DATA 604
Final Project
Bank Teller Simulation

Sie Siong Wong

2022 Summer

Problem Statement

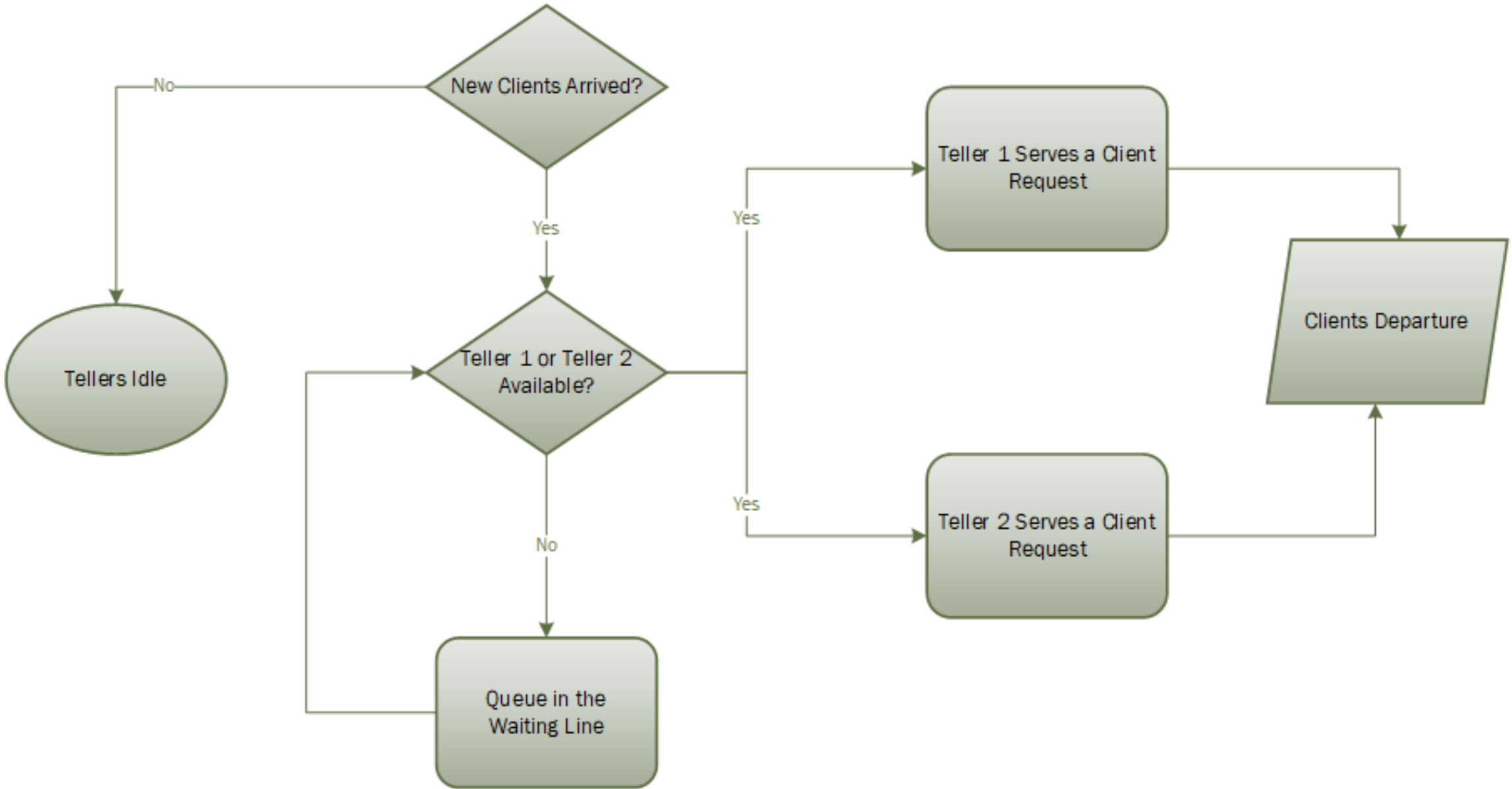
- How is changing the number of bank tellers might affect the waiting time for clients.

