A Better Covid-19 Elevator Algorithm

Ethann Yakabuski 100961945
Supervisor: Evangelos Kranakis
Carleton University Fall 2020

December 18, 2020

Table of Contents

1: Introduction

Preamble: The "Standard Elevator Algorithm" is a very efficient and arguably the most optimal method to deal with a buildings job requests. This "Standard" algorithm will accept the highest request in the building, and then make subsequent stops on all busy floors on its way down in order to pick up the extra passengers waiting on the other floors. The elevator will never move upwards if there is already a passenger in the cab, but it will make extra stops to attempt to grab extra passengers. [1] In normal everyday circumstances this is an ideal situation, and can allow the elevator cab to be productive and service multiple tenants at once up to its max capacity. However, the Covid-19 pandemic has added another curveball to the elevator situation, and brouth to light a very evident issue with the current implementation of elevators algorithms. During the pandemic, it has been advised that elevator cabs should have at most one passenger. This unforseen constraint creates the situation in which an elevator will stop and open its door in order to attempt and pick up more passengers even though there is already one in the cab. This results in lots of wasted time, not only from the time spent on the door opening and closing, but also the extra time required for the elevator to slow down to a halt. Inside this paper a new Covid-19 era optimal elevator has been proposed, and through software simulation it can be verified that this algorithm has substantial improvement to other elevator algorithms that allow multiple passengers per cabin in a one-passenger per cabin situation. This simulation also has a double use as a simulator for high rise entrepreneurs and researchers to test elevator upgrades, implement algorithms and see output graphed statistics in order to be informed about their investment or any potential time improvements.

2: Problem

This section will attempt to explain the problem in a simple straightforward way. Time is being lost because there are mistakes being made by the standard elevator algorithm implementations when faced with a pandemic situation in which cab capacity is limited to a single party for health concerns. There are multiple real life typical concerns with this, such as increasing the average total elevator trip time of tenants which violates some of the most important elevator algorithm criteria.

Non-Optimal conditions: The ultimate goal of this paper is to present an optimal Elevator Algorithm for use during the Covid-19 Era, that ensures there is not time lost to unneccessarily opening the elevator cab doors. As described above, due to the Covid-19 pandemic there have been restrictions in place that limit the amount of passengers in an elevator cab to just one family unit. This is not usually a problem when the building is not busy, but during morning down rush current elevator systems and algorithms have no tolerance for dealing with these kinds of unforeseen constraints. The "Standard Elevator Algorithm" will attempt to make many extra stops along the way, that are not appropriate when there is already a passenger in the cab. The main goal of this paper is to highlight the inefficiencies created when elevator cabs are stopped unnecessarily and thus require extra door opening/closing time which in turn increases the average wait time of everyone in the system. Not only is this effect being seen in large apartment buildings during morning down rush, and evening up rush but it is also affecting large office buildings – where time is money, ultimately delaying staff from sitting at their desk and being productive.

Wasted door time: Data from 12 different apartment and high rises in Ottawa was collected to ensure that the simulator has the correct range of interchangeable elements and attributes that the simulation can perfectly model any unique building and elevator situation. These apartment buildings were selected at random from within Ottawa, and were studied on visits between October 1st 2020 to November 10th 2020. Trip distance inside the building, was also selected at random within the limits of the given building. Times were collected using a stopwatch and recorded on paper which were later transferred to an excel document provided on the github repository [6] but also shown below:

Address	Trip Distance	Door Time	Floors/sec
221 Lyon St	15 floors	6 sec	0.58 floors/sec
88 Somerset St W	10 floors	5 sec	0.25 floors/sec
234 Rideau St	19 floors	5 sec	0.76 floors/sec
96 Landry St	10 floors	8 sec	0.45 floors/sec
190 Lees Ave	17 floors	7 sec	0.61 floors/sec
170 Lees Ave	7 floors	6 sec	0.44 floors/sec
255 Bay St	12 floors	6 sec	0.55 floors/sec
100 Boteler St	5 floors	7 sec	0.33 floors/sec
111 Champagne Ave	18 floors	12 sec	0.72 floors/sec
185 Lyon St	12 floors	4 sec	0.71 floors/sec
305 Nelson St	5 floors	7 sec	0.21 floors/sec

Figure 1 [5]

Standard elevator algorithms in use across cities perform extremely poorly in a COVID-19 (1 family unit per cab), situation. Using the below figure as an illustration, it is clear that there is time being lost to elevators opening their doors unneccassarily. There are many floors that have a request inbetween the current floor after a pick up, and the destination floor. This means that as current elevator algorithms are implemented, these cabs would stop on each floor and attempt to pick up the extra passengers - which only results in wasted time because the cab is already full resulting in the current passenger apologizing and hitting the close door button again. This exact situation is all too common during the pandemic.

