ECM1414 Coursework - Lift Control

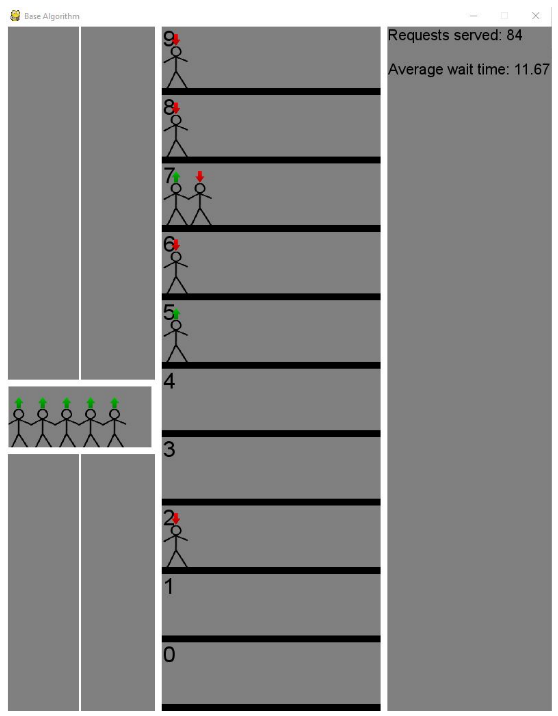
My Improved Algorithm

My improved algorithm starts the lift from the bottom floor. While there are no requests on any floor or in the lift, the lift will travel to the middle floor and wait there for requests to be in the optimal position pending a new request on a random floor. When a request spawns and the lift is empty, the lift will travel to the request that has been waiting the longest picking up any requests heading in the same direction as the lift. The lift will then travel to the destination floor of that request also picking up any heading in the same direction as the lift.
This algorithm is better than the base algorithm as it can dynamically respond to new requests which is particularly effective compared to the base algorithm when the volume of requests is low. This is because the base algorithm has the potential to just miss a new request on a lower floor and have to travel all the way to the top floor and back down again to pick it up. Whereas, if the improved algorithm was in the same situation it would switch direction and pick up the request much quicker. This advantage starts to level out slightly as the volume of requests increases. However, in realistic scenarios, the improved algorithm will always be faster on average.
I used three dictionaries to store the requests each with a different state of the requests. The first contained all requests waiting for the lift on each floor. The second contained all requests currently in the lift. The third contains all requests that have been served by reaching their requested floor. Each request in these dictionaries contained the floor they spawned on, the floor they would like to go to, the direction they would like to go and their total wait time. The total wait time is incremented by one in the first and second dictionaries each time the lift travels past a floor. When a request is picked up or dropped off from the lift at their destination floor, the request is transferred from the first to the second or the second to the third dictionaries respectively. Requests waiting to be served are in a queue within the first dictionary. The lift will travel to the first element in that dictionary if it’s empty picking up requests on the way if they’re going in the same direction. Requests inside the lift are also in a queue within their own dictionary.

Weekly Logs

January 6th - January 12th

Firstly, I developed the logic for the base algorithm in python whereby the lift is continually going up and down picking requests up if they are going in the same direction as the lift and the lift is not already full. I used three different dictionary objects to store the requests. The first storing all requests waiting to enter the lift, the second storing all requests currently in the lift and the third storing all requests that have been served. Once this appeared to be working I implemented this logic with some graphics using pygame. Once working, these graphics revealed some bugs to my logic that were not as obvious when printing data to the command line. The lift would occasionally miss requests that it could've taken. This was a problem with removing the last element in a list rather than a specific element.



```
Requests served: 84

Average wait time: 11.67
```

January 13th - January 19th

In the second week I was looking to implement a GUI to enable the user to select which algorithm to run and configure the simulation settings such as number of floors, spawn rate of requests and the speed of the lift. I tried thorpy, pgu and Pygame GUI however none of these worked very well with the existing simulation graphics. I was unsuccessful in implementing an existing pygame gui library with my simulation graphics.