- Bank executives don't want to hire too many tellers, and too little tellers that can cause customers to wait in line too long which cause customer dissatisfaction.

- This goal of developing this simulation program is to help bank executives to determine an optimal number of tellers they want to hire for


1
BANK

Bank Tellers Flow-Chart Model



Simulation Codes

- The libraries that I use to develop this simulation program is as below.
  - simpy
  - random
  - numpy
  - modsim
- The simulation program was built mainly based on the simpy. I use other libraries in additional to the simpy because of some functions. For example, random library has the randomized exponential distribution function which is a critical function for generating client arrival time.

Simulation Codes Cont.

- There are 3 functions with parameters built into this simulation.
  - client_generator(env, mean_inter_arrival_time, mean_serve_time, teller)
  - activity_generator(env, mean_serve_time, teller, c_id)
  - run_simulation(capacity, arrival_time, serve_time)
- The details of these functions are showing in the following slides.

Simulation Codes Cont.

```
# Generator function for bringing new clients into the model
def client_generator(env, mean_inter_arrival_time, mean_serve_time, teller):

    """ env = simulation environment
        mean_inter_arrival_time =  mean inter-arrival time for clients coming in
        mean_serve_time =  mean time clients spend with a teller
        teller =  teller resource """

    # Initiate client id
    c_id = 0

    # Keep doing this indefinitely
    while True:

        # Create an instance of client activity generator
        wp = activity_generator(env, mean_serve_time, teller, c_id)

        # Run the activity generator
        env.process(wp)

        # Calculate the time until next cient arrives using randomly exponential distribution
        t = random.expovariate(1.0 / mean_inter_arrival_time)

        # Freeze this function until inter-arrival time has elapsed
        yield env.timeout(t)

        # Increment the client id after inter-arrival time has elapsed
        c_id += 1
```

Simulation Codes Cont.

```
# Generator function for the activities that clients will queue for
def activity_generator(env, mean_serve_time, teller, c_id):

    """ env = simulation environment
        mean_serve_time =  mean time clients spend with a teller
        teller =  teller resource
        c_id = client id """

    # Grab a current simulation time when a client start to queue
    time_entered_queue_for_client = env.now
    frame.at[c_id, 'client_id'] = c_id
    frame.at[c_id, 'time_entered_queue'] = time_entered_queue_for_client
    #print("Client ", c_id, " entered queue at ", time_entered_queue_for_client, sep="")

    # Use a "with" statement to indicate that all the codes within it needs to be done before it can released
    # Call the request function of the client resource
    with teller.request() as req:

        # Call a yield until a teller is available
        yield req

        # Grab the current simulation time clients left queue
        time_left_queue_for_client = env.now
        frame.at[c_id, 'time_left_queue'] = time_left_queue_for_client
        #print("Client ", c_id, " left queue at ", time_left_queue_for_client, sep="")
```

Simulation Codes Cont.

```
        # Calcualte the time clients spend in waiting line
        time_in_queue_for_client = (time_left_queue_for_client - time_entered_queue_for_client)
        frame.at[c_id, 'time_in_queue'] = time_in_queue_for_client
        #print("Client ", c_id, " queued for ", time_in_queue_for_client, " minutes.", sep="")

        # Calculate the time clients spend with a teller using randomly exponential distribution
        sampled_serve_time = random.expovariate(1.0 / mean_serve_time)
        frame.at[c_id, 'time_serve'] = sampled_serve_time

        # Freeze this function until served time has elapsed
        yield env.timeout(sampled_serve_time)
        frame.at[c_id, 'time_finished_at'] = env.now
        #print("***Client ", c_id, " finished at ", env.now, sep="")
```

