CS166 Elevator Simulation

Tomer Eldor

Group Members: Ben Imadali, Guilherme Nazareth de Souza, Collete Brown

Minerva Schools at KGI

Elevator Simulation

Introduction

Have you ever waited for a single elevator in a building so long that your mind starts wondering "how is this elevator even programmed to work?"

Well, there are many strategies for elevators to perform. It sounds like a simple problem, but the more you get in to it, you discover it involves more work than you thought. You may come up with some strategies (starting with: "open doors at every floor" to complex optimizations if you have modern destination setting interfaces for passengers and multiple elevators), but how do you know which one is best? Well, we could simulate it and test it according to some metrics of our choice.

We created a simulation of an elevator and passengers in a building. The building's floors and the number of passengers are inputted by the user. Then, we created two strategies and compared their efficiency in terms of what we wanted to maximize for: utility for passengers, in terms of waiting time cost and other metrics.

Strategy #1: The Naïve F.I.F.O strategy

As the first strategy, we created a naïve "First-In, First-Out" strategy. In this strategy, the elevator respects every call in order of pressing the button from start to end. It goes to the floor of the caller, picks her up, goes straight to her destination floor, and drops her off. Thus. Then, it reaches the calling floor of the next passenger in line, and brings him to his destination. Thus, it contains maximum 1 passenger at a time, or 0 in between passengers. It is not an optimized

FIFO, but simply the most naïve FIFO strategy. This is, intuitively, a very bad strategy in terms of efficiency of any sort. We will see these results quantified later.

Strategy #2: "Max_Floor" – Go Until Max/Min Floor

As the second strategy, we designed the following: the elevator would go up to the highest destination of passengers inside it, or to the highest floor with passengers wanting to go down, while stopping to drop off its passengers on the way or picking up passengers going in the same direction. It does the same in the other direction, with opposite directions respectively. The elevator re-evaluates this "highest floor" (or lowest, if going down) that it would stop at *on each floor*, such that if there would be new calls or new pickups with higher destinations, it would update its maximal destination accordingly.

Evaluation of Strategies

Naturally, Strategy #2 proved to be highly superior in terms of our efficiency metrics. We define "cost" in terms of time-steps: each floor traveled is one time-step, and each time the doors are opened and someone is being picked up or dropped off, we add another "time-step" worth cost, since it took more time. The metrics we use are:

1. Average cost – the most common metric, testing what is the average time cost between all passengers
2. Average squared cost – squaring costs first makes the higher ones exponentially higher, thus weighs higher costs more heavily. This metric would be higher when the higher quantiles of the costs are higher, and thus serve as a metric that considers both average costs while accounting high costs. As we know, we are accustomed to wait for the elevator a short time period, so there is little irritation at first; but when we have to wait for long minutes, then we get drastically more irritated.

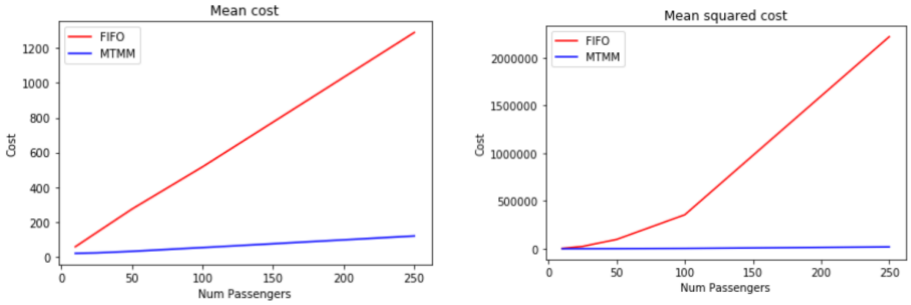
3. Median cost – the average would be skewed by the high outliers. The median tells us more about the common case in the center of the distribution of costs.
4. Maximum cost - the worst case scenario.

