Stochastic Optimization of Airport Bus System

Larry Fenn

December 2014

Contents

List of Figures

1 Overview

1.1 Problem Statement

An airport runs a bus service from the terminal to various long-term parking lots nearby. People come and park their cars at the parking lot, get on the bus, then get dropped off at the departures terminal to catch their flight. Returning, passengers take the bus from the arrivals terminal to the lot they have parked at.

The constraint is a quality of service constraint: 95% of passengers in this system should not have to wait more than 10 minutes for a bus; the problem is to find the minimum number of buses that the airport needs to maintain to achieve the quality of service they desire.

1.2 Simplifying Assumptions

The problem as stated has to be simplified through a variety of assumptions. For reference, they are listed here:

- Bus assumptions:
  - Each bus is the exact same with regards to capacity, and speed.
  - Each bus driver acts independently, without a central dispatch organizing them. The only exception is at a checkpoint, located in between departures and arrivals, which enforces a minimum separation in time between buses arriving at the arrivals terminal.
  - Buses may never pass each other, instead waiting one at a time behind each other at stops.
- Passenger assumptions:
  - Passengers arrive as a Poisson process to each station, including the arrivals terminal.
  - Passengers have a static amount of time they require to load onto the bus.
  - Passengers queue in a single file line to board buses.
- Station assumptions:
  - Each station has infinite capacity.
  - Each station only has one place for a bus to load, and one place for a bus to unload.

2 Model

The model we will use is the "ghost simulation" model- its primary difference from the discrete model is that it updates the queues at each station whenever a bus arrives to pick up new passengers. From a mathematical perspective this means we will be modeling the number of people in a queue conditioned on the number of people left in the queue from when the last bus left. The boarding process is modeled using a Markov chain which terminates when either the number of people in line is zero or the bus is full.

2.1 Passenger Model

From the problem description we can concoct the quality of service measure:

- S: number of parking lots + 1 (the arrivals terminal).
- δ: time time for one individual to board a bus.
- C: the capacity of a single, fully empty bus.
- u: the enforced headway between buses at the checkpoint.
- b: the number of buses employed.
- τ_s: the travel time from station s to station $s+1$ (station $s = 0$ is the arrivals terminal).
- w_i: the waiting time for the ith passenger.
- $\mathcal{G}_N(u, b) = \mathbb{E}\left(\frac{1}{N}\sum_{i=1}^{N} \mathbb{I}_{w_i>10}\right)$: the quality of service measure. N is the number of people in this summation.

Our problem statement is thus translated into finding the smallest value b such that the constraint $\lim_{N\to\infty} \mathcal{G}_N(u, b) \leq .1$ holds. The limit is necessary because we need the steady state quality of service measure. A direct mathematical expression for the wait time of person i, w_i, is too difficult to determine. Instead we will implement a computer program instead to simulate passengers and track waiting times. However, due to the time scale and population scale involved in this problem, a direct simulation of passenger usage at an airport is unfeasible; further modeling is necessary.

2.2 Ghost Model

The key observation powering the ghost model is to take notice of passenger arrivals and lump them together in groups. Key events are the arrival and departure of every bus to every stop; the passenger arrivals for the time in between can be treated as a single Poisson process with parameter λ_s for stop s. The justification for the usage of a Poisson process is in line with the assumption that the underlying processes behind the usage of these airport lots is unchanging in time. In other words, we are assuming that for any given day the same demand for people using their cars to take flights. In doing so, the following variables are introduced to capture this behavior:

- λ_s: Poisson parameter for the passengers queued up at station s. We will assume that there is no large-scale immigration or emigration occurring; so $\lambda_0 = \sum_{s=1}^{S} \lambda_s$. Let $\lambda = \lambda_0$ be the "total" in (or out) Poisson parameter.
- $g_n = (I_n, c_n)$: the nth passenger group, representing c_n passengers in the interval I_n where $I_n = (a, b)$ is an interval of time. We will take c_n to be a Poisson variable with parameter λ_s where s is the stop along the bus route they are at.
- π_s: the probability that a passenger from arrivals terminal will get off at stop s. From the same assumption that passenger usage does not change much in time, $\pi_s = \dfrac{\lambda_s}{\lambda}$; in other words, the same proportion of people coming in through a station should be the proportion going out through a station.
- W_i: the number of people in passenger group i who have to wait longer than 10 minutes.

With the wait times now arranged by groups, the quality of service measure can be rewritten "by groups" in time where T is the time passed. Under this interpretation, the total people served by the system are λT arrivals and λT departures; so in T time the mean population served is $2\lambda T$. Letting $N(T)$ represent the number of distinct passenger groups generated by time T, the quality of service condition can be re-expressed over groups using the Poisson distribution's memoryless property.

$$\mathcal{G}(u, b) = \lim_{T \to \infty} \mathbb{E}\left(\frac{1}{2\lambda T} \sum_{i=1}^{N(T)} W_i \right)$$

Our problem statement is thus finding the smallest value of b such that $\mathcal{G}(u, b) \leq .1$, where $\mathcal{G}(u, b)$ is now the quality measure over passenger groups and not individual passengers.

2.3 Boarding Process Markov Chain

For a given station s, the following Markov chain exists describing the boarding process throughout the simulation.

- T_0 = the time from the last bus's departure to the current bus being available to load.
- Q_0 = the existing number of people waiting in line when the current bus is available to load (but before a Poisson process has generated new people to enqueue).
- C_0 = the capacity in the current bus at the start of the process.
- P_n Poisson$(\lambda_s T_n)$: the number of new arrivals to station s after the nth stage time interval T_n time passes.
- $X_n = \min(Q_n + P_n, C_n)$: the number of people boarding at stage n of the process. It can be no larger than the room left in the bus C_0.
- $T_{n+1} = \delta X_n$: the time to board X_n passengers.
- $Q_{n+1} = \max(0, Q_n + P_n - X_n)$: the people enqueued after the X_n people have boarded.
- $C_{n+1} = C_n - X_n$: the capacity of the bus after X_n people board.

Theorem 2.1. *The Markov chain has an absorbing state.*

Proof. The stochastic variable $Q_n \geq 0$. This is because $Q_0 \geq 0$ (it is impossible to have "negative" people in line) and $Q_{n+1} \geq 0$ (since $\max(0, x) \geq 0$). If $Q_n \geq 0$ then either $Q_n > C_n$ or $Q_n \leq C_n$. If $Q_n > C_n$ then $X_n = \min(Q_n, C_n) = C_n$; hence $C_{n+1} = C_n - C_n = 0$ and thus $X_{n+1} = \min(Q_{n+1}, C_{n+1}) = 0$. If $Q_n \leq C_n$ then $X_n = Q_n$ and $C_{n+1} = C_n - Q_n \geq 0$. In either case, $C_n \geq 0$; moreover, C_n is decreasing. By the monotone convergence theorem, as $n \to \infty$ it must be that $C_n \to 0$.

□

We have used the absorbing state of $C_n = 0$ as an absorbing state. In fact, it is easy enough to check that the state $X_n = 0$ is another absorbing state; this can happen either when $C_n = 0$ or when $Q_n = 0$. These two correspond exactly to the situation when the bus is empty, or the queue is empty- no passengers can be loaded in either case, and the bus leaves.

For a given station s, let $Q_s(t)$ be the number of people enqueued at time t. We can build $Q(t)$ precisely as follows:

- $Q_s(0)$ = initial population of the station queue, zero at the start.
- When the next bus arrives, at t_o:
- $Q_s(t_0) = Q_s(0) + P_0$ where P_0 is from the Markov chain definition of Q, above.
- Let $t_{0_i} = t_0 + T_i$, the (global) time of the ith stage of the boarding process.
- $Q_s(t_{0_i}) = Q_i$ from the Markov chain, above.
- When the bus leaves, at time T_K, let $Q_s(t_K) = Q_K$: the Markov chain's description of the queue size when $X_n = 0$.
- When the next bus arrives, at time t_1, repeat.

