Author: m-pol, m-pol@users.noreply.github.com
Elevator Research Project

Abstract

Elevator dispatch algorithms are algorithms specifying methods of assigning calls to elevators in a multiple-elevator system. I did research into how elevators work and implemented my findings in a software model to compare the theoretical efficiency of elevator dispatch algorithms. There are several algorithms that elevators use in the real world to determine which floor to go to, including Nearest Car, Sectoring, Dynamic Sectoring, and others. I first wrote the infrastructure for an n-car elevator system implementing a general abstract "Strategy", and then I wrote the logic for the following algorithms: Nearest Car, Sectoring, and, as a benchmark, Random, which assigns calls to random elevators. See Results and Discussion sections for detailed performance analysis. Main takeaway: Nearest Car and Sectoring performed better than Random, which is expected, with Sectoring having lower variance but higher average time.

Introduction

Elevators are an important part of everyday life for millions of people across the world. There are approximately 900,000 units in United States alone, and it is estimated that the average elevator carries 20,000 people per year [1]. As such, research into the efficiency of the systems which control them are of considerable practical importance. Minor inefficiencies add up over time and passenger count, and can reduce people's productivity in aggregate.

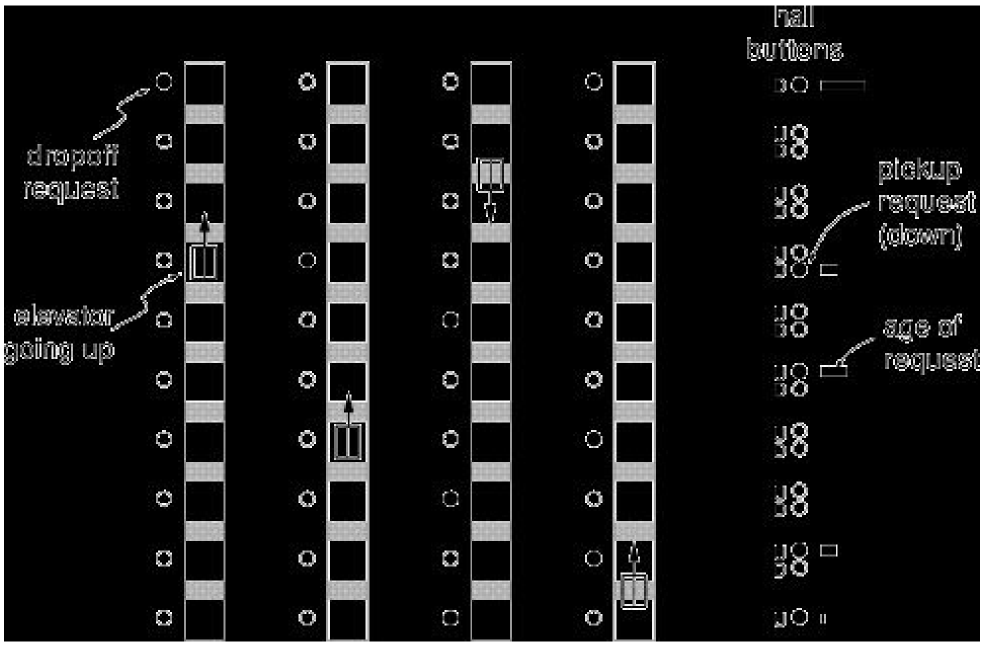
In general, there are two primary constraints on elevator movement: energy and time. A considerable challenge lies in balancing these two often conflicting constraints. For example, it is almost always better for passengers' wait times for idle elevators to return to the ground floor after some time, but movement that is not strictly necessary consumes extra energy. Conservatively speaking, around 5% of a building's energy consumption is due to elevators [2], and building owners are always looking to cut costs. However, most analysis of elevator algorithms is centered around reducing time, rather than cost, since minimizing cost is a fairly straightforward problem (just use one elevator and load it until capacity and move like a one-elevator system would move). This is of little practical importance, since it is likely to result in massive wait times. As such, the main goal of the main algorithms is to reduce wait time.

However, different metrics are used to measure wait time reduction. For instance, average waiting time, average system time (waiting time + time in elevator), etc. Some analysis focuses on average squared waiting time, which tends to keep the waiting times low while also keeping the maximum waiting time for a given passenger down [3]. I will focus on a number of different wait time metrics in my analysis.