January 20th - January 26th

N/A

January 27th - February 2nd

Still looking for a pygame gui library, I found Pygame Menu. This library had good documentation and example programs which were helpful when implementing the menu code into my existing program. Using one python file proved to be much easier than trying to have different windows in various python files which was causing a whole host of issues.

February 3rd - February 9th

During this week I further improved the user interface by implementing the settings menu, a way of switching algorithms before running the simulation and a button to quit the simulation to return to the previous menu. I also added some figures to the simulation window displaying which algorithm was currently running, the total number of requests that had been served and the mean wait time measured in the number of floors the lift had travelled from the point at which the request had spawned to when that request had reached its floor.

February 10th - February 16th

The last main thing I had left to program was the improved algorithm. Before programming the base algorithm, I wasn't exactly sure my improved algorithm was going to work. However, as I was programming the base algorithm the flaws in its design were becoming more visible. I realised the requests needed to have some sort of priority system. I attempted to program this for my improved algorithm but I could never get it to work. Instead the lift would judder up and down when it reached a floor with a request on it.

February 17th - February 23rd

At the beginning of the week I fixed the improved algorithm and made some small changes to the way it works. For example, when the lift is empty and there are no requests to be served the lift will go to the middle floor and wait for more requests to spawn. I then started to start running some simulations and graphing the average wait time as the number of floors varied for each algorithm. This started off as expected however after 15 floors the average waiting time started to decrease with a peak every 5 floors.

February 24th - March 1st

After a few days, I eventually found it was a problem with the ‘current_floor’ variable that the lift was on. As the number floors in the graphics window increases they get closer together giving the program less time to detect a floor change. This was resulting in floors being skipped from the waiting time of the current requests even though requests were being picked up and dropped off of these skipped floors. I fixed this by calculating the total distance travelled by the lift in pixels and used the height of the floors to calculate the total number of floors travelled. Each time this variable incremented, the wait time of all requests waiting on a floor or in the lift would also be incremented by one.

March 2nd - March 8th

Added some additional code to run when the simulation is quit. The number of floors and the average waiting time is saved to one of two csv files depending on the algorithm used for the simulation.

March 9th - March 15th

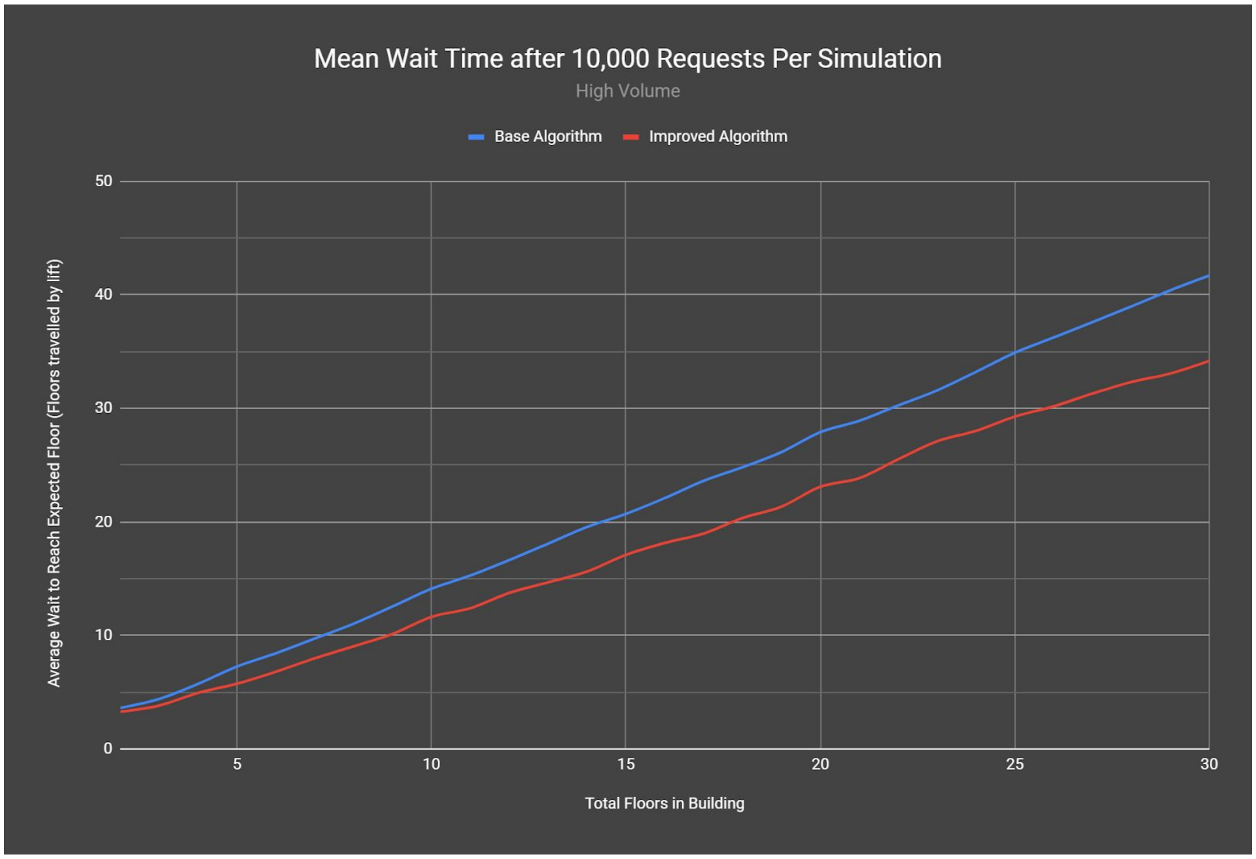
Began running simulations and making graphs from the csv data varying the frequency of requests and the number of floors varying from 2 to 30. Created and uploaded the video demo containing various numbers of floors, request frequency and lift speed demos for both algorithms side by side running on two instances of the same script.