In essence, we are using the stages of the Markov chain for each separate bus loading procedure, specifically the time intervals T_n, to assemble a global description of $Q_s(t)$. The purpose of doing so is to indicate that the queue size at time t is dependent on the queue sizes before time t: the Markov chain's absorbing state at $C_n = 0$ means that Q_n is potentially nonzero, making the Q_0 for the next bus to arrive dependent on the Q_n for the previous bus.

2.4 Model Stability

A stable instance of this problem is one where every passenger is eventually served by a bus. It turns out there are two considerations here we must consider. The first condition that any particular station has to have its queue of people waiting empty out infinitely often with probability one. Otherwise, there are simply not enough buses to service the entire network and the number of people in line goes to infinity. The second condition for stability is that the total amount of arrivals/departures must be serviceable by the total capacity of the bus fleet. We will employ the Markov chain definition of our problem to answer these questions.

Theorem 2.2. *Given a simulation with the following parameters:*

- b number of buses.
- C capacity of one bus.
- δ loading time per passenger.
- λ_s Poisson rate for parking lot station s.
- π_s probability a passenger gets off at parking lot s.
- K travel time to visit all stations in a loop (without passenger loading).

The queue for each station empties infinitely often with probability one if the following condition on λ_s holds:

$$\lambda_s \leq \min\left(\frac{1}{\delta}, \frac{b \cdot C \cdot \pi_s}{K}\right)$$

Proof. Given a bus loading at a station, first the expected people in queue after one passenger has loaded must be smaller than one; $\lambda_s \delta \leq 1$. Otherwise, in the time it takes one person to load, the mean number of people enqueued has grown- this means that in expectation the size of the queue is infinite. Thus one stability condition is $\lambda_s \leq \frac{1}{\delta}$

The total capacity of the bus system is $b \cdot C$ where b is the number of buses and C is the capacity per bus. A rudimentary bound is determined by measuring the longest time it might take a bus to complete one loop around stations. Since passengers do not queue for other locations other than the parking lots/departures terminal, this amounts to a scenario where the bus at each station is at maximum capacity. Critically it does not matter to us *where* on the bus route the loading and unloading happens: since passengers bound for arrivals always empty out before the departures terminal, and passengers bound for the departures terminal never get off anywhere else. This means that over the course of one loop it spends $C\delta$ time unloading, and $C\delta$ time loading. Thus the total time in the worst case it takes is $K + 2 \cdot C \cdot \delta$, where K is the sum of the travel times between stops. This means that in one loop around the mean number of people that arrive to use the bus system is $\lambda(K + 2 \cdot C \cdot \delta)$. Thus the bus network must accommodate this worst-case traffic scenario: $\lambda(K + 2 \cdot C \cdot \delta) \leq b \cdot C$. In other words, $1 \leq \frac{b \cdot C}{\lambda(K + 2 \cdot C \cdot \delta)}$. Since $2 \cdot C \cdot \delta$ is a positive quantity, we have that $1 \leq \frac{b \cdot C}{\lambda(K + 2 \cdot C \cdot \delta)} \leq \frac{b \cdot C}{\lambda K}$. From $1 \leq \frac{b \cdot C}{\lambda K}$ we have our global stability condition: $\lambda \leq \frac{b \cdot C}{K}$. For a particular station this stability condition is equivalent to $\lambda_s \leq \frac{b \cdot C \cdot \lambda_s}{K\lambda} = \frac{b \cdot C \cdot \pi_s}{K}$ where π_s is the probability of a passenger getting off at station s. Both stability conditions must hold; hence we take the minimum. □

3 Simulation

The simulation implemented in Java. The particular program we wrote stores the simulation parameters in a config file, and exports data into a .csv format where successive values of b are stored in rows, and successive values of u form the columns.

3.1 Implementation Details

3.1.1 PassengerGroup

PassengerGroup objects have three fields- their count, and the time interval that they represent the arrivals for. A PassengerGroup object additionally has a method for removal of some passengers, in the instance that a bus comes and can only take on board a small part of the PassengerGroup due to capacity.

3.1.2 Station

Station objects store their own clock, a queue of PassengerGroup objects, and the departure time of the previous bus. The methods are PassengerGroup creation, queue management, and in particular support removing the first PassengerGroup in queue, decrementing some of its members, and placing the PassengerGroup back in the front of the queue. Additionally, there are several data logging methods for tracking things such as total wait for all people at the station.

3.1.3 Bus

Bus objects track their capacity, the number of passengers on board whose destination is the arrivals terminal, the number of passengers whose destination is the lots, and their own clock. Methods are for loading passengers and unloading them.

3.1.4 Logger

The Logger object exists to track various data points throughout the simulation. In particular, every time a bus begins the boarding process, the Logger object records how long the currently boarding passengers had to wait.

3.1.5 GhostSystem

The GhostSystem object stores the actual logic of the simulation. It has the buses in a list, the stations in a list, and the various simulation parameters read from file (with support for the user to modify the parameters). The next few paragraphs are about implementation details of the simulation only.

The simulation first takes a tolerance and cycle limit from the user. These parameters determine how long the simulation is run to determine a particular value of $\mathcal{G}(u, b)$, the quality of service measure.

For each bus, starting from the first bus ending at the bth bus, we visit each station in turn. First, pick up passengers at the Arrivals terminal by a Poisson process. At each station:

- Increment the bus internal clock by travel time to the station.
- Unload passengers using a binomial process to determine how many and increment the bus internal clock by the time it takes to unload them.
- Bus boarding process:
  - Compare the bus internal clock with the station clock; generate using a Poisson process a PassengerGroup for that interval of time. Enqueue this PassengerGroup at the end of the queue. The station's clock is now updated to the bus's clock time.
  - If there are no PassengerGroups in queue, or if the bus is full, leave the station.
  - Else, board the front most PassengerGroup in the queue. If the bus cannot fit the entire group, then load only a portion of the front most PassengerGroup.
  - Increment bus clock by the time it takes to load the passengers from the PassengerGroup. Repeat the bus boarding process.

At the end of the line, unload all the passengers at the departures terminal, and increment the bus clock accordingly. Then wait at the checkpoint- if the last bus through the checkpoint passed through less than u time units ago, delay (by incrementing the bus clock) until the last bus passed through the checkpoint exactly u units ago.

Once a bus has looped back to the checkpoint, increment the bus counter. Once we have iterated over all buses, we repeat the entire simulation back at bus 1, until either the cycle count has been reached or the tolerance threshold has been achieved in estimates of $\mathcal{G}(u, b)$.

3.2 Variance of $\frac{W_i}{P_i}$

Since the simulation is tracking PassengerGroups and tallying up the proportion that satisfies the quality of service constraint, a natural question is the relationship between the cycles of the simulation and the error in the simulated value of the quality of service. The approach we have taken is to track each $\frac{W_i}{P_i}$, the quality measure for passenger group i, and measure the variance in our observed (simulated) data after each full "cycle" of the simulation. Provided the simulation is stable, this will allow us to determine a stopping point for the algorithm. The rationale behind this application is that a stable simulation has, by definition, queues that empty infinitely often with probability 1. Thus, in a stable simulation, every station eventually returns to the state it was at originally- and moreover, that the number of people in queue for a particular station can be thought of as a Markov process with an ergodic state of zero people in queue. From there we invoke the Central