Simulation Codes Cont.

```
def run_simulation(capacity, arrival_time, serve_time):

    # Setup a simpy environment
    env = simpy.Environment()

    # Create a simpy resource and give it to the simulation environment and the capacity
    # Capacity here is the number teller available.
    teller = simpy.Resource(env, capacity=capacity)

    # Set model parameter values where mean_inter_arrival is the mean inter-arrival time for clients coming in,
    # and mean_served is the mean time clients will spend with a teller.
    mean_inter_arrival_time = arrival_time
    mean_serve_time = serve_time

    # Set seed
    seed(12345)

    # Startup client generator function to start creating new clients
    env.process(client_generator(env, mean_inter_arrival_time, mean_serve_time, teller))

    # Set the simulation to run for 480 minutes (8 hours)
    env.run(until=480)
```

Simulation Codes Cont.

```
    # Calculate the idle time of no customers to serve until next customer arrival
    for c_id in range(len(frame)):
        if frame.at[c_id,'client_id'] == 0:
            frame.at[c_id, 'time_idle'] = 0

        elif frame.at[c_id, 'time_entered_queue'] - frame.at[c_id-1, 'time_finished_at'] < 0:
            frame.at[c_id, 'time_idle'] = 0

        else:
            frame.at[c_id, 'time_idle'] = frame.at[c_id, 'time_entered_queue'] - frame.at[c_id-1, 'time_finished_at']

    return frame
```

Results

- Below is a dataframe return from run simulation function:

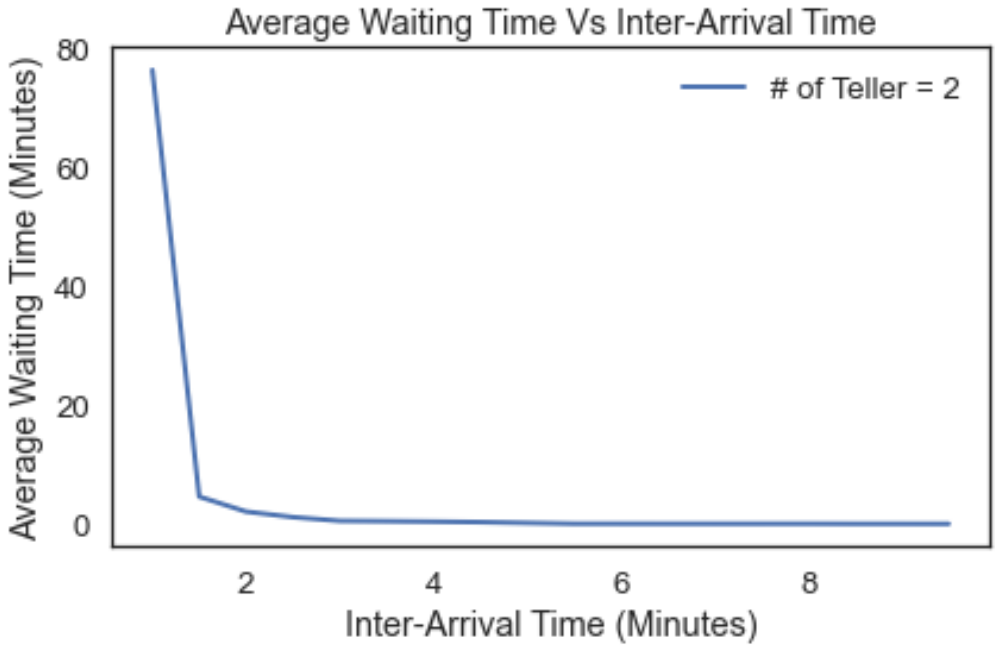
```
run_simulation(2, 2.5, 3).head(25)
```

- E.g. Client 0 arrived at the bank without queueing and went straight to the counter and left bank at 0.030664 minutes. All tellers have been idled for 1.316627 minutes until the second client (id=1) arrived at the bank at 1.347291 minutes.

