CS166 Assignment 1 - Elevator Simulation

Armin Hamp, Hung Nguyen, Fabian Okafor, Elena Cai

2019 September 17

Code

Our code is available both in the following link, as well as the .ipynb file submitted alongside this pdf.

https://gist.github.com/hamparmin/e0b7a9fde6db0319ee7d1c8715a68742

Elevator Strategies

We used three different strategies in our implementation:

1. First come – first serve. Passenger's requests are served according to the order they have entered the elevator.
2. Going up all the way, then going down one floor at a time, picking up and dropping off passengers on the way.
3. Going up only when a passenger in the elevator wants to go up or someone requested the elevator on the upper floor. Once at either local/global top, the elevator descends.

Efficiency Metrics

We used two efficiency metrics to measure the performance of our outlined strategies:

1. Total number of steps taken by the elevator to serve all passengers.
2. A total trip time measured for each passenger using the time module.

Results

We created two experimental settings to test out strategies' performance:

1. Measuring performance by keeping the number of floors in our building constant, while increasing the number of passengers. This method comprised the steps:
    i. Populate the building with the given number of passengers based on a uniform distribution
    ii. Averaging metrics for multiple trials for a single input
    iii. Storing these averages, then plotting these figures against increasing passenger numbers
2. Measuring performance by keeping the number of passengers in our building constant, while increasing the number of passengers. This method comprised the steps:
    i. Populate the building with the given number of passengers based on a uniform distribution
    ii. Averaging metrics for multiple trials for a single input
    iii. Storing these averages, then plotting these figures against an increasing number of floors

The graphs below show our results broken down by metric and experimental setting.[1]

[1] #modeling: In this section I explain how we framed our simulation in order to describe our Elevator strategies' efficiency. The metrics, the experimental settings and graphs plotted provide clear, easily interpretable data about our simulation and how it performed.

Results for Experimental setting 1.

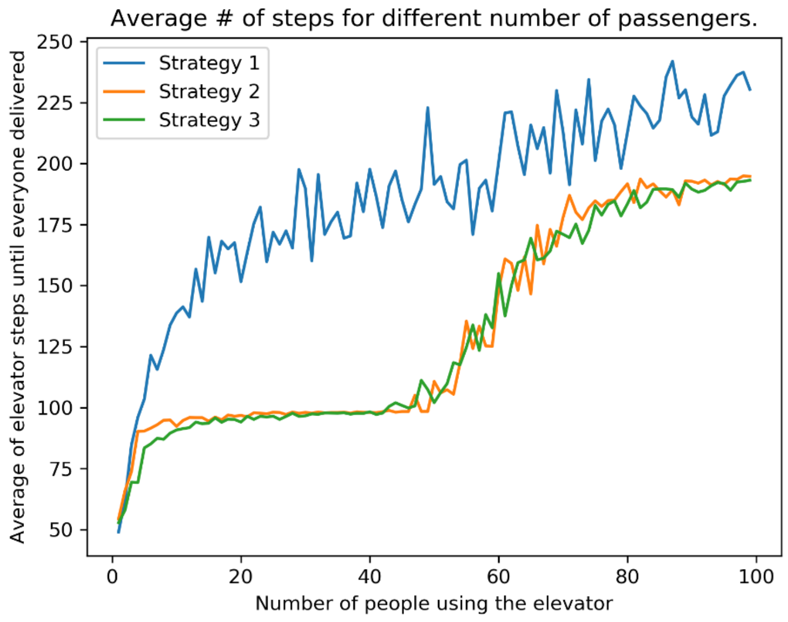

Figure 1.