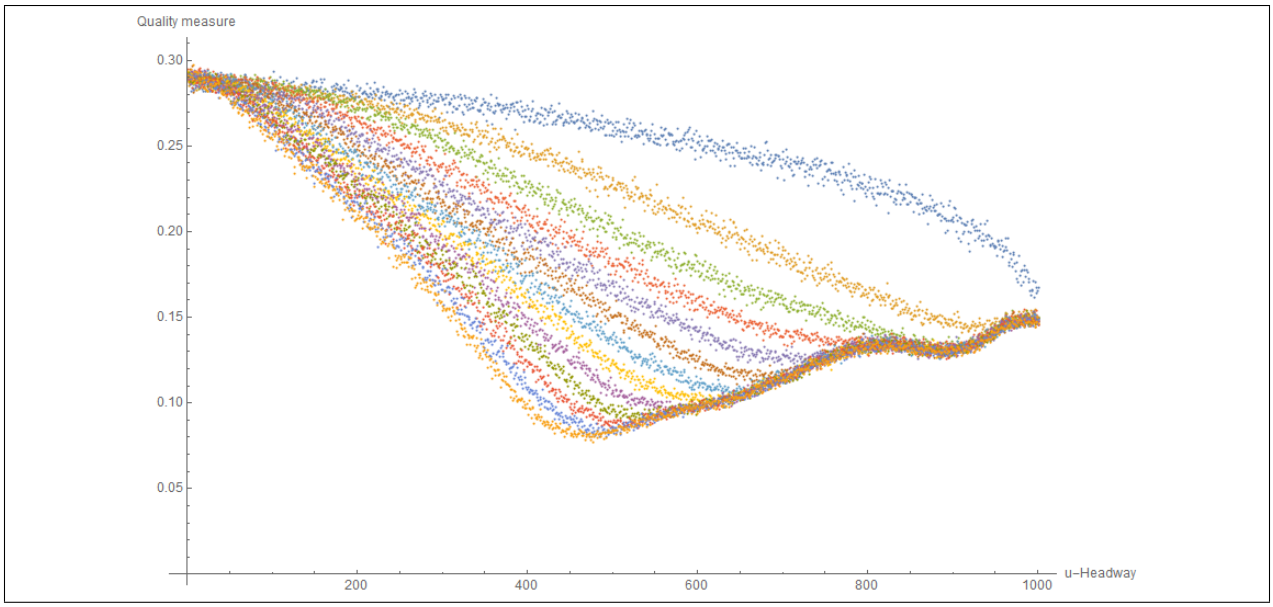

Figure 1: Noisy estimates $\mathcal{G}(u,b)$ vs u for various fixed values of b.

Limit Theorem for weakly dependent variables and establish a confidence interval.

Conjecture: Let $\frac{W_i}{P_i}$ be the quality measure for the ith passenger group; suppose that after k full cycles through all buses and stations there are n_k passenger groups total. The simulated statistics $\bar{\mathcal{Q}}_k = \mathbb{E}[\frac{W_i}{P_i}]$ for $i \in [n_k, n_{k+1}]$ are weakly mixed.

The simulation implements this technique by computing the sample statistics for "quality of service measure of the kth cycle" $S_k = \mathbb{E}\left(\sum_{i=1}^{n_k} \frac{W_i}{P_i}\right)$ where n_k is the number of passenger groups in the kth cycle. The "actual" measure for quality of service is $\lim_{N\to\infty} \mathbb{E}\left(\sum_{i=1}^{N} \frac{W_i}{P_i}\right) = \lim_{k\to\infty} \mathbb{E}\left(\sum_{i=1}^{n_k} \frac{W_i}{P_i}\right)$ by a regrouping of the infinite sum. The sample variances and means of S_k are computed, and then using a Central Limit assumption a confidence interval for simulation of $\mathcal{G}(u,b)$ is halted if the width of the confidence interval for $\lim_{k\to\infty} \mathbb{E}\left(\sum_{i=1}^{n_k} \frac{W_i}{P_i}\right) = \lim_{N\to\infty} \mathbb{E}\left(\sum_{i=1}^{N} \frac{W_i}{P_i}\right)$ is within the requested tolerance.

Unfortunately we have no proof of the weak-mixing property. The simulation as-coded currently issues a warning if the tolerance goal is not met, but without a proof the current simulation code should only be considered as an ad-hoc improvement of simulation efficiency.

4 Optimization

4.1 Approaches

For reference an entire data set was generated for some predefined simulation parameters to define $\mathcal{G}(u, b)$ for $u \in [0, 10], b \in [9, 20]$. The optimization algorithms run alongside the simulation and do not rely on any pre-existing data, and the data set is presented here to give context only to the problem. The units of u are tenths of a minute; the units of b are size of fleet -7 (since the problem is unstable for $b \leq 7$).

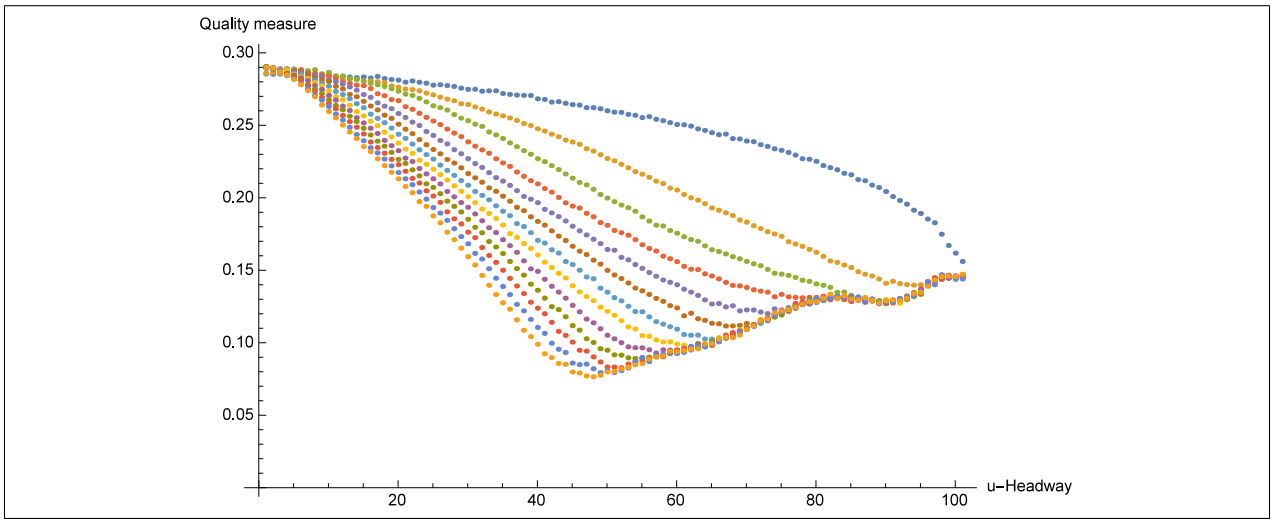

Figure 2: $\mathcal{G}(u, b)$ vs. u for various fixed values of b.

The two parameters are b and u, the number of buses and the control headway between buses at the checkpoint. The optimization problem is of finding the smallest b such that $\mathcal{G}(u, b) \leq .1$. The optimization schemes first reduced to 1-D optimization by computing values for fixed b. The key observation here is that for a fixed headway parameter u the value of $\mathcal{G}(u, b)$ is monotonically decreasing as b increases: $\mathcal{G}(u, b_1) < \mathcal{G}(u, b_0)$ for all $b_1 > b_0$. This is because adding a bus to the system can only improve the quality of service since it increases the overall capacity of the network. Thus we can start with our b parameter sufficiently large; let $b = b_0$, and find the minimum of $\mathcal{G}(u, b_0)$ at u_0 by whatever technique (u is a continuous variable, "headway time"). Depending on $\mathcal{G}(u_0, b_0)$, the minimum for a fixed b, we can either decrease b (if $\mathcal{G}(u_0, b_0) \leq .1$) since there may be some $b_1 < b_0$ such that there is some u_1 where $\mathcal{G}(u_1, b_1) \leq .1$. Alternatively, if the minimum of $\mathcal{G}$ is still larger than .1, we know that there can be no improvement for any smaller values of b; hence, we must have previously detected the smallest b such that our quality condition is satisfied.