	client_id	time_idle	time_entered_queue	time_left_queue	time_serve	time_in_queue	time_finished_at
0	0	0	0	0	0.030664	0	0.030664
1	1	1.316627	1.347291	1.347291	1.064201	0.0	2.411492
2	2	3.296174	5.707666	5.707666	0.645754	0.0	6.35342
3	3	0.503039	6.85646	6.85646	0.529094	0.0	7.385554
4	4	1.55773	8.943284	8.943284	1.701851	0.0	10.645134
5	5	0	9.275018	9.275018	0.57473	0.0	9.849748

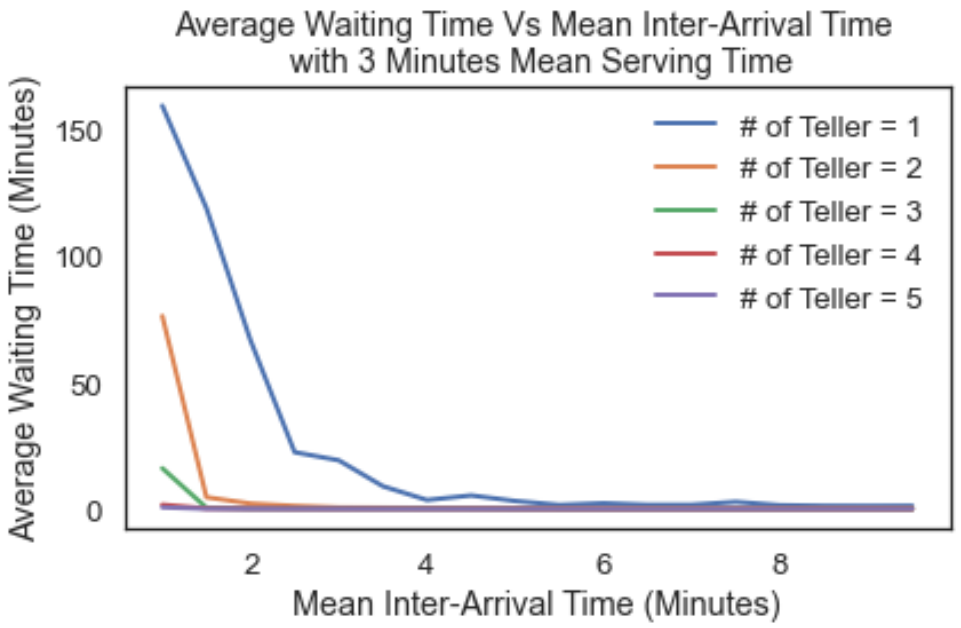
Results Cont.

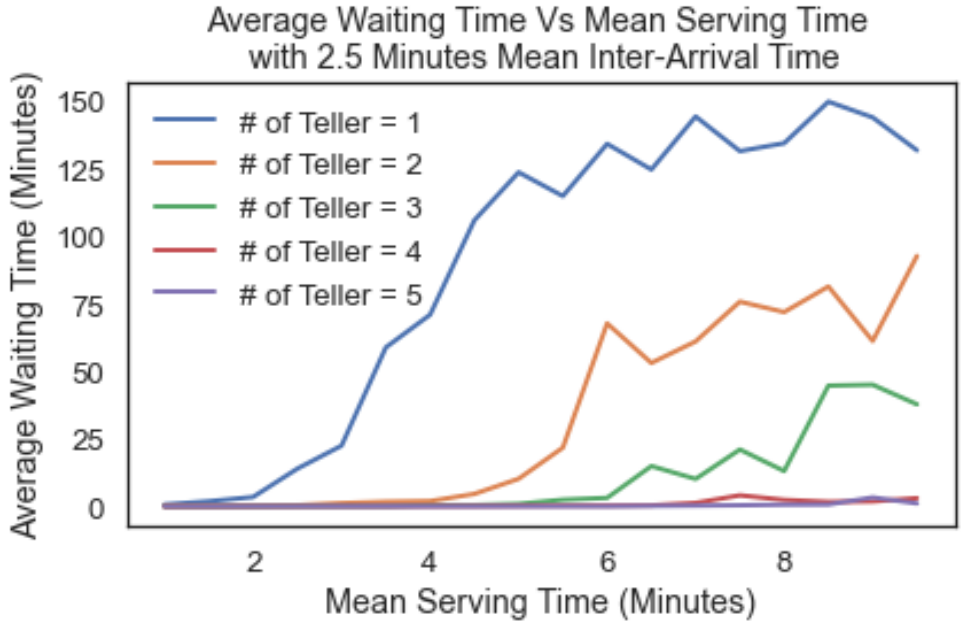
- Average waiting time vs. variation of mean inter-arrival time.
- Average waiting time dropped sharply when the gap of client arrival time is at 1.5 minute.