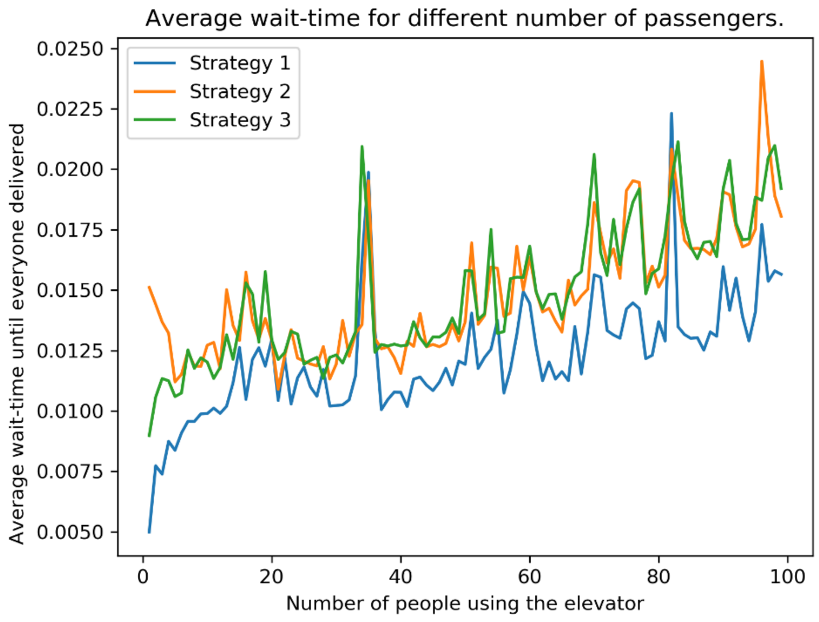

Figure 2.

Results for Experimental setting 2.

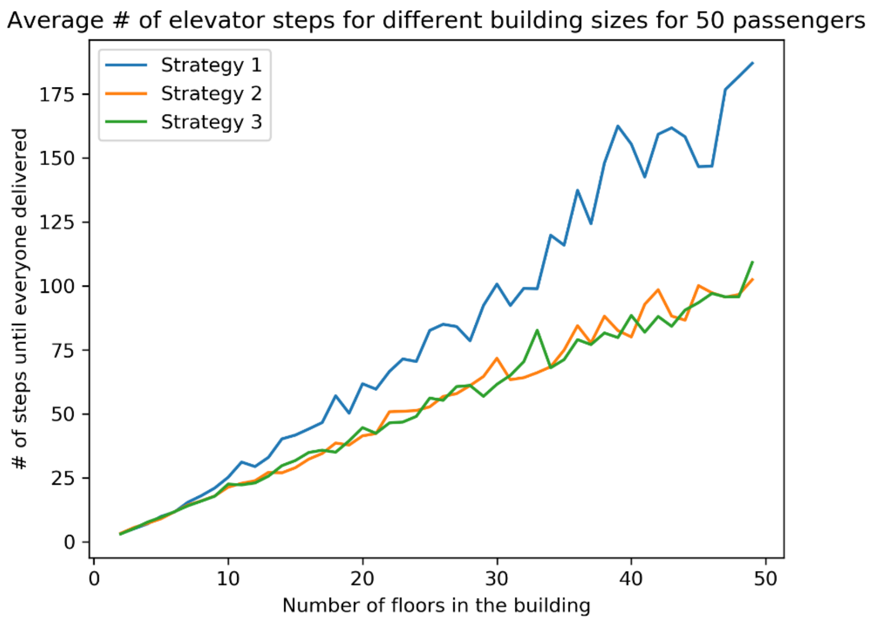

Figure 3.

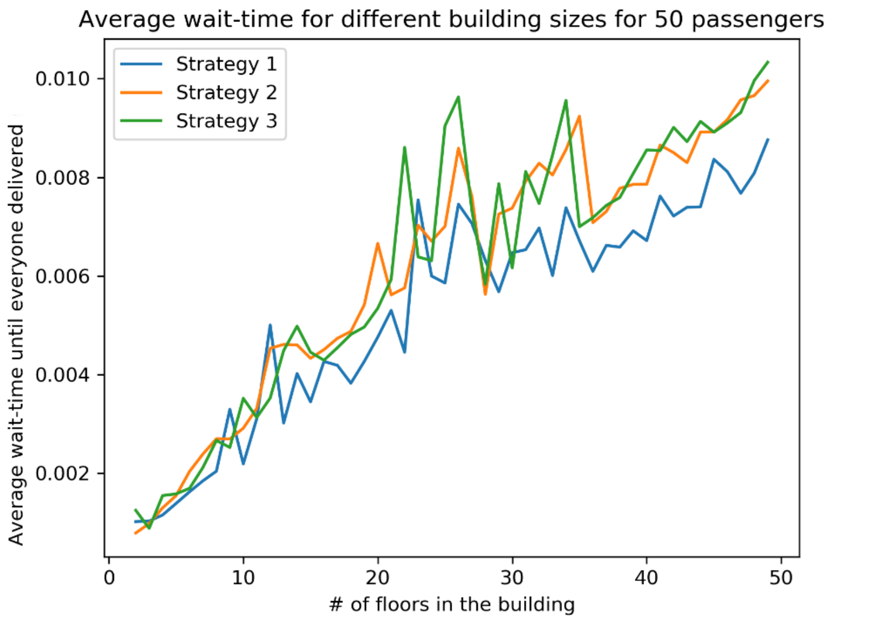

Figure 4.