The major difficulty in applying any optimization techniques is the observation of the values of $\mathcal{G}(u, b)$ are the results of a simulation. This means that, dependent on how much computational time we devote to the simulation, our observed values for $\mathcal{G}(u, b)$ are noisy observations from the "true" value.

For a fixed value of b, we first employed target tracking techniques with fixed or dynamic step sizes. The target tracking technique is the most straightforward technique for finding solution from noisy observations. Next, a forward difference FDM approximation is employed for Newton's method on each function $\mathcal{G}(u, b)$ for fixed b. Finally, an IPA estimator for the derivative was created and used instead of an FDM estimate for Newton's method.

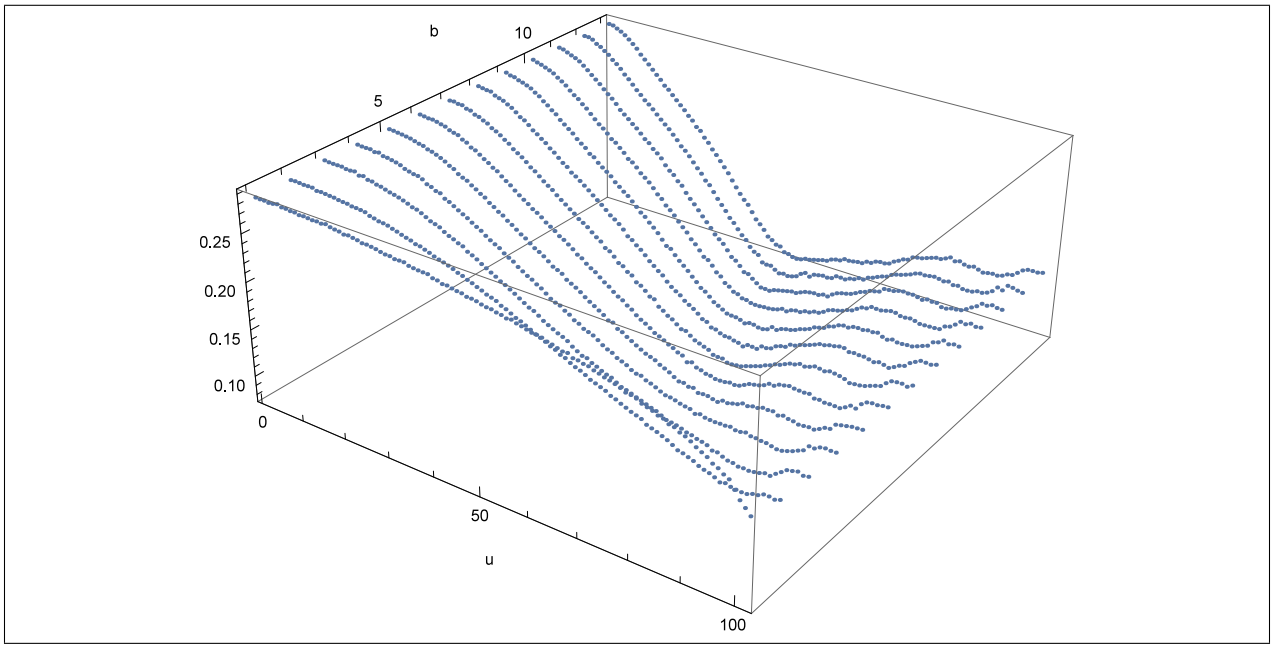

Figure 3: $\mathcal{G}(u, b)$ vs. u and b.

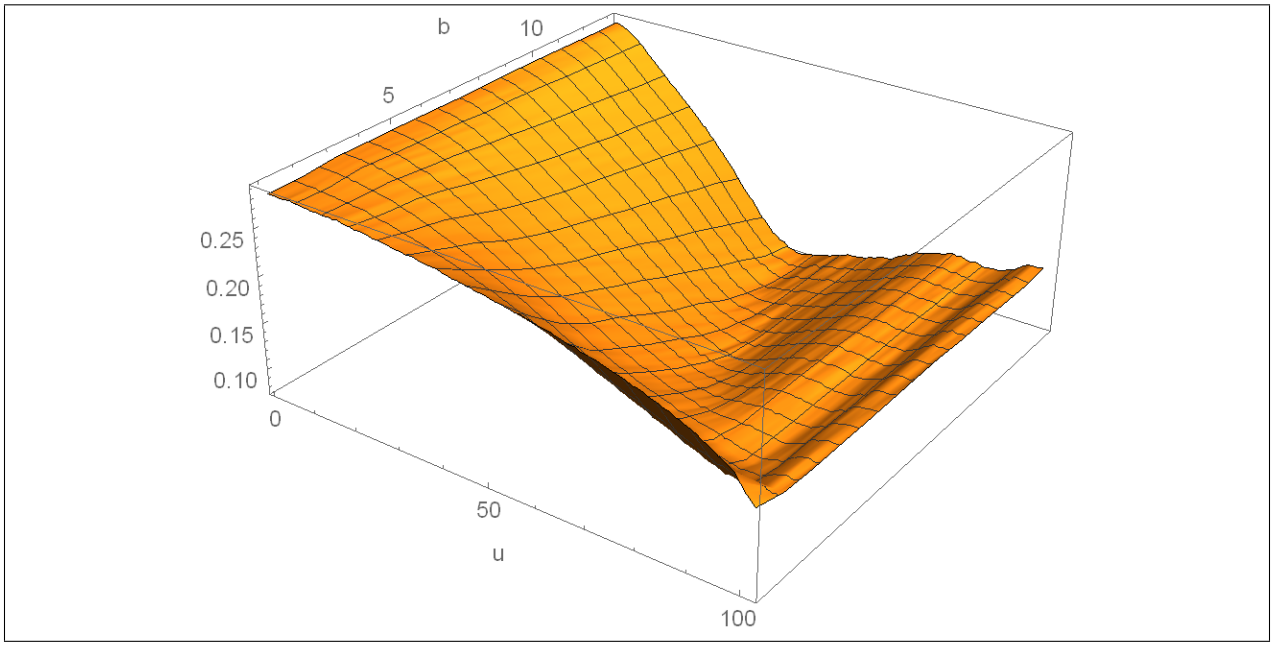

Figure 4: $\mathcal{G}(u, b)$ vs. u and b interpolated as a surface.

4.2 IPA Estimation

Our simulation provides us with a (noisy) way to observe $\mathcal{G}_N(u,b) = \frac{1}{2\lambda N}\mathbb{E}\left(\sum_{i=1}^{N} W_i^*\right)$ where N is the number of passenger groups simulated and W_i^* is the number of people in the ith passenger group who have to wait longer than 10 minutes. First, we use the ghost model assumption: for each passenger group W_i, we will treat the distribution of their arrivals as a uniform distribution over I_i, the interval of time the passenger group is covering. Let t_i be the time that passenger group is loaded onto a bus; the measure of W_i^* given I_i is called $\widehat{W}_i^*(t_i)$:

$$\widehat{W}_i^*(t_i) = \mathbb{E}(W_i^*|I_i)(t_i) = \begin{cases} P_i\left(\dfrac{t_i - 10 - t_1}{t_2 - t_1}\right) & \text{if } t_1 < t_i - 10 < t_2 \\ P_i & \text{if } t_2 < t_i - 10 \end{cases}$$

By assuming the passengers are uniformly spread through the interval, the number of passengers who have to wait longer than 10 minutes given a loading start time of t_i is a simple linear function in t_i.