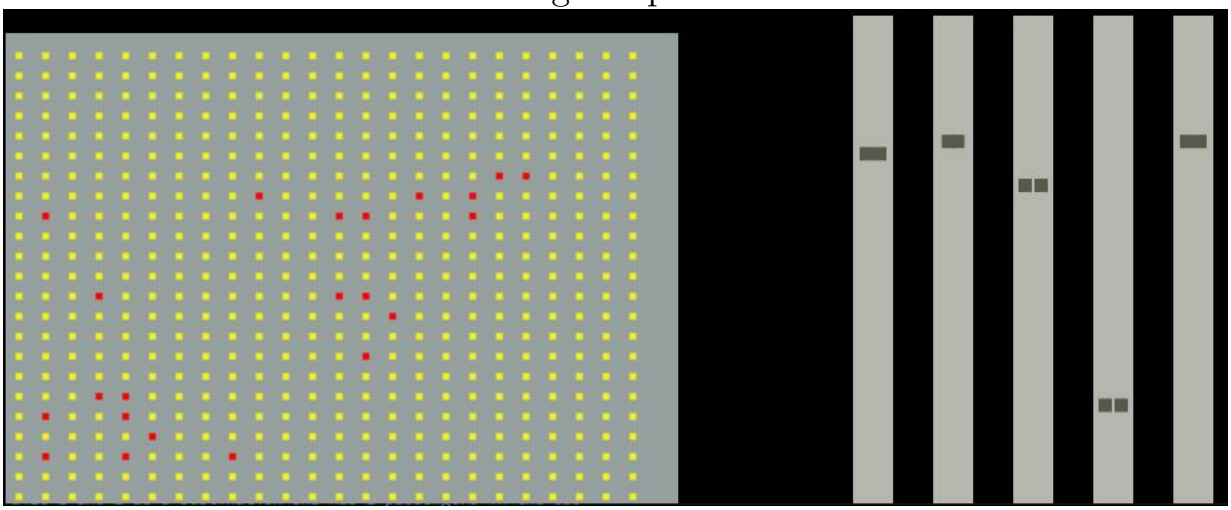


Figure 2 [6]

Each time the elevator slows down to pick up the extra passengers, it must spend on average roughly 6 seconds to open and close its doors. It the cab does this multiple times during a trip, (which is more likely when the building is busy) this will quickly increase the trip time of the passenger in the cab, but it will also simultaneously be increasing the trip time of all other tenants that are still waiting for a free elevator.

Elevator Algorithm Researcher: An individual interested in testing different elevator algorithm implementations against the "Standard" version is able to manipulate the source code available on public repository. Multiple elevator algorithm implementations are available, as well as interchangeable GUI elements. The output statitistics and histograms would also be of interest to the Researcher. There are 3 elevator algorithms provided with the simulation. Covid-19 implementation will not make extra stops in an optimal fashion. Standard implementation will make the extra stops and pick up the extra passengers, this button is to be used when you wish to test your algorithm against non-pandemic constraints. The third implementation available is Regular. This version will make the extra stops, but will not make the extra passenger pickups. This version of the algorithm is for when you wish to test an elevator algorithm against a pandemic situation.

High Rise Entrepreneur: A high rise entrepreneur may wish to increase their buildings efficiency by upgrading a few of the shafts to be quicker. Perhaps if the doors are very slow, speeding these up is a very practical way of saving a lot of time in the long run, but how can the owner be sure? Is there a possibility of adding an entirely new set of elevator shafts to another part of the building and how would this affect efficiency? What are the efficiency damages to the system when we close an elevator for move in purposes? Could we maybe close two elevators and still be okay? Currently, there are few open source resources available to see if their elevator investment will be truly worth it. This simulation software, will allow high rise owners to simulate potential changes to their elevator system and then view statistics about the results in real time and in a saved state to allow them to make more informed decisions about elevator upgrades.