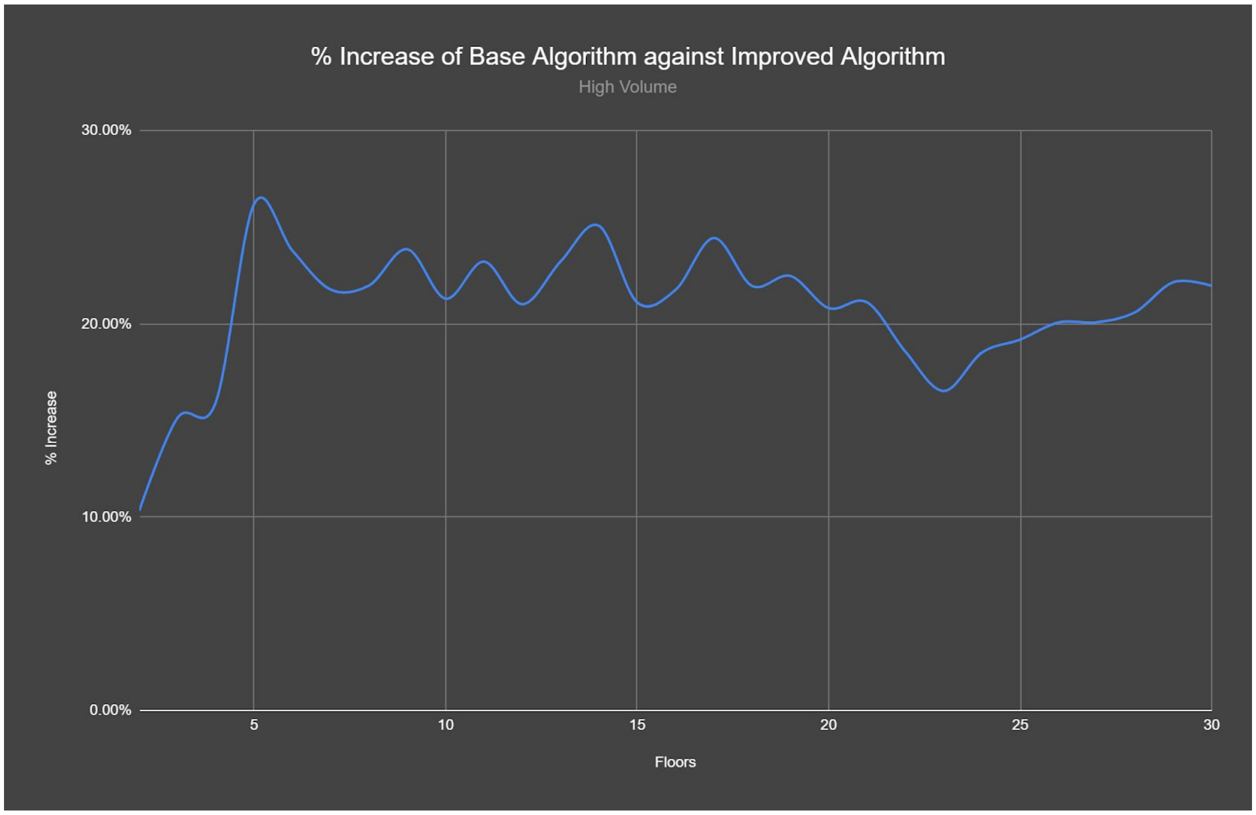
Performance Analysis

Each simulation ranges from 2 to 30 floors and is run until the total number of requests served reaches 10,000. The speed of the lift is increased for these simulations however this shouldn't affect the results as the time is based on the number of floors passed by the lift. Overall, the lower the demand, the greater the difference in performance is between the base and the improved algorithm. This is due to the fact that when the demand