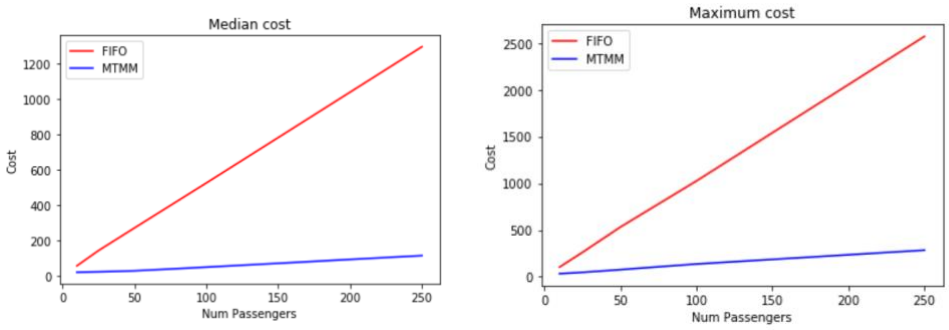
We tested a few points by increasing the number of passengers from 10 until 250, while keeping the number of floors at 10. The results show that while increasing the number of passengers has little effect on passengers' costs in the Max_Floor strategy, it causes extreme cost increases in the FIFO strategy.

Let's take for example the initial costs accrued by the two strategies with 10 passengers. Max_Floor had an average cost of 16.5, while FIFO had over double as much: 38.74. The medians were similar. However, they are both quick relatively to what happened when we increased the number of passengers. The Maximum cost and the average squared costs are already considerably higher for the FIFO (times three or 6.67, respectively).

MAX FLOOR	**FIFO**
Average cost: 16.5	Average cost: 38.74
Average squared cost: 303.9	Average squared cost: 2030.36
Median cost: 15.0	Median cost: 37.0
Maximum cost: 29	Maximum cost: 91

As we increased the number of passengers and tested the cases of [10, 25, 50, 100, 250], these metrics grew in an extremely concerning way for FIFO, while not so much for our Max_Floor strategy:





We see that in all metrics, Max Floor ("MTMM") strategy is unquestionably superior. Not only that, it is also quite good by itself – it actually didn't increase the costs per passenger drastically while increasing the total number of passengers. If we had to choose a strategy, it would be our second one, hands down.

Assumptions

Lastly, we should note this model is overly simplified and has many assumptions which don't equal the real world. Since the list is almost endless, I'll give just some examples. First, the actual time costs of traveling between floors and opening doors should be different: we assumed that each passenger entering increases the time cost, whereas in reality the doors opening would have a fixed cost of, say, 5x the time it takes to move up one floor, and then only above 3-4 people getting in the elevator at the same time there would be more time added if passengers press "open-door" to keep it open while entering. This would also happen at random sometimes, and a more realistic simulation could even account for that, but that would be too detailed for our purposes. Moreover, if we want to be even more precise, there is "takeoff" and "slowdown" time costs – the time for the elevator to accelerate or decelerate into its normal speed, which further increase the time cost per stopping at a floor.

Additionally, the FIFO strategy made people wait so long that they would have taken the stairs instead. Then they would give up and our simulation would change.

Main Contributions

I wrote the base and majority of the Elevator.py class, which was the heart of the algorithm. I suggested that we could start by implementing a naive FIFO strategy and then modify it to a more reasonable one, as a proof-of-concept at first, which ended up being used as one of the strategies. Guilherme and I formalized the second strategy and finished programming together the entire classes. Later, after Ben and Guilherme worked on some bugs, Ben and I worked for another half-day on worked on many bugs and completion of the project. Ben had the most professional and practical experience so he helped much in leading and teaching us best practices, including GitHub.

My Main Takeaways

This was my first time writing a simulation, and (almost) my first time working extensively with Classes. I learned a lot about how to create classes and work with them. I learned from Ben a lot about better working practices, about dividing the code into different files for the different classes and uses and importing functions from other files, about how to work with GitHub, and about team programming, which I'm very happy to learn. I also learned that it's crucial to start simple. This simulation sounded like it should be very simple at first, but the more we got in to it, the more complex it became. I had many ideas of making it more realistic or optimal, but we wouldn't have finished it in time. Beyond the mere building of the algorithm, which was more complicated than I thought, the days of debugging take long. I learned from Ben

his methods for debugging and working methods – from his IDE recommendations to how to write more concisely.