3: Programming

This section outlines the programs provided in order to provide a full solution to the given problems above. The main program is the high rise elevator simulator, but some of the work has been offloaded to smaller programs for ease of use due to language compatabilities with matlab for nice histograms, as well as Pseudo-code for the proposed algorithm is presented later in the paper. Implementation source code is publically available on Github. [6]

A high rise elevator simulator: (Processing) A significant GUI portion has been created to illustrate the process. The simulation collects and outputs statistical information live to the viewer, but also writing to file for later consumption. Any unique building and elevator situation within dimensions of 30 floors x 30 rooms and up to 8 distinct elevator shafts each with their own manageable speed, door time and passenger limit. Each room can be edited and viewed individually to add/remove tenants in case the rooms under study do not have en even distribution of tenants. Found at the project repository [6] the entire simulation is contained within the sketch200916a directory, and is the combination of many different classes. Documentation and various diagrams will follow in future sections.

A command line test generator: (Java) This simulator comes equipped with a command line interface program that can be used to quickly create pseudo-random testing data sets that are provided to the simulation in order to drive passenger requests or the savvy high rise owner looking to improve their efficiency can create an exact data set that mimics what happens in their building each morning. Every single attribute of each individual el-

evator along with each individual room of the high rise are programmable and in this way a developer or high rise owner will be able to plug and play their individual situation so that they can accurately test their exact population flow in their specified simulated building. Tests are generated on the command line with 6 options concerning the output file desired. [number of floors, number of rooms, population per room, total requests to generate, minimum pause time between requests generated, maximum pause time between requests generated]. These six options will allow the developer to specify the size of building they wish to generate a test file for, as well as being able to simulate the "busy-ness" of the high rise by adjusting the maximum and minimum pause times between requests. Found at the project repository, this code can be found in the sketch200916a/TestGenerator directory name TestGenerator.java.

A simple data visualizer: (Python) This simulator comes equipped with a simple program that transforms the outputted statistical information from the simulation and transforms it into individually viewable and save-able histograms to help better understand the performance of the simulation against the given data set. Using matlab, this program will output 3 separate graphs each concerning different data points. One histogram will concern elevator time waited for the elevator to come pick the passenger up. The second graph output will concern the time waited inside the elevator cab, and the third and final graph output will concern the total trip time of each passenger. Found at the project repository, this code can be found in the sketch200916a/dataProcesser directory name dataShower.py.

4: Design Criteria and Constraints

This section outlines a checklist of provided features, as well as some real world issues that we must concern ourselves with before discussing the software design. Many things must be provided and implemented correctly by an elevator simulation. Elevator cab capacity, speed and door speed will all affect the output statistics and overall runtime of each job comletion. The software must also provide GUI features to switch elevator implementations.

There are three main metrics that will be analyzed to determine how the system is working overall. The metrics chosen concern themselves primarily with time and maintaining an enjoyable passenger experience without un-needed interruptions. In general ideally a good elevator algorithm will:

- **Minimise the amount of time that a passenger is on the elevator:** [1] Specifically in order to do this, an elevator should not travel updwards when there is already a person in the cab. An Elevator should also stop and pick up more passengers that are waiting along the way when moving in the downwards direction, only if the elevator cab is running the "Standard" version of the elevator algorithm. The Covid-19 version of the elevator algorithm will sense the passenger, and pass by such requests.
- **Minimise the amount of time a passenger waits for an elevator to pick them up:** [1] Elevator cabs should evenly and fairly work together to distribute the tasks in order to have reasonable average wait times across the entire building service request history without creating wait time outliers in the data set.

- **During a Covid-19 situation another important criteria is involved:** Eliminate all wasted time spent on unnecessary door openings in a Covid-19 Era situation. If an elevator cab makes an unneccessary stop it will increase both the above metrics for all tenants currently waiting for an elevator cab.

These 3 criteria will be the guiding force for the design and implementation of the simulation and provided elevator algorithms. Output statistics will also concern these 3 main criteria, and thus the simulation must track, save and output information concerning these metrics.

What follows is a concrete list of features provided by the source code [6]. Many of the criteria concerning the elevator cabs, the elevator shafts as well as the building dimensions and passenger attributes have been modeled after the simulation provided by Frederick Ceder and Alexandra Nordin [2]. The criteria has slightly changed due to the removal of information about energy and force being expended. The simulation being presented below is not concerned with power usage and has a more direct focus on passenger trip and wait time while maintaining an interface to the developer, which leads to slightly different priorities. Test generation and Graph generation criteria is also provided here, for a simple overlook as to what the job of each individual component is defined as.