increases and there are a large proportion of floors containing requests to be served, the improved algorithm's lift takes a similar path to the base algorithm in which it continually moves up to the top floor and back down to the ground floor.

High Volume

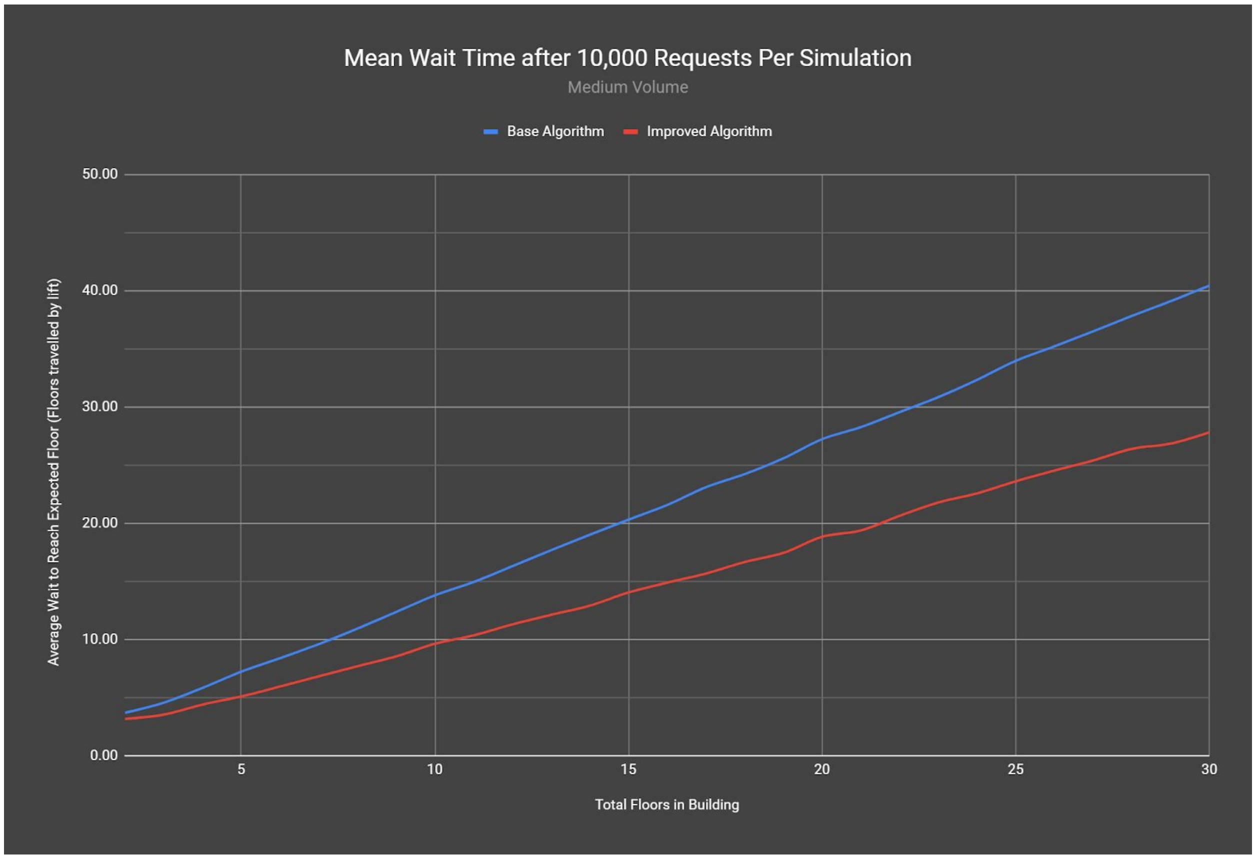
On average, the improved algorithm is 20% faster than the base algorithm when under high volumes of requests. However, when the number of floors is small, the base algorithm can be as little as 10% faster.