Results Cont.

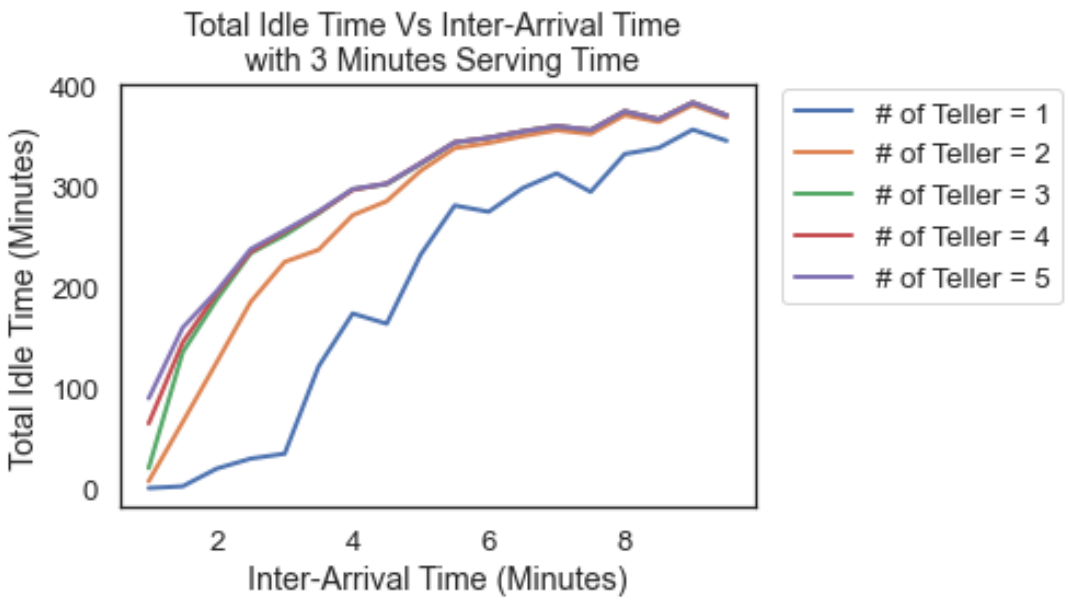
- Let’s see how the average waiting time changes for different number of tellers in place with variation of mean inter-arrival time and mean serving time.

