i++){
        fprintf(fpwait,"%4d - %4d    -  %2.4f\n"
            ,i+1,waittime[i],ratios[i]);
       }

       for (i=0;i<GRANDTOTAL;i++){
        j=maxwait+1;
        k=maxwait+1;
        while (k>=0){
             if (waittime[i]<j)j--;
             k--;
        }
        waittotals[j]++;
       }
       fprintf(fpwait,"\n\n\nWaiting Time totals\n\n");
       fprintf(fpwait,"min, total\n");
       for (i=0;i<maxwait+2;i++){
        fprintf(fpwait,"%3d, %3d\n",i,waittotals[i]);
       }
       fprintf(fpwait,"Note: The last value is sum of all wait times ");
       fprintf(fpwait,"greater than %d minutes ",maxwait);

       fclose(fpwait);
       fclose(fpoutput);
       fclose(fpmatrix);
}
```

A.8

```
/*this program performs generation of random numbers that are exponentially
distributed with a mean of 5 minutes*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double expon(double x);

main()
{
    float numbers[1000];
       int i=0;
       double lambda=5.0,average=0.0;
       FILE *fpoutput;

       fpoutput = fopen("exponential.txt","w");
    for (i=0;i<1000;i++){
      numbers[i]=expon(lambda);
      fprintf(fpoutput,"%2.3f \n",numbers[i]);
          average+=numbers[i];
     }
         average=average/1000.0;
         fprintf(fpoutput,"%2.5f",average);
}
double expon(double x)
{
 double z;                    // Uniform random number (0 < z < 1)
 double exp_value;              // Computed exponential value to b
 int rounded;

 do
 {
  z = ((double) rand() / RAND_MAX);
 }
 while ((z == 0) || (z == 1));

 // Compute exponential random variable using inversion method
 exp_value = -x * log(z);
 rounded=ceil(exp_value);
 return(rounded);
}
```