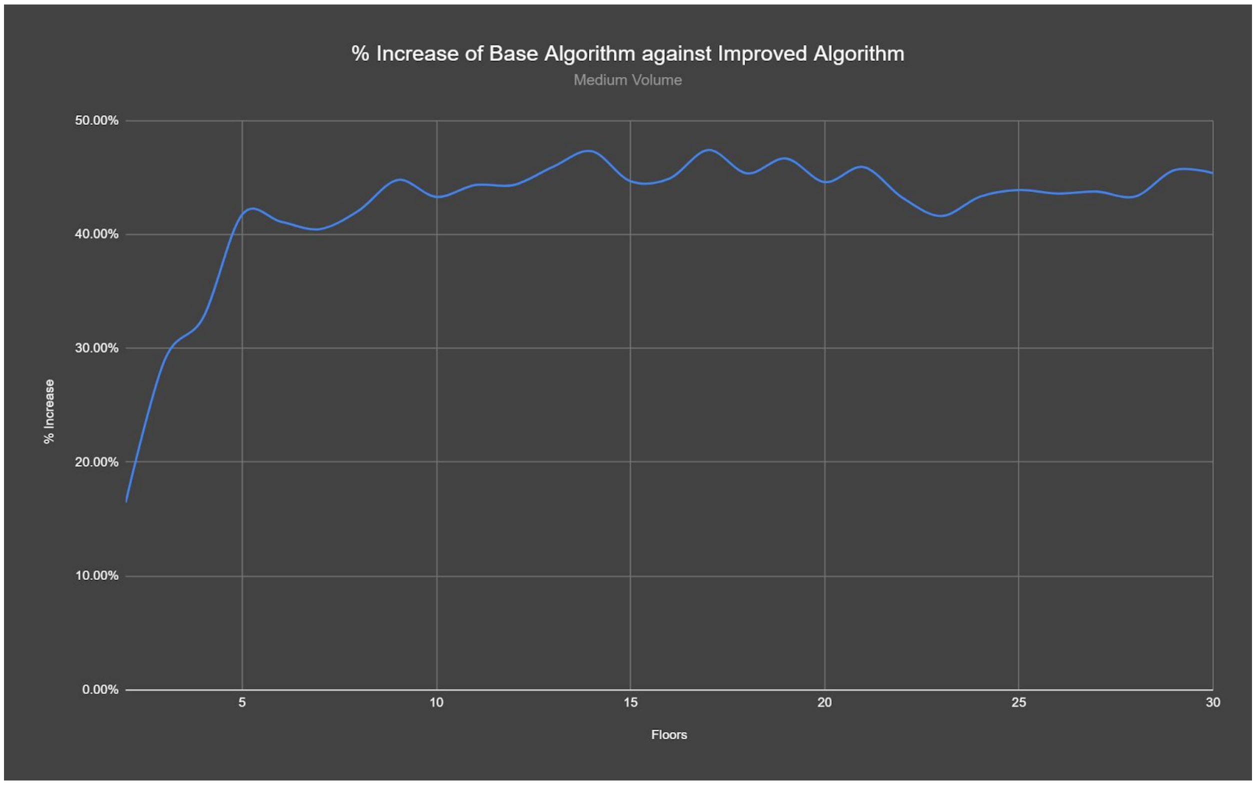



% Increase of Base Algorithm against Improved Algorithm
High Volume
30.00%
20.00%
10.00%
0.00%
% Increase
5
10
15
20
25
30
Floors