From here, we can connect $\mathcal{G}_N(u,b)$ (the quality measure over groups) to $\widehat{W}_i^*$, keeping in mind that the passenger group pick up time t_i is itself a random variable:

$$\begin{aligned} \mathcal{G}_N(u,b) &= \frac{1}{2\lambda N}\mathbb{E}\left(\sum_{i=1}^{N} W_i^*\right) \\ &= \frac{1}{2\lambda N}\sum_{i=1}^{N}\mathbb{E}(W_i^*) \\ &= \frac{1}{2\lambda N}\sum_{i=1}^{N}\mathbb{E}(\mathbb{E}(W_i^*|I_i)) \text{ (law of total probability)} \\ &= \frac{1}{2\lambda N}\sum_{i=1}^{N}\mathbb{E}(\widehat{W}_i^*(t_i)) \end{aligned}$$

Now, we need to establish the IPA result:

Theorem 4.1. $\dfrac{\partial}{\partial u}\mathcal{G}_N(u,b) = \dfrac{\partial}{\partial u}\dfrac{1}{2\lambda N}\sum_{i=1}^{N}\mathbb{E}(\widehat{W}_i^*(t_i)) = \dfrac{1}{2\lambda N}\sum_{i=1}^{N}\mathbb{E}\left(\dfrac{\partial}{\partial u}t_i\dfrac{P_i\cdot\mathbb{I}_{t_1<t_i-10<t_2}}{t_2-t_1}\right)$

Proof. Since the sum is a finite sum, what needs to be shown is that $\dfrac{\partial}{\partial u}\mathbb{E}(\widehat{W}_i^*(t_i)) = \mathbb{E}\left(\dfrac{\partial}{\partial u}\widehat{W}_i^*(t_i)\right)$. The approach we take will be to employ Theorem 27 from the textbook. Using the definition of $\widehat{W}_i^*(t_i)$ above we observe that the bus loading start time t_i is the random variable here; the function $\widehat{W}_i^*(t_i)$ was defined above as a linear function in t_i. Since

$t_1 \neq t_2$, $\widehat{W_i^*}(t_i)$ definitely satisfies the requirement of Lipschitz continuity with integrable Lipschitz modulus; the function is moreover differentiable over t_i with derivative equal to either $\dfrac{P_i}{t_2 - t_1}$ or 0 (which we will encode using an indicator variable). Finally, the sample path derivatives for t_i always exist- changing the headway will either produce zero change in the pick up time (if the bus that picked up the passenger group never had to be stopped at the checkpoint or was never delayed *because* of a bus that was stopped at the checkpoint), or a finite change (the headway induces some non-finite delay). Thus the sample path derivative always exists, and we call it $\dfrac{\partial}{\partial u} t_i$. □

To implement the IPA, the sample path derivative needed to be correctly tracked throughout the course of simulation. This was accomplished by giving each bus a flag that was initially set at the checkpoint as `true` if the bus was held at the checkpoint and `false` otherwise. If a bus had the checkpoint flag, it meant that (from a mathematical perspective) changing u would definitely change the arrival times for the bus; hence the sample path derivative here would be nonzero. Additionally, even if the original checkpoint flag was set to `false`, if the bus ever had to wait for the bus ahead of it to unload it would check the flag of the bus ahead of it: in the case that the bus ahead of had the flag set to `true`, then the waiting bus would flip its flag from `false` to `true`. Again, this is to capture the notion of the sample path derivative: changing the headway would cause this bus to be delayed, even if it was not held back at the checkpoint- because the bus must wait sometime in its cycle for a bus that was impacted by the headway.

4.3 Optimization Results

The following parameters were used for the simulation:

- Cycle timeout: 10000
- Bus capacity: 60
- Passenger load time: 10 seconds
- Quality of service: 95% of people must wait no longer than 10 minutes.
- 5 parking lots, with Poisson arrival probabilities of 1.
- Travel time between lots: 10 minutes.
- Travel time from arrivals to first lot: 2 minutes.
- Travel time from last lot to departures: 2 minutes.

The algorithms were all started with initial guesses for u at 10 and b at 20.

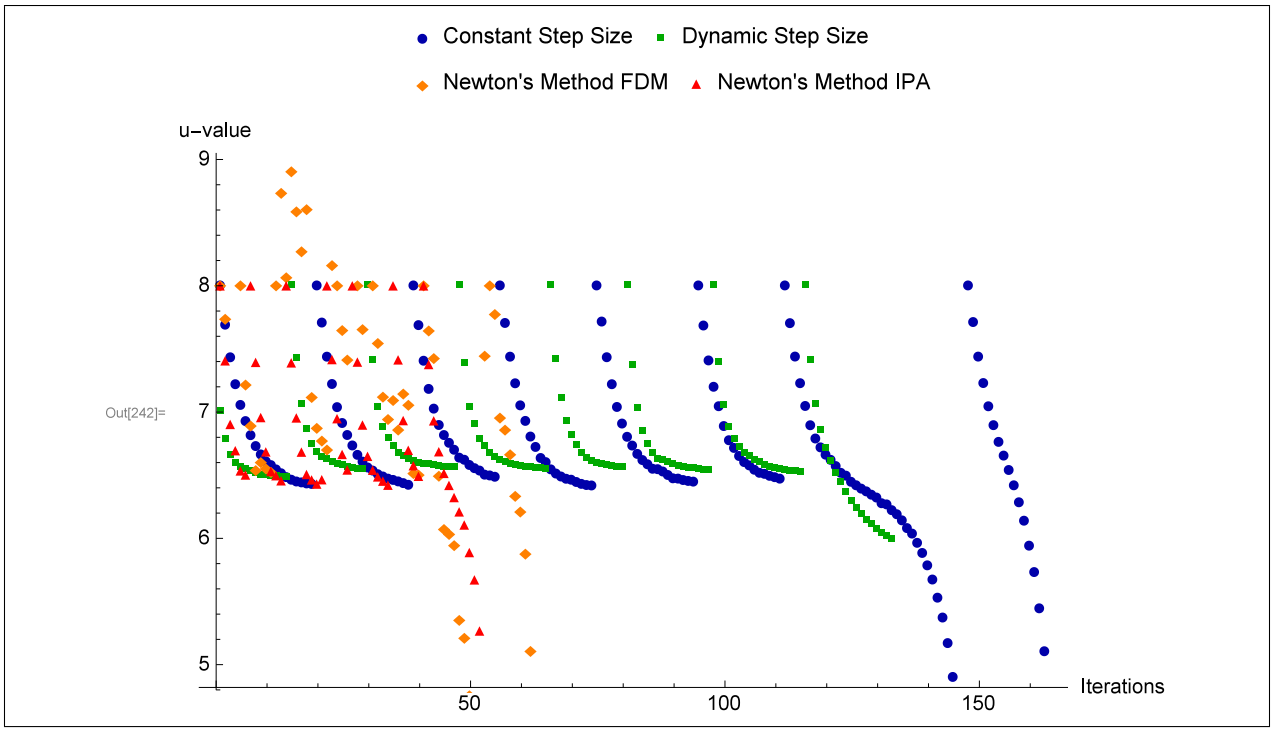

Figure 5: Optimization algorithms, iterations vs. u values.

