CS1538 Term Project:
The Elevator Pitch

29 April 2017

Alec Rosenbaum, Terry Tan, Lauren Thomson, Cory Trbojevic

Overview

The primary goal of this simulation was to determine the optimal algorithm for reducing user user wait times for the elevators in Benedum Hall while classes are in session. It was also be used to determine which algorithms allow the elevators to meet the standards set by University of Pittsburgh Facilities Management. Because elevator scheduling is an NP-Hard problem, it cannot be solved analytically in polynomial time and, thus, a simulation is required.

The algorithms under test included Scan, Look, Nearest Car First, Fixed Sectoring Common Sector, and Fixed Sectoring Priority Timed. Scan and Look are "simple" algorithms, in that they do not take into consideration the activities of other elevators because there is no common controller that has the necessary information. Scan elevators move up and down continuously while picking up passengers if there is a call on the floor on which it is arriving, and dropping off passengers if anyone wants to get off on a floor on which it is arriving. Look follows the same idea as Scan, except that it only goes as high as the highest call or passenger destination, and as low as the lowest call or passenger destination. By doing this, it avoids needlessly going further than it has to, but potentially turns around right before a person was about to call an elevator.

Nearest Car First, Fixed Sectoring Common Sector, and Fixed Sectoring Priority Timed are all more complex algorithms in which the elevators are all controlled by a common controller which has knowledge of the activities of each elevator. For Nearest Car First, as a call is received, a calculation is performed for each elevator based on the location of the call and the direction of the elevator. The elevator that ends up with the highest score, known as the Figure of Suitability, will respond to the call. In Fixed Sectoring Common Sector, the elevators will preferably respond to calls only within their predefined sector, unless there is an arrival near an elevators sector and that elevator is close to the arrival. This "preference" is set based on weights derived from similar calculations to the Nearest Car First algorithm. Fixed Sectoring Priority Timed is similar, with the added constraint that if someone has been waiting for too long, the weight for other elevators to pick them up even outside of their sectors is increased.

Questions

The first question we attempted to answer through this simulation is which of the algorithms that we tested gives the best performance. We measured performance by using the average time ω between when an elevator is called and when the elevator arrives at the caller's destination. This is the primary metric that we compared to the

Facilities Management elevator standards. We needed to answer this question first since our other questions build off of its answer.

Using the algorithm decided upon in this first question, we then investigated the questions of what times and days have the lowest traffic and the correlation between wait times and origin and wait times and destination floor. These questions informed our final question regarding under what circumstances it would be faster to travel via the stairs instead of taking the elevator.

Expectations

We expected that Look and Scan would perform poorly compared to the more complex algorithms. These algorithms function with far less required knowledge about the domain, but that also means that they miss out on optimizations that would make use of this data. Since neither algorithm takes into account the actions of the other elevators, all of the elevators could all end up bunched up while passengers wait on a distant floor. They give equal consideration to every floor instead of giving priority to floors with a larger amount traffic, such as floors where many classes are held or where people arrive (ground floor and first floor). Furthermore, there should be longer wait times with Scan compared to Look, since it will travel to the top floor and bottom floor regardless of passenger requests and waiting calls.

Nearest Car First seems like it should minimize wait times, overall. This is because, like the Fixed Sector algorithms, there is a controller that knows what every elevator is doing and, unlike the Fixed Sector algorithms, every elevator is more likely to answer any call at any time. We believe that this will be faster than Fixed Sectoring in Benedum hall, because some floors are much more utilized than others at different points in time. With Fixed Sectoring, there would be less of a chance of an elevator answering a call on one of these floors if it is outside its sector.

The performance of the Fixed Sector algorithms will depend on if the division of sectors chosen is intelligent and well suited to the traffic pattern in the building. If the sectors are chosen well, they could outperform Nearest Car First by ensuring that areas far from the main concentration of traffic do not wait for an extremely long period of time for an elevator. Fixed Sectoring Priority Timed, especially, should work to reduce long wait time outliers on floors far away from the main hub.

Since we are only modeling student traffic in this design, the obvious time for low elevator traffic is the gaps where there are no classes starting or ending. The majority of classes begin on the hour or half hour, while most classes end on the hour, at fifteen minutes past or ten minutes before the hour. Any time more than five minutes away from these intervals is a potential candidate. It’s harder to estimate the impact that

elevator schedule will have on traffic and how that traffic will vary over the course of a day or week.