Medium Volume

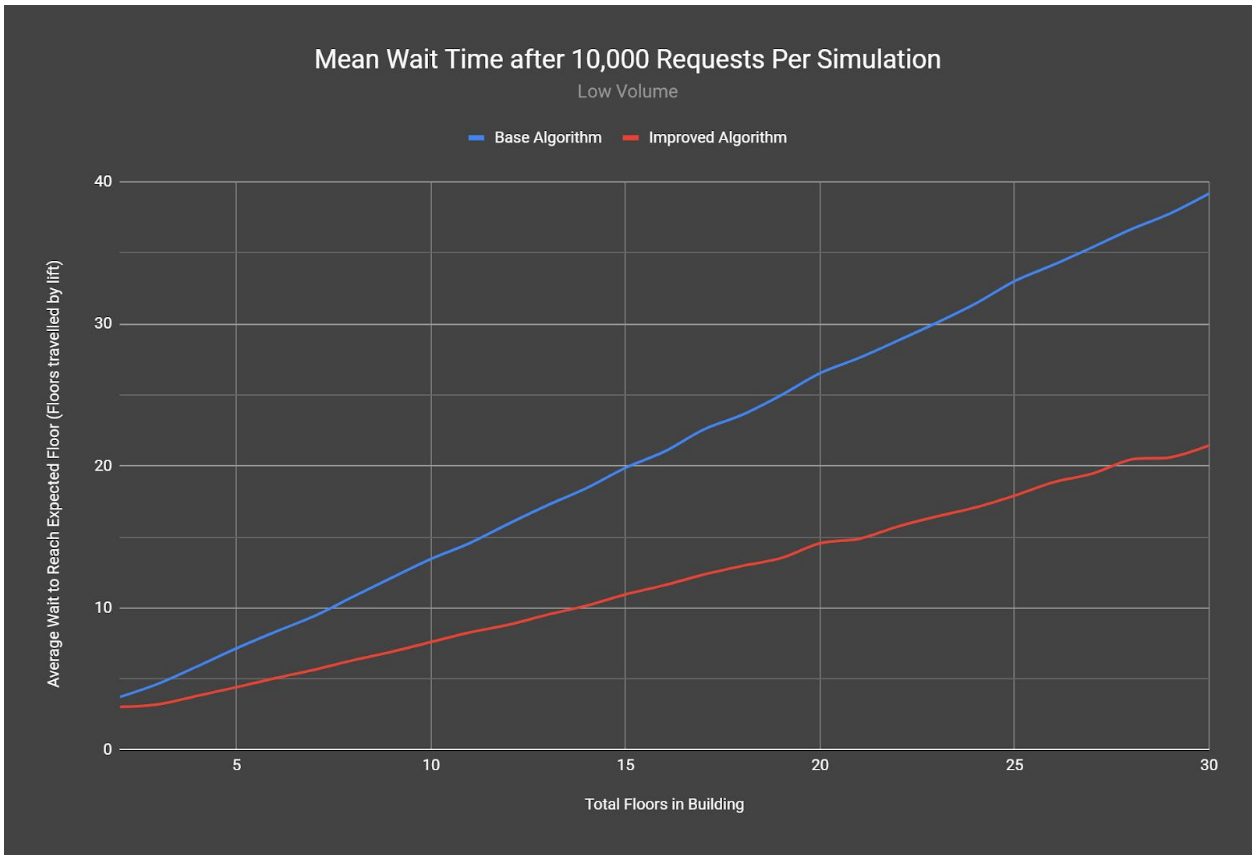
A medium volume of requests displays a middle ground between the low and high volume charts. On average, the improved algorithm is 42% faster than the base algorithm when under high volumes of requests.