The **Elevator Simulation** software must provide controls to and show data concerning:

- Specify building dimensions up to 30 floors x 30 rooms
- Specify amount of elevators up to 8 elevator cabs and associated shafts
- Specify one of three provided elevator algorithms to use on the provided data set
- Specify simulation speed up to 8x speed and simulation run time up to 8 hours
- Specify the cab velocity in floors per second, door speed in seconds, and cab passenger size of each individual elevator
- Specify custom or command line generated queue input file from local drive
- Render a colour coded representation of the buildings state, and of each room individually
- Render elevators moving up/down inside their respective shafts as well as opening/closing door animations.
- Render a live data screen concerning all relevant elevator attributes
- Render a live data screen concerning each individual room population when selected

The criteria that concerns the 2 smaller programs; Test generation and Graph generation follows:

The **Test Generation** command line program must provide:

- Simple command line options to quickly create large queue files to run against the simulation
- The output tests should be completely configurable with 5 options:
  - Amount of floors in building
  - Amount of rooms in building
  - Minimum wait time before generating next request in frames (15 = 1 second)
  - Maximum wait time before generating next request in frames (15 = 1 second)
  - Total amount of requests to generate (cant be more than available population)
- Request locations should ge generated Pseudo-random

The **Graph Generation** program must provide:

- Source code to transform statistic files given from the output of the Elevator Simulation into histograms.
- These histograms should be able to be individually viewed, and saveable as a .png for later consumption.
- Output a histogram concerning passenger wait time for cab before getting picked up
- Output a histogram concerning passenger wait time inside elevator cab
- Output a histogram concerning total trip time of each passenger

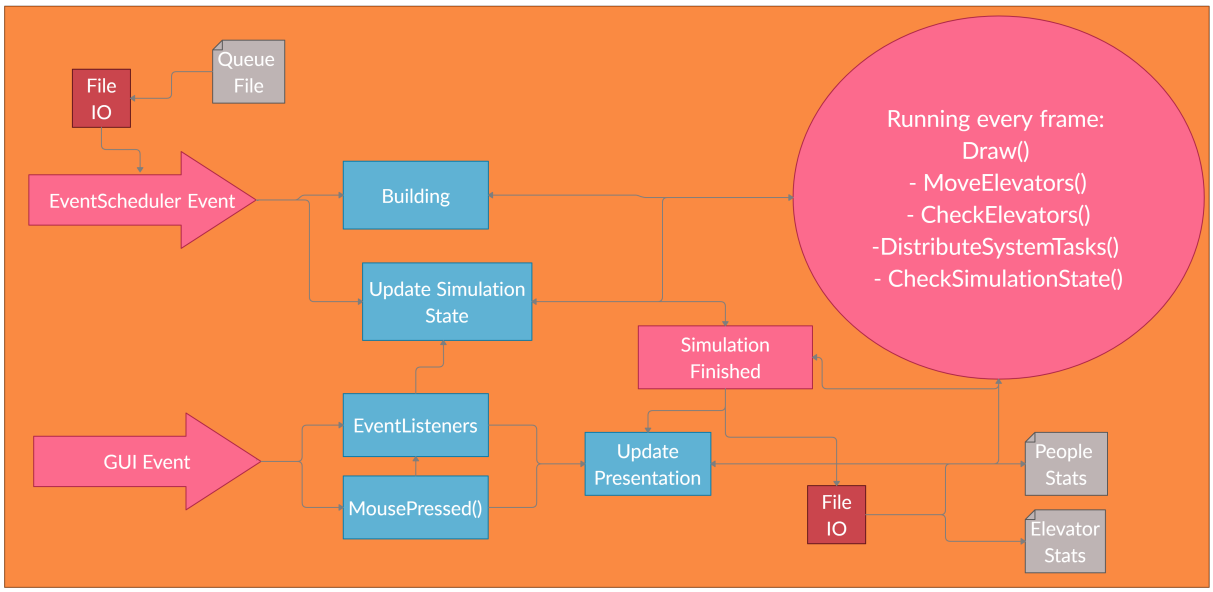
5: Architecture and Design

What follows in this section is mainly documentation concerning architecture and design choices of the elevator algorithm simulation. Included are multiple diagrams, as well as a small explanation for each.

Event Based:
Events are created by the EventScheduler from processing information given to it from a text file, which was created by the command line test generator. These events are then translated into jobs for the elevators to consume. There may be multiple elevators in play at any point, so to facilitate fast and smart distribution of these tasks to the appropriate elevator a Distributor algorithm has been added in between the EventScheduler and the Elevators themselves.