Modeling the System

We began by obtaining data about the classes in Benedum, including the floors, the time, and the number of people enrolled. We then collected data about which floor people first access the elevator from (ground or first) by counting the number of people that arrived on certain floors during certain times of the day (using a program which created a .csv file from the data). With this arrival data, we determined the probability that a person will arrive on the ground floor versus the first floor. We modelled this with a normal distribution to generate arrival floors. We then used our data about classes to generate when people will arrive at the elevator before and after class following a chi square distribution. We structured our program as a state machine where state changes are triggered by popping events off of a Future Events Queue. Events in our system were passenger arrivals and elevator arrivals. When a person arrives, we push them into a queue on their origin floor. When an elevator arrives, we load it with passengers, and then determine where it will go next using the various elevator algorithms.

Experimental Design

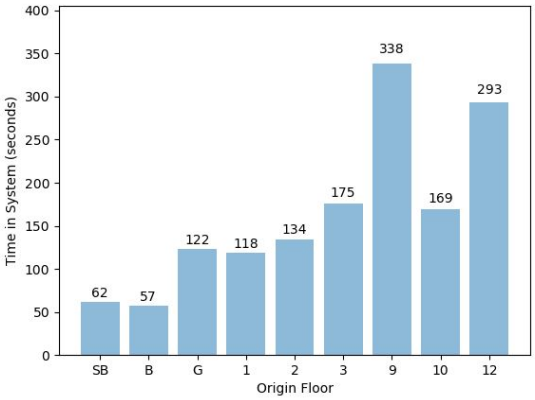
To run our experiments, we began by creating a building object with parameters that match Benedum Hall. For every day of the week, we generated arrivals following our data model as described in the section “Modeling the System.” We then created the necessary elevators and initialized the Future Events Queue with the arrivals and began the simulation. As events were popped off the queue, we recorded data about passenger state changes and the time that the changes occurred. This data was then used to determine the average time that the passenger spent in the system based on their origin floor, the average time that the passenger spent in the system based on their arrival time, the average time that the passenger spent in the system based on their destination, and the average passenger wait time based on their arrival time.

Experimental Results

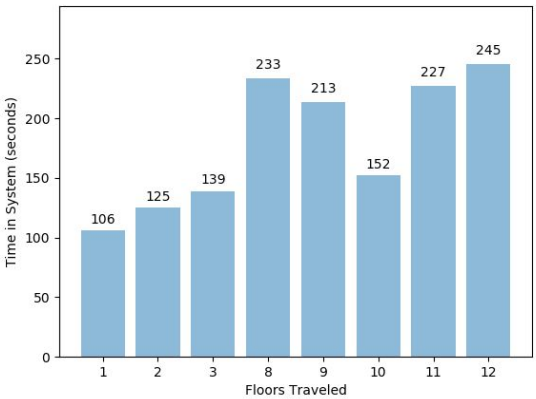
Scan

Utilizing the scan algorithm yielded an average wait time of 78.91 seconds (standard deviation of 136.37 seconds), and an average time in system of 163.92 seconds. Graphs of the data are included below:

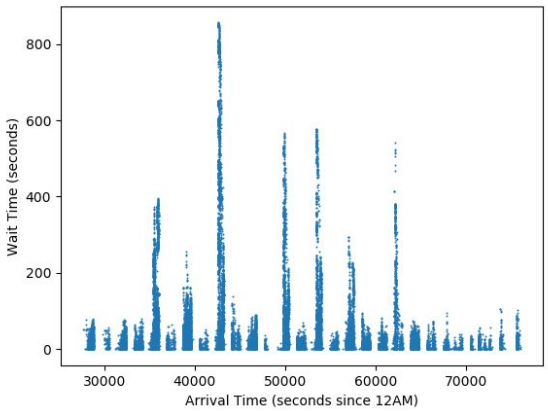
Time in System vs Origin



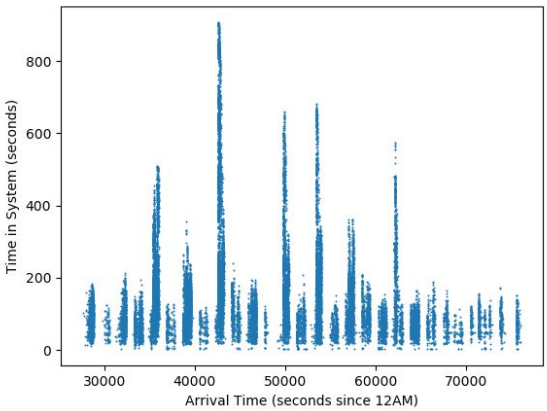
Time in System vs Travel Distance



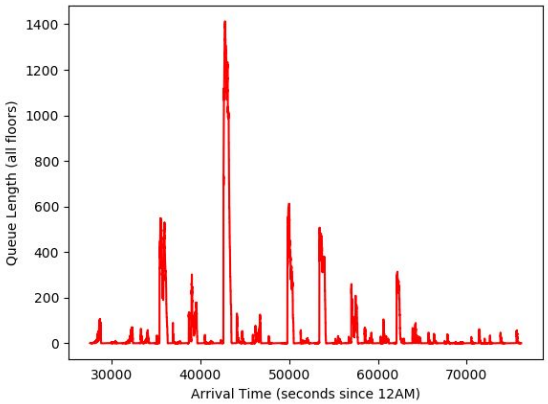
Wait Time vs Time of Day



Time in System vs Time of Day



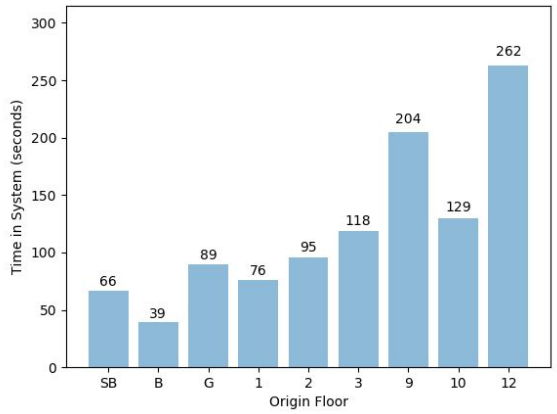
Queue Length vs Time



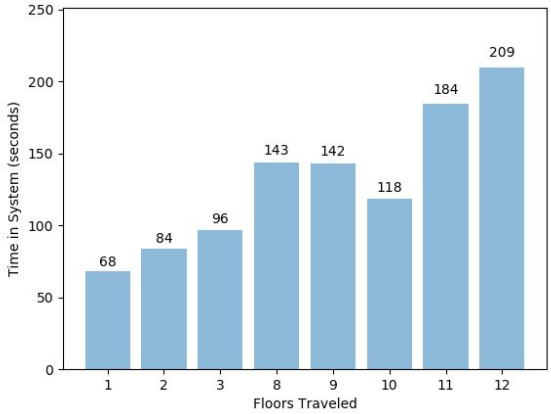
Look

Utilizing the look algorithm yielded an average wait time of 52.67 seconds (standard deviation of 104.19 seconds), and an average time in system of 114.83 seconds. Graphs of the data are included below:

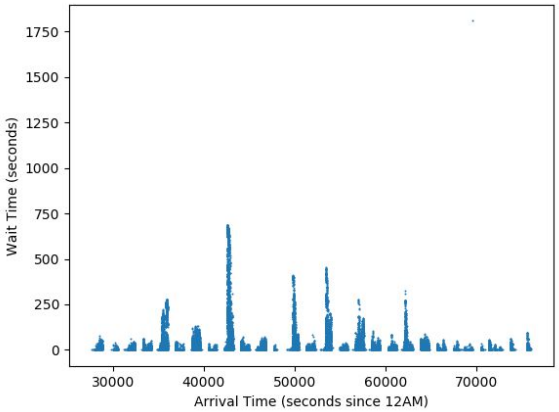
Time in System vs Origin



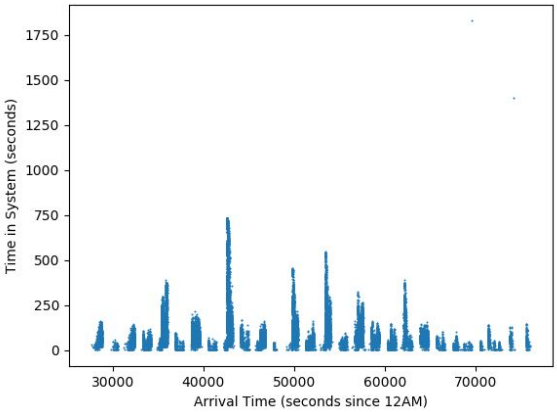
Time in System vs Travel Distance



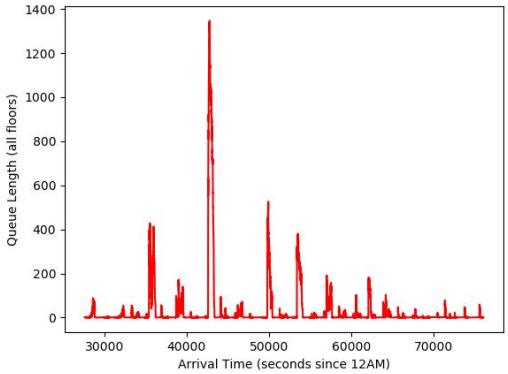
Wait Time vs Time of Day



Time in System vs Time of Day



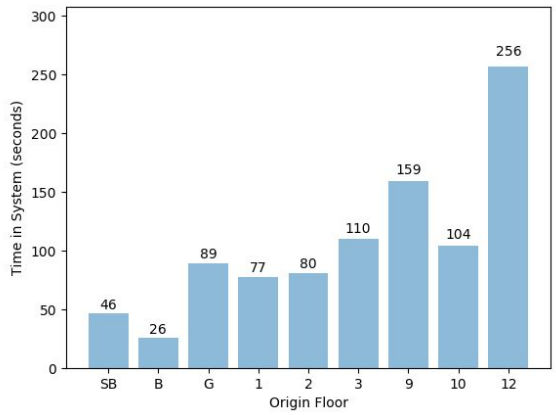
Queue Length vs Time



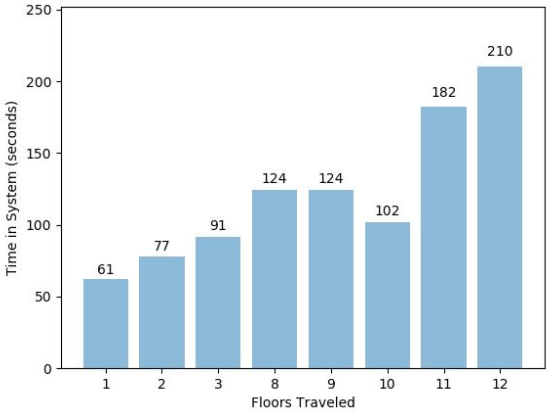
Nearest

Utilizing the look algorithm yielded an average wait time of 49.51 seconds (standard deviation of 108.14 seconds), and an average time in system of 105.59 seconds. Graphs of the data are included below:

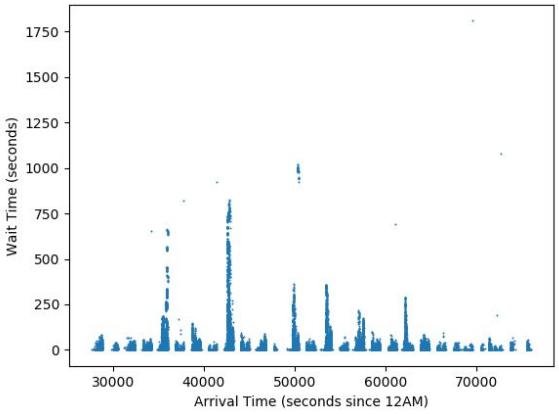
Time in System vs Origin



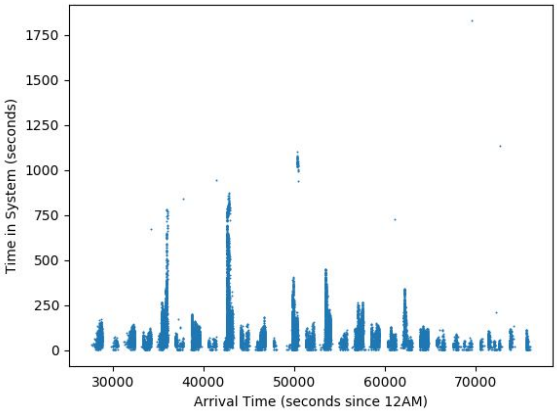
Time in System vs Travel Distance



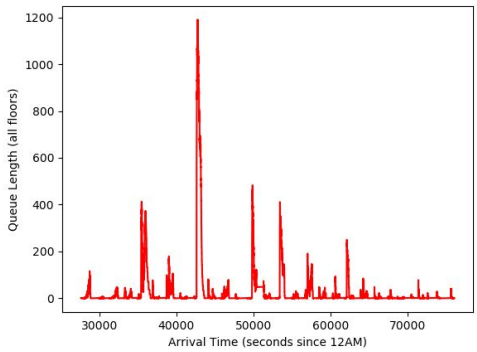
Wait Time vs Time of Day



Time in System vs Time of Day



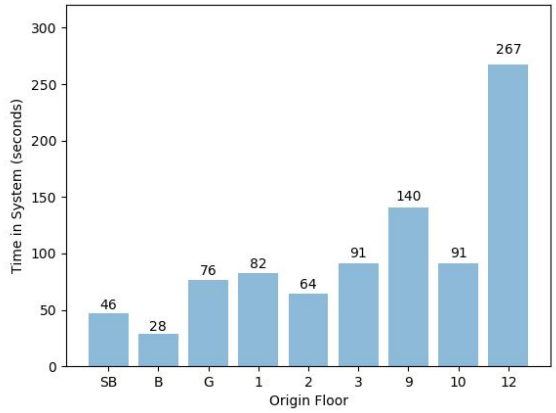
Queue Length vs Time



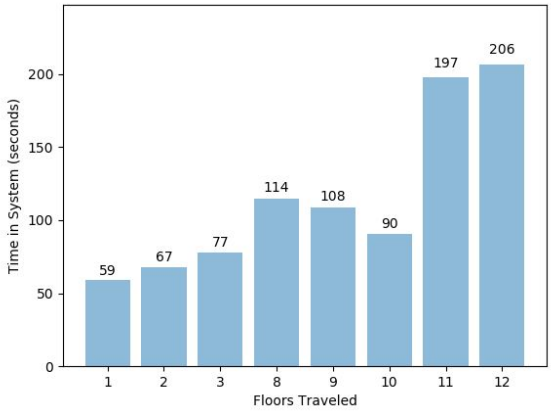
Fixed Sector (FS0)

Utilizing the look algorithm yielded an average wait time of 46.26 seconds (standard deviation of 98.96 seconds), and an average time in system of 95.73 seconds. Graphs of the data are included below:

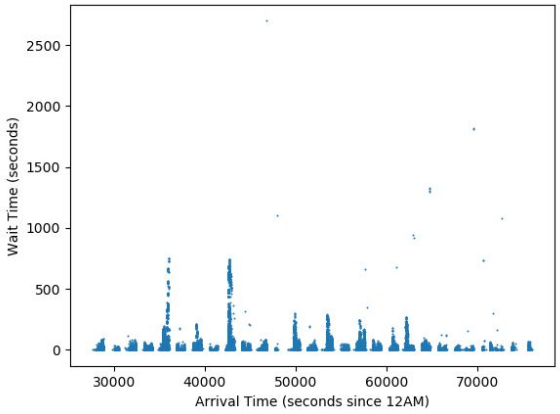
Time in System vs Origin



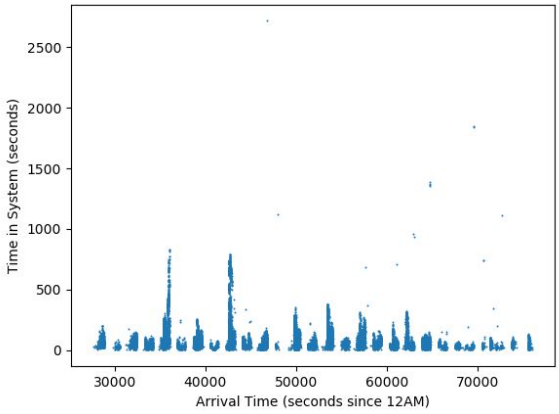
Time in System vs Travel Distance



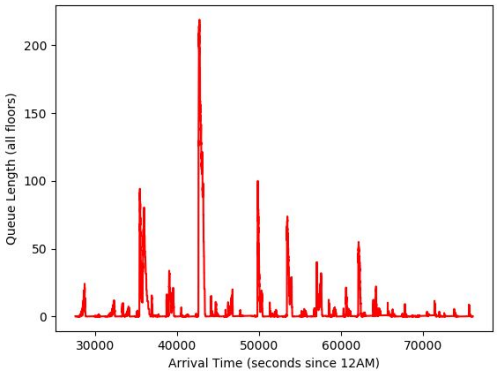
Wait Time vs Time of Day



Time in System vs Time of Day



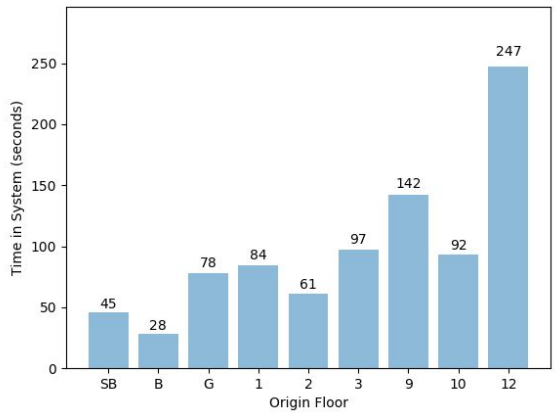
Queue Length vs Time



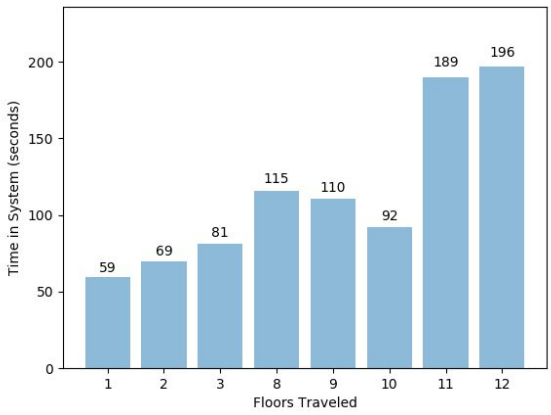
Time Dependent Fixed Sector (FS4)

Utilizing the look algorithm yielded an average wait time of 46.87 seconds (standard deviation of 115.72 seconds), and an average time in system of 96.66 seconds. Graphs of the data are included below:

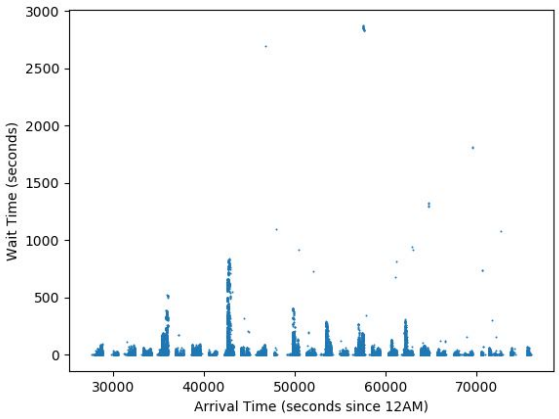
Time in System vs Origin



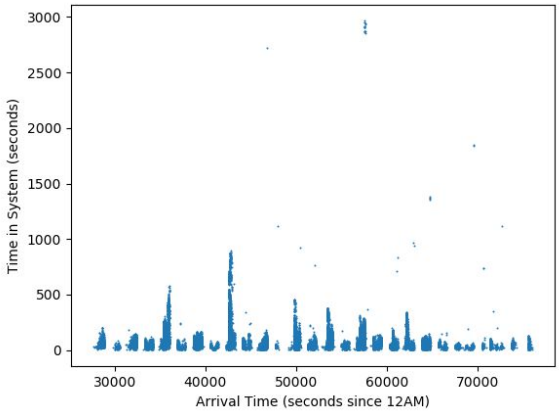
Time in System vs Travel Distance



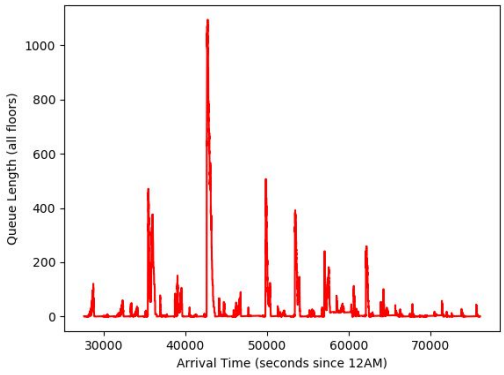
Wait Time vs Time of Day



Time in System vs Time of Day



Queue Length vs Time



Analysis

Best Algorithm

As expected, Scan and Look did not perform very well compared to the other algorithms. Nearest Car First is a greedy algorithm and performed fairly well. FS0 and FS4 had the best results. Through multiple runs, depending on the input, FS0 and FS4 would vary between which one performed better by always remained close in performance. The algorithms are extremely similar with a slight difference (Time Priority) that is aimed at reducing the number of outliers and increasing peak performance. However, the reduction of “peaks” during peak performance (refer to graph) may result in a higher overall wait time for the system but in some instances may do better.