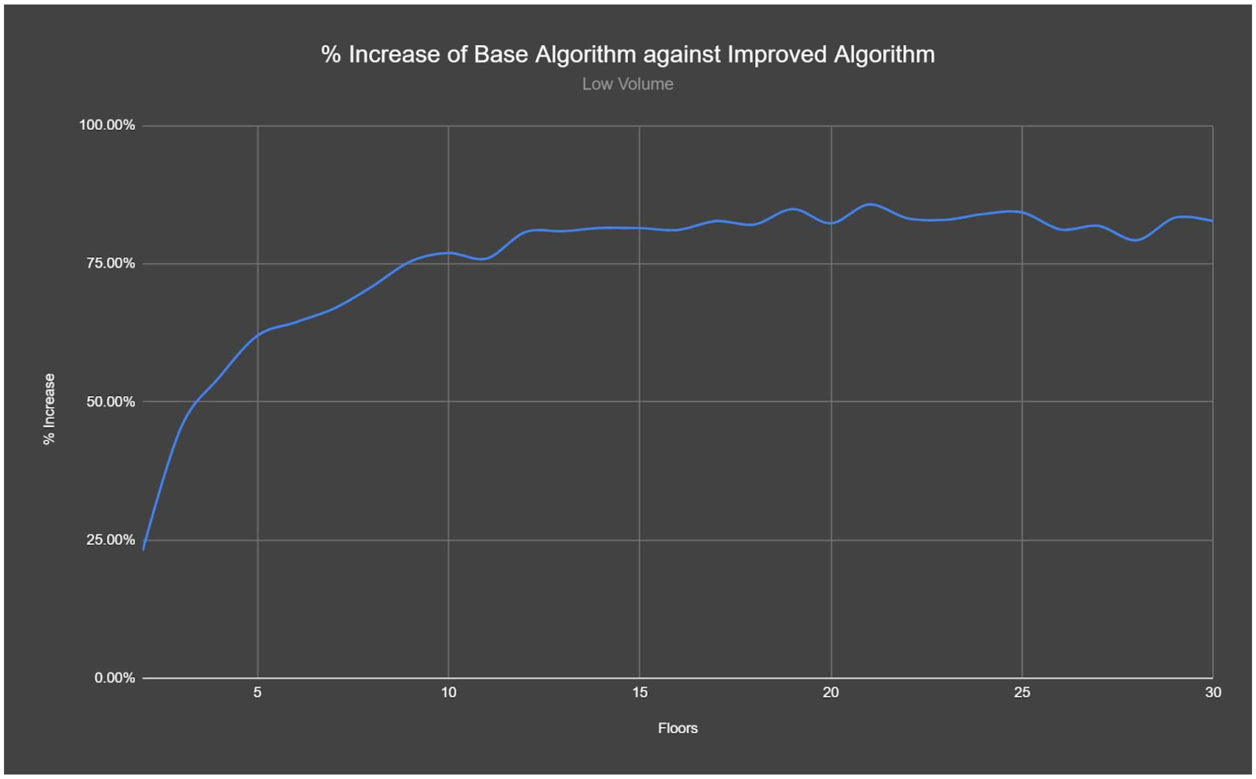



% Increase of Base Algorithm against Improved Algorithm
Medium Volume
% Increase
50.00%
40.00%
30.00%
20.00%
10.00%
0.00%
5
10
15
20
25
30
Floors

Low Volume

The algorithm's speed diverges much quicker when the volume of requests is lower. With just two floors the base algorithm is 23% faster at serving requests. At around 12 floors or more, the percentage increase of the speed of the improved algorithm over the base algorithm levels off at just over 80%.




% Increase of Base Algorithm against Improved Algorithm
Low Volume
100.00%
75.00%
50.00%
25.00%
0.00%
% Increase
5
10
15
20
25
30
Floors

Video Demo

https://youtu.be/l_47qbni9mY

References

Pygame Menu library - https://github.com/ppizarror/pygame-menu