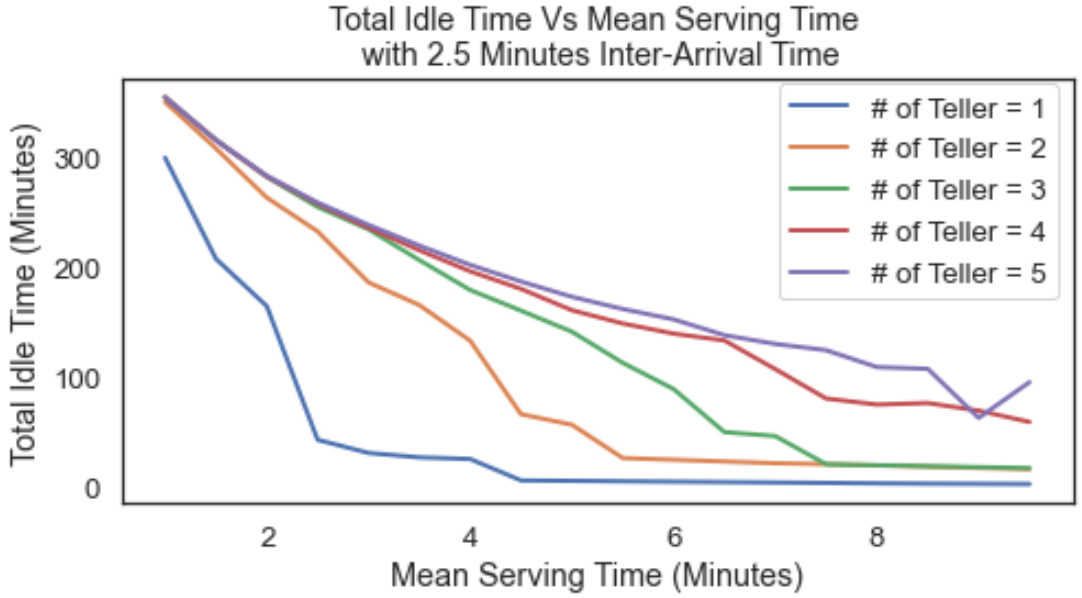




Results Cont.

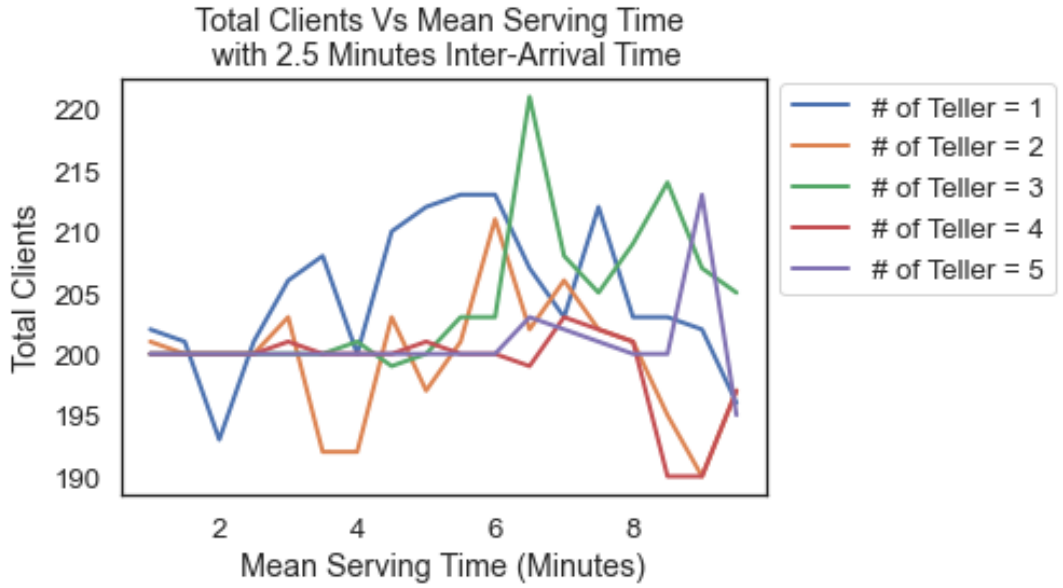
- Now, let's see how the total idle time changes for different number of tellers in place with variation of mean inter-arrival time and mean serving time.