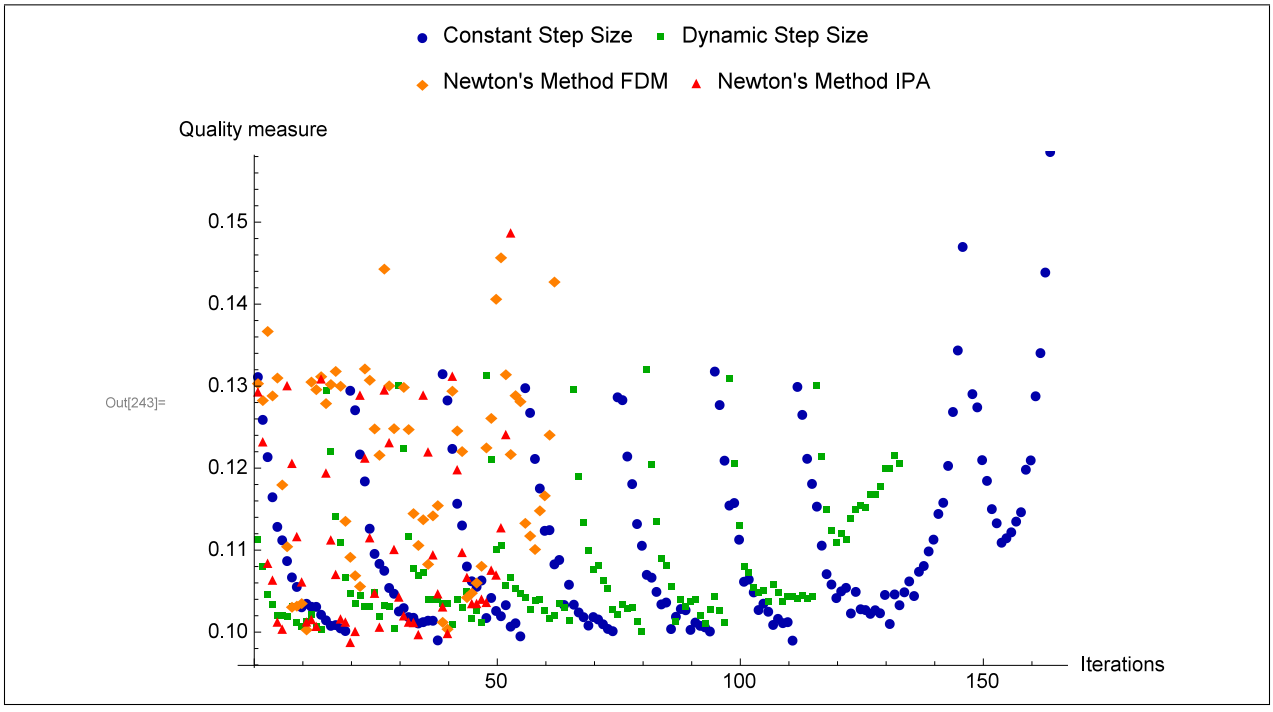

Figure 6: Optimization algorithms, iterations vs. $\mathcal{G}(u, b)$ values.

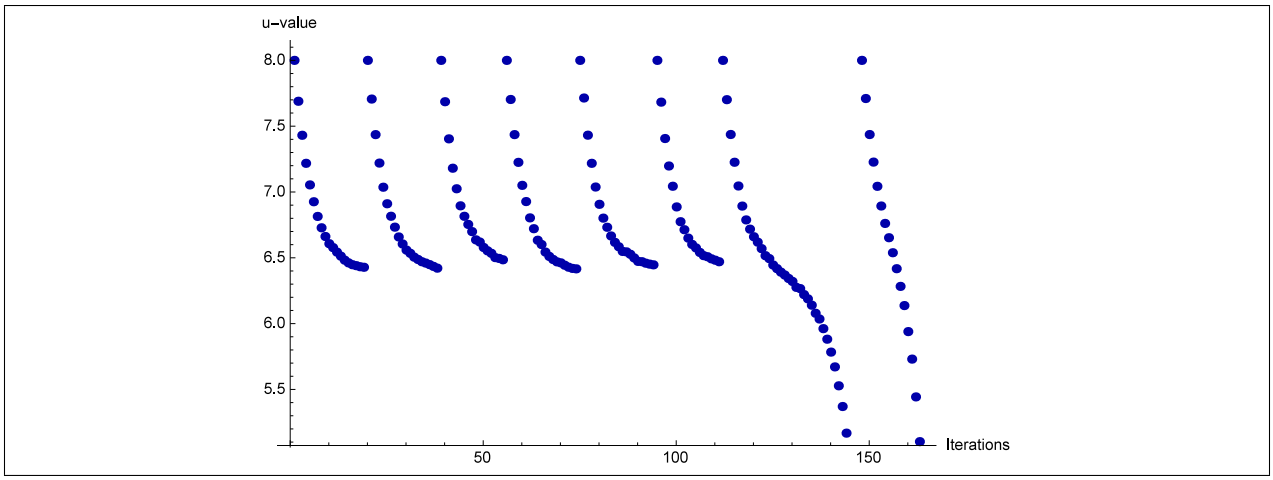

Figure 7: u values for the constant step size target tracking algorithm.

4.3.1 Constant Step Size Target Tracking

The constant step size target tracking after 164 iterations yielded a solution of $(u, b) = (6.470, 15)$; 15 buses with a headway of 6.470 minutes. One can see the pattern of the target tracking algorithm: for the first 6 values of b it tries (i.e. up to the 15th bus), the algorithm manages to find a u such that $\mathcal{G}(u, b)$ is below .1. At $b = 14$ it finally fails to satisfy the constraint, and from there the algorithm stops by taking advantage of $\mathcal{G}(u, b)$ increasing monotonically for a fixed b: if for all u that $\mathcal{G}(u, b) > .1$ then $\mathcal{G}(u, b-1) > \mathcal{G}(u, b)$ means that for all u: $\mathcal{G}(u, b-1) > \mathcal{G}(u, b) > .1$. Hence there are no other possible solutions.

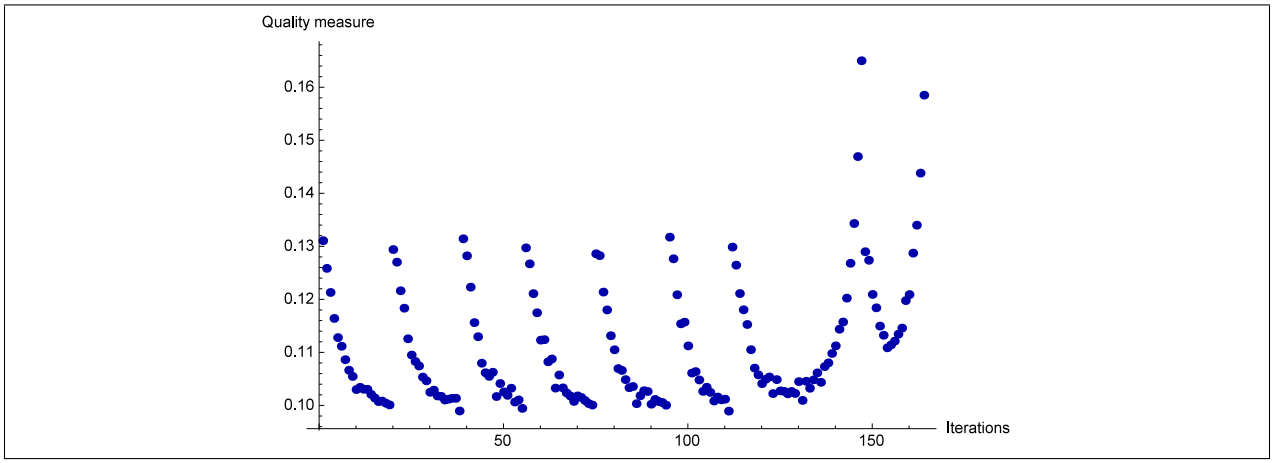

Figure 8: $\mathcal{G}(u, b)$ for the constant step size target tracking algorithm.