There are a number of factors that make this an algorithmically “difficult” problem. For example:

- Number of elevators in the system
- Building specifics
    - Number of floors
    - Popular floors
    - Peak hours

As such, there is no one algorithm that works best across all buildings. Due to the large number of complicating factors, the problem is NP-hard (reducing to a travelling salesman problem) [4]. Consider that the following elevator system



has approximately 10^{22} possible states [5] – this gives some indication of the problem’s complexity.

Materials and Methods

I modelled a building with 21 floors and 4 elevators to simplify the analysis, and focused on implementing the Nearest Car and Sectoring algorithms. The specification for the Nearest Car algorithm is found in the foundational resource on this subject, Barney's *Elevator traffic handbook: theory and practice.* For each call, and for each elevator, it calculates FS, a figure of suitability, for that elevator. The call is given to the elevator with the highest figure of suitability.

Specification: Let N = # floors - 1, d = distance in floors between elevator and passenger call. If an elevator is moving towards a call, and the call is in the same direction, FS = (N + 2) - d. If the elevator is moving towards the call, but the call is in the opposite direction, FS = (N + 1) - d. If the elevator is moving away from the point of call, FS = 1. See the NearestCar class for the code.

Sectoring was implemented by assigning calls in sector 1 to elevator 1, sector 2 to elevator 2, etc. Each sector is a fixed set of floors of approximately equal length. See Sectoring class for the code.

Random was implemented by assigning calls at random, such that the probability of being assigned to one elevator vs. another was equal.

Results

A random set of 20 calls was generated, and a random set of starting positions and starting directions was generated for each elevator. The calls were then assigned to elevators either by the Random algorithm, the Nearest Car algorithm, or the Sectoring algorithm.

This was run for 10,000 trials for each algorithm. For the raw data sets, see the zipped .txt files. The first entry in each row indicates the number of steps (i.e. going up or going down) that the algorithm required before each call was done being serviced. The second entry is the total time that each elevator took servicing the calls. Time was calculated for each elevator by adding 4 seconds for each floor traversed and 10 seconds each time the elevator stopped. If the elevator was done servicing calls, no time was added. The third entry is the average time across the 4 elevators. The fourth and last entry is the maximum of the times of the four elevators. This gives an idea of how long the whole system spent working before each call was serviced.

Across 10,000 trials, we see:

<u>Random:</u>

Steps: 46.986
Total time across all elevators: 632.301
Average time across elevators: 158.07525
Maximum elevator time: 211.5748

Nearest Car:

Steps: 41.9069

Total time across all elevators: 500.9484

Average time across elevators: 125.2371

Maximum elevator time: 223.3452

Sectoring:

Steps: 38.8678

Total time across all elevators: 538.5728

Average time across elevators: 134.6432

Maximum elevator time: 178.4096

Discussion

We see that the number of steps is shorter for Nearest Car than Random, as expected. We also see that total and average time across elevators is significantly lower for Nearest Car than Random. This gives a closest idea of how short the wait time + transit time is for the average person, so is probably the best metric to look at for overall quality. Lastly, and curiously, we see that maximum time is lower for Random rather than Lowest Car. This perhaps indicates that Lowest Car can be bad for some unlucky people, but for the good of the average passenger overall.

Sectoring also does better than Random, with a lot fewer steps, and even fewer steps than NC. This makes sense because each elevator is dedicated to only one section, which makes the variance lower. The total and average is better than Random, but worse than NC. In line with the variance being lower is the maximum elevator time being much lower than both.

Future work would include implementing more algorithms, such as Dynamic Sectoring. The code I've written should make it straightforward to implement other algorithms using the Strategy interface. Future work could also include making the system so that new calls can be made dynamically, i.e. with the algorithm not just assigning a pre-made set of calls but continuously assigning new incoming calls to appropriate elevators. Lastly, work could be done adding the ability to work with other constraints, such as making some floors more popular than others and seeing how each algorithm fares.

References

[1] National Elevator Industry, Inc. Elevator and Escalator Fun Facts. http://www.neii.org/presskit/printmaster.cfm?plink=NEII%20Elevator%20and%20Escalator%20Fun%20Facts.cfm.

[2] H. M. Sachs, "Opportunities for elevator energy efficiency improvements." Washington, DC: American Council for an Energy-Efficient Economy, 2005.

[3] G. C. Barney. *Elevator traffic handbook: theory and practice*. Routledge, 2003.

[4] J. Dong and Q. Zafar. *Elevator Scheduling*. Columbia University. Available: http://www.columbia.edu/~cs2035/courses/ieor4405.S13/p14.pdf

[5] A. Barto and R. Sutton. *Reinforcement Learning: An Introduction.* The MIT Press, 1988.