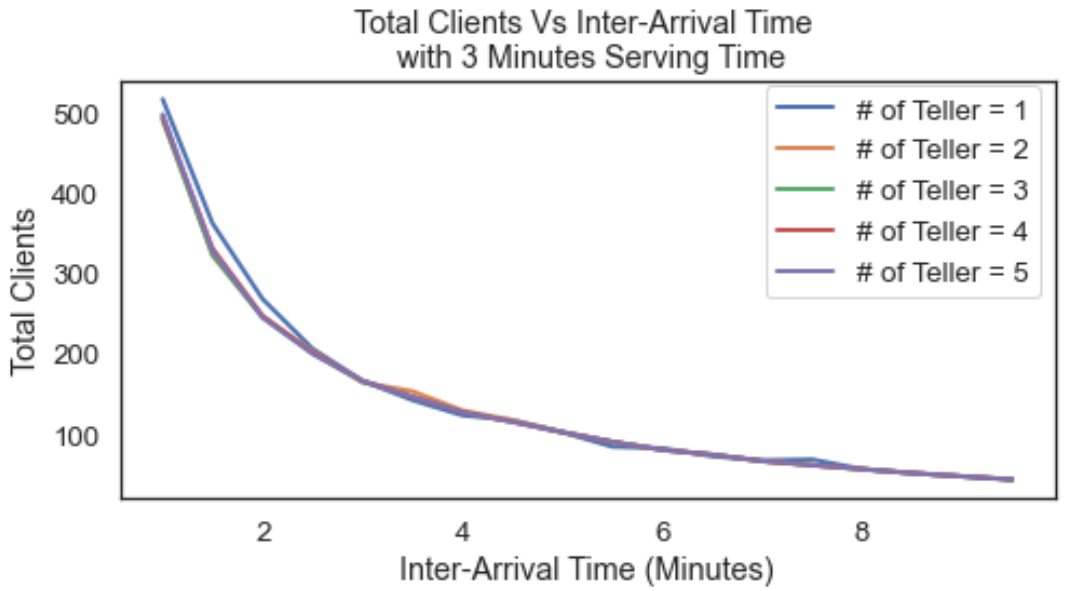




Results Cont.

- Finally, let's see how the total number of clients changes for different number of tellers in place with variation of mean inter-arrival time and mean serving time.





Verification and Validation

- 2 tellers is the optimal number of tellers to hire to meeting the goal of average waiting time at 3 minutes or less.
- According to this similar simulation article (*Hammond and Mahesh 1995*) , majority of the banks has a desired policy to provide service for their clients within 3 minutes.
- In the article, it also mentioned that "*The two tellers calculated to be adequate 95 % of the time under average conditions provided the desired level of service only 68 % of the time in the simulation model.*“
- The optimal parameter values indicated from the simulation results above are 3 minutes of mean inter-arrival time and 5.5 minutes of serving as shown in below.

Verification and Validation Cont.

- Below calculation results verify and justify the validity of the simulation model.

```
frame = TimeFrame(columns=['client_id','time_idle','time_entered_queue','time_left_queue',
                           'time_serve','time_in_queue','time_finished_at'])
print("Average waiting time is ", run_simulation(2, 3, 5.5)['time_in_queue'].mean(), " minutes")
```

```
Average waiting time is  2.460972554543405  minutes
```

```
frame = TimeFrame(columns=['client_id','time_idle','time_entered_queue','time_left_queue',
                           'time_serve','time_in_queue','time_finished_at'])
print("Total idle time is ", run_simulation(2, 3, 5.5)['time_idle'].sum(), " minutes")
```

```
Total idle time is  98.46303053621017  minutes
```

```
count_3min = run_simulation(2, 3, 5.5).apply(lambda x : True if x['time_in_queue'] < 3 else False, axis = 1)
pct_3min = round(np.sum(count_3min)/len(count_3min)*100, 2)

print("The percentage of clients whose waiting time is 3 minutes or less is ", pct_3min, "%")
```

```
The percentage of clients whose waiting time is 3 minutes or less is  70.97 %
```

Conclusions

- This bank teller simulation study has demonstrated that bank executives can use this simulation program to figure out the optimal number of tellers based on the mean inter-arrival time and the mean serving time for a client.
- The number of clients coming in every working day looking for tellers vary greatly across different regions and communities. For example, the bank located in high population density areas like Manhattan will require much more tellers.
- It requires executives to have a good estimate of mean inter-arrival time and mean serving time in order to use this simulation program to find out the optimal number of tellers.


enjoy your summer