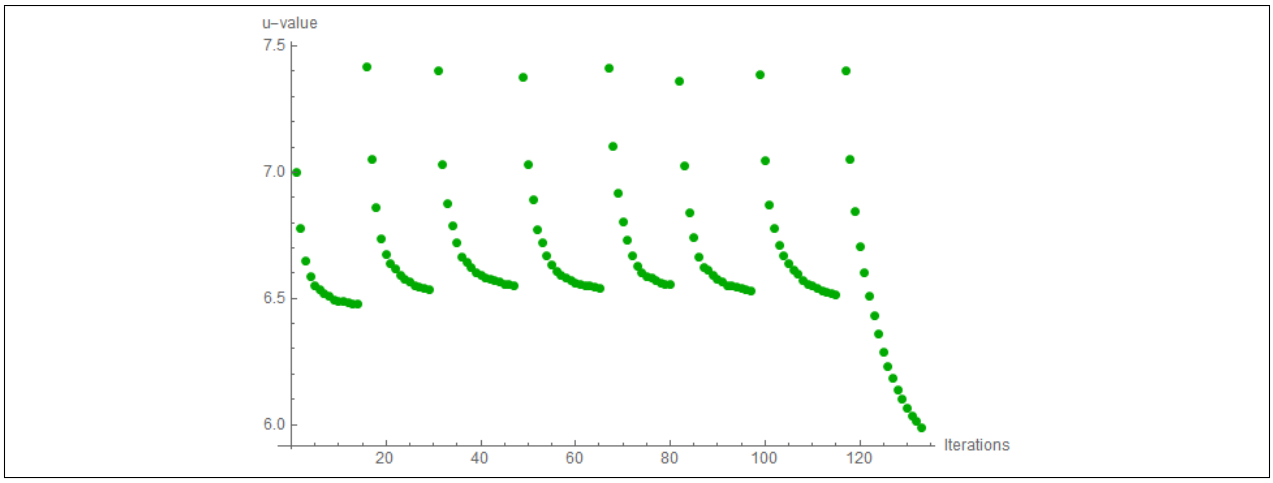

Figure 9: u values for the dynamic step size target tracking algorithm.

4.3.2 Decreasing Step Size Target Tracking

The dynamic step size target tracking after 133 iterations yielded a solution of $(u, b) = (6.532)$; 15 buses with a headway of 6.532 minutes. The discrepancy here comes from the noise from simulation. The dynamic step size target tracking algorithm required fewer iterations, but did require more programming logic to implement. The change in step size has to be carefully tuned to both the precision of the simulation and the underlying parameters of the simulation; before settling on an ϵ_n with $\epsilon_{n+1} = \frac{5}{6}\epsilon_n$ there were several applications of the algorithm that failed to find minimums due to the step size shrinking too rapidly. The complexity is that the different level curves for fixed b values $\mathcal{G}(u, b)$ have very different shapes.

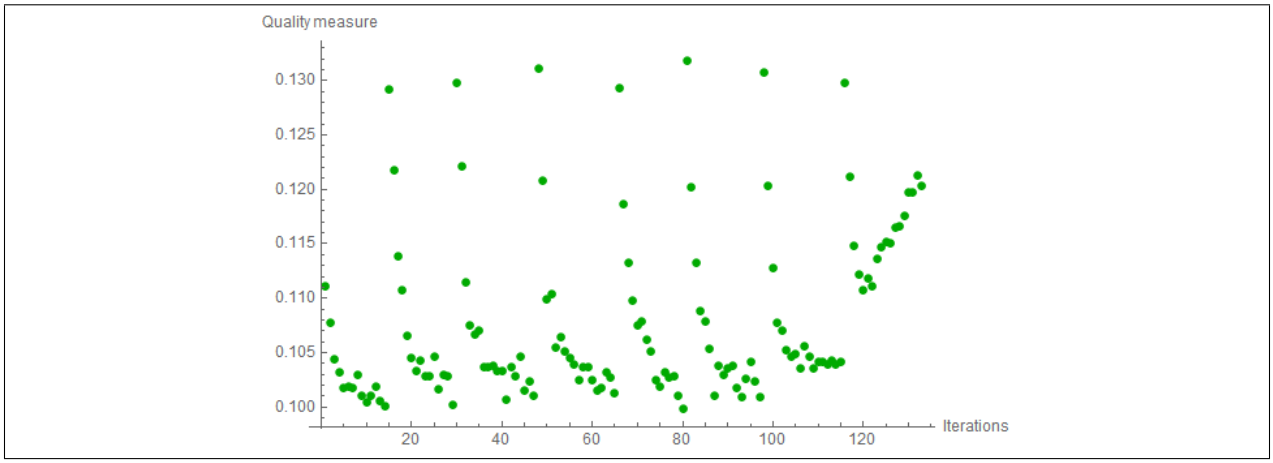

Figure 10: $\mathcal{G}(u, b)$ values for the dynamic step size target tracking algorithm.

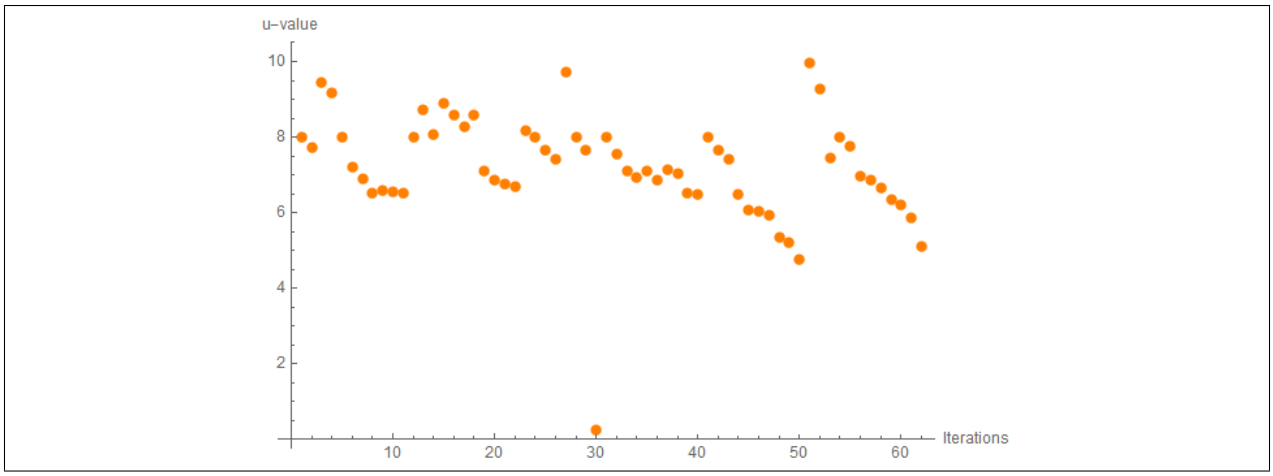

Figure 11: u values for the FDM Newton's method algorithm.

4.3.3 FDM Newton's Method

Using FDM for Newton's method on each level curve of b yielded a solution after 63 iterations of $(u, b) = (6.501, 15)$; 15 buses with a headway of 6.501 minutes. Notably, the pattern in the various iterations looks vastly different than the target tracking algorithm. The primary benefit of Newton's method is much faster solving for the solution. The weaknesses are that the algorithm requires much more precise computation, both in terms of the noise coming from the simulation but also in computing a reliable figure for the FDM approximation to the derivative. This requires fewer "iterations" of the Newton's method algorithm, but far more computation time is spent driving the simulation to compute a derivative. The other weakness is that the method can fail to ever converge in the instance that the algorithm is checking points far away from the stable regions of the simulation. For example, if u is sufficiently large, then $\mathcal{G}(u, b)$ is going to be close to 1 in a neighborhood of that u. Newton's method explicitly fails when the first derivative is zero, so in regions where the simulation is unstable this algorithm will also be unstable.