Interpretation of Results

In terms of the average number of steps, the elevator had to make to deliver all passengers, we have a clear losing strategy. As seen in Fig. 1 Strategy 1 was a clear loser following the line of a radical function from the onset, while Strategy 2 and 3 increased marginally until 50 passengers. Also, Strategy 1 performed considerably worse when holding the number of passengers constant and increasing number of floors in the building, see Fig. 3.

However, Strategy 1 performed the fastest for individual passengers. In both experimental settings, it performed faster than Strategy 2 and 3, see Fig 2. and Fig 4. This means that we have encountered a strategy that is the worst in one and the best in another metric. Strategy 1 works on a first come – first serve basis, and thus it could mean that it worked well enough in our building set-ups to be able to transport passengers the most efficient in terms of time, at the cost of making more overall steps than other strategies. We must be careful to interpret Strategy 1 as the best strategy overall for passengers. This is because the maximum number of passengers we have simulated was 100, which might just have been small enough to quickly transport everyone to their destination without having to make anyone wait considerably longer. I suspect that with an increase in passenger numbers (e.g. 1000) this might become the worst strategy overall.[2]

We also have to mention that we designed Strategy 2 and 3 to be very different in their nature. Strategy 2 is a rather simple (first go all the way to the top, then back to the bottom) algorithm, while Strategy 3 tried to mimic the real workings of an elevator (taking into consideration where the elevator has been requested and where current passengers are going). However, in their performance as seen in the graphs above they performed eerily alike. For both metrics (number of steps, wait-time), in both experimental settings, they produced the same results. Since the elevator when running prints out updates as to where it stopped, picked up and dropped off passengers, these updates served as a useful clue as to understanding what was happening. For the same distribution of people, the two elevators performed the same movements in approximately reverse order. This means that regardless of our careful planning to make Strategy 3 as realistic as possible, it simply ended up doing the opposite of what our simple Strategy 2 did. This is a good example of unexpected emergent behaviour of a complex simulation.[3]

[2] #utility: By recognizing the contrasting results of efficiency of Strategy 1, I explained how this might be a beneficial strategy for passengers (as they spend less time waiting) but worse overall for the operator of the elevator (as it does travel considerably more than under other strategies).

[3] #cs166-interpretresults: In this section I summarized the findings of our efficiency testing and drew the conclusions about most and least efficient strategies. I also highlighted the care that must be taken when evaluating these strategies based on our collected data (e.g. is strategy 1 really the best?) and I demonstrated my understanding of our model when comparing the behavior of Strategy 2 and 3.

Contributions

I have contributed to the coding of this assignment by:

- The first draft of our working algorithm
- Organizing calls to decide on the division of tasks
- Debugging code when further complications were added
- Recording metrics and plotting results

Takeaways about simulations in Python

Looking at the assignment initially, I thought I was going to be dealing with a straightforward task: Three classes, a couple of attributes and methods, the main loop to execute the simulation and we will be done. However, when it came to the writing of the code together with my teammates, this became a much more difficult undertaking. Careful planning was necessary to communicate to each other the logic according to which we planned to build the simulation. We realised that there are almost infinite ways to implement the Elevator simulation in code and had to decide on what aspects of the simulations we want to focus on. The initial choices of attributes and methods determined how we had to implement our strategies. This meant that when we had to add a complication, such as an elevator request button, we had to rewrite great chunks of our whole code.

While I am glad about the overall results, I want to emphasise the importance of planning before sitting down to code. This preparatory step, had we taken it more seriously, could have saved us a lot of time and effort. I also realized that having been given a particular goal to optimize our simulation for, would have made things more straightforward. If I were to do this task again, I would first select what aspect of Elevator behaviour (e.g. total distance travelled, passenger wait time, etc.) I want my simulation to best model, and consequently, design my code around it.[4]

[4] #strategize: In this section, I recognize how our group's lack of pre-planning was overall detrimental to our project and I explain how specific strategic approaches for a similar task in the future might help us save time and lead to more efficient work.