Times of Day with Lowest Average Wait Times

Although there were definitely peaks, there didn’t appear to be any normal-like distribution. The peaks tended to be sporadic, tall, and short-lived. In nearly all of the charts showing queue length vs time, the behavior can be clearly seen. The largest peak tends to be around lunch time (with many classes getting out at that time), but there tends to be a peak nearly every time a set of classes let out, and the peak dies down shortly after. The best times to take the elevators (according to our simulations) seems not to be a specific time of day, but the times in between classes. Most classes are scheduled every hour and a half or two hours starting at 8AM, with many classes letting out right around noon. If an elevator is requested at an odd time in between classes, there is next to no other traffic.

This also makes sense, and highlights a flaw in the simulation assumptions, as all of the traffic was based on class times. In order to reduce this in the future, either more noise should be added or additional data should be gathered on other interfloor traffic within the building.

Correlation Between Origin and Wait Time

Although across algorithms the time in system tended not to exhibit large discrepancies in wait time between floors, there was a notable exception of the 12th floor. In nearly every algorithms simulated, arrivals on the 12th floor took significantly longer to service. In the FS0 and FS4 algorithms, wait times on the 12th floor were nearly double the next largest wait time (~250 vs ~140 seconds). It is most likely that the way the algorithms were implemented failed to prevent this problem of slight starvation. Due to the proprietary nature of the specifics of real-life elevator algorithms,

there is not a very good basis to see how the implementations differ. With the exception of the 12th floor, there didn't seem to be a significant correlation between origin floor and wait time.

Comparison to Walking Time

While there isn't a clear answer to this question, the experimental results make the tradeoff more quantifiable. The average time in system when traveling only one floor tends to hover right around 60 seconds. Using this knowledge, individuals can make an informed decision as to whether they think they can beat the elevator to their floor, and it it's worth the effort. What may make the argument for taking the stairs more compelling, however, is that in many cases the variance of the waiting time tends to be quite large. This means that if arrival time is important (for example, if someone shows up exactly 2 minutes before class), then taking the stairs might be a better bet than waiting for an elevator.

Further Discussion

One of the biggest hurdles facing this project was obviously the elevator algorithms. However, one specific roadblock for which we had the biggest problems was "idle checking". Originally, the algorithms were written without consideration of checking of the "idleness" of an elevator. However, upon running some of the simulations and examining the output, we realized that there were rare edge cases that would result in outliers in wait time. Thus, we attempted to fix these outliers by adding in checks to the code to do something if an elevator was in its "idle" state (not moving towards any destination or stopped with passengers). Thus, to fix this, we tried to alter the "figure of suitability" algorithm implemented in nearest car first that was extended to Fixed Sectoring Common Sectors and Fixed Sectoring Priority Timed. Oddly enough, adding these idle checks notably decreased overall performance, and only resolved outliers by making more outliers. Not only did it decrease the results, it also greatly impacted the runtime and made it take our simulations longer to run. Ultimately, we decided to scrap the idle-checks for more consistent and better overall runtime at the cost of a few outliers due to infrequent edge cases. From our observations, typically out of ten thousand generated arrivals, the edge cases generated one to four outliers in wait time for an elevator.

Moving forward, the algorithms we implemented were generalizations of the most publically well-known algorithms. In industry, elevator algorithms are comparable to trade secrets and are highly valuable. Another alternative that we omitted from implementation were dynamic sectors. When an algorithm uses dynamic sectoring, it

actively changes the sectors for each elevator based on the current state of the entire system to optimize performance.

Conclusion

Overall, we learned that elevator algorithms are extremely delicate. A small change or tweak can have a drastic affect on the outcome. Therefore, the optimal algorithm cannot be decided until you have simulated the system with different parameters and input data. As usual, the answer to the most optimal algorithm is “it depends.” Elevator scheduling is an NP Hard question, so approaches will vary wildy in results depending on the given environmental variables.