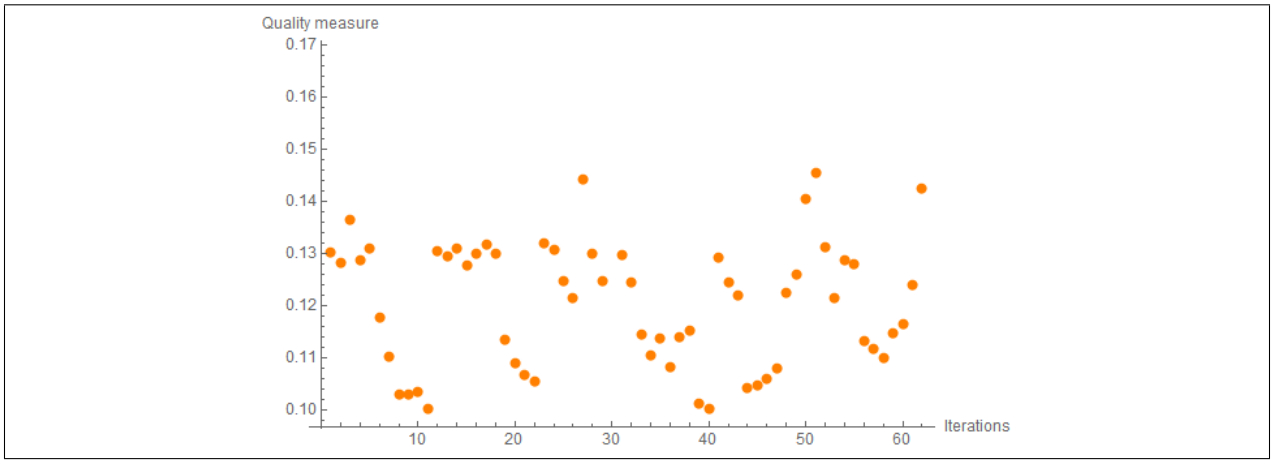

Figure 12: $\mathcal{G}(u, b)$ values for the FDM Newton's method algorithm.

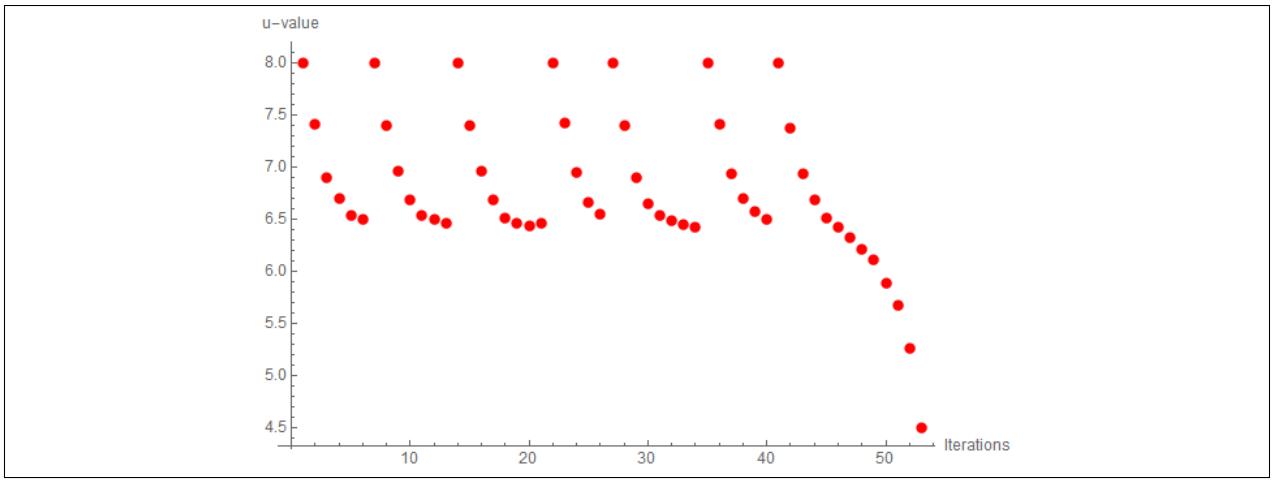

Figure 13: u values for the IPA Newton's method algorithm.

4.3.4 IPA Newton's Method

Using IPA for Newton's method on each level curve of b yielded a solution after 52 iterations of $(u, b) = (6.492, 15)$; 15 buses with a headway of 6.492 minutes. On the implementation side, the IPA estimators required far less "tuning" than the FDM for Newton's method did- where FDM created error from the step size used in the forward difference, the IPA estimator has no such construct. The use of IPA here makes the iterations much more clearly organized, as the optimization no longer suffers from this additional error and instead only has to deal with simulation error. As a result, the overall process stops in fewer iterations. In summary, IPA provides much more robust optimization and much more predictable algorithm behavior.

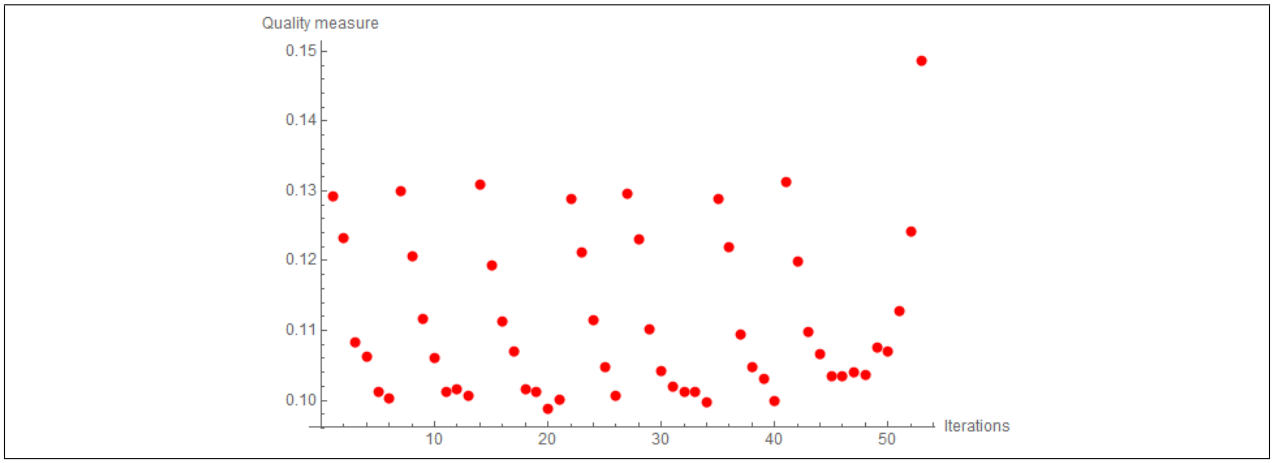

Figure 14: $\mathcal{G}(u, b)$ values for the IPA Newton's method algorithm.

5 Conclusion

The model of employing grouped up passenger arrivals to speed up the time scale of simulation has enabled a speedy computer simulation to be implemented. What would have been a problem involving potentially simulating weeks of duration is instead reframed as a simulation of events and arrival processes over the course of simulated weeks.

Two types of optimization algorithms were employed- one based on target tracking, and another based on root finding using Newton's method. On the basis of speed the Newton's method optimizations performed better than the target tracking algorithms; in particular, it was found that the Newton's method algorithm based on IPA estimation of the derivative was the best performing.

The simulation as coded has some hard limits for what it can and cannot do. However, even with the simplifications made in order to create the computer simulation, there are potential extensions:

- Passengers can have multiple destinations, instead of just the terminal and the individual lots. To implement this change, the stability analysis will need to take into effect the much slower travel as more passengers load and unload from the buses. Additionally, the program itself must be modified slightly, to track these different types of services.
- The bus fleet need not be homogeneous. Stability analysis will need to take into account the diversified fleet, but from a programming perspective it is natural enough to create extensions of the existing Bus class object- a slow moving, high capacity bus and a quick, lower capacity bus seem like natural options.
- Travel time between stations can be randomized. This could be to simulate having to use surface streets. Implementation would be to make the travel time dependent on some random process. Stability analysis will have to take into account the change in model, and will have to work over the expected value of travel times.
- Different types of passengers can be created rather than a single class of population. This could entail having different load time parameters, or even more extreme having some passengers capable of "skipping" the queue. Their arrivals will be modeled with a Poisson process- the stability analysis will not change since it is a precondition of stability that all passengers are accommodated, but the program would have to be modified to implement some form of priority queue.
- Population parameters can be made to change over time. This would allow the model to take into account holiday season travels or economic recession influencing airport usage. This would impact stability analysis as each Poisson parameter now changes in time; the computer simulation will have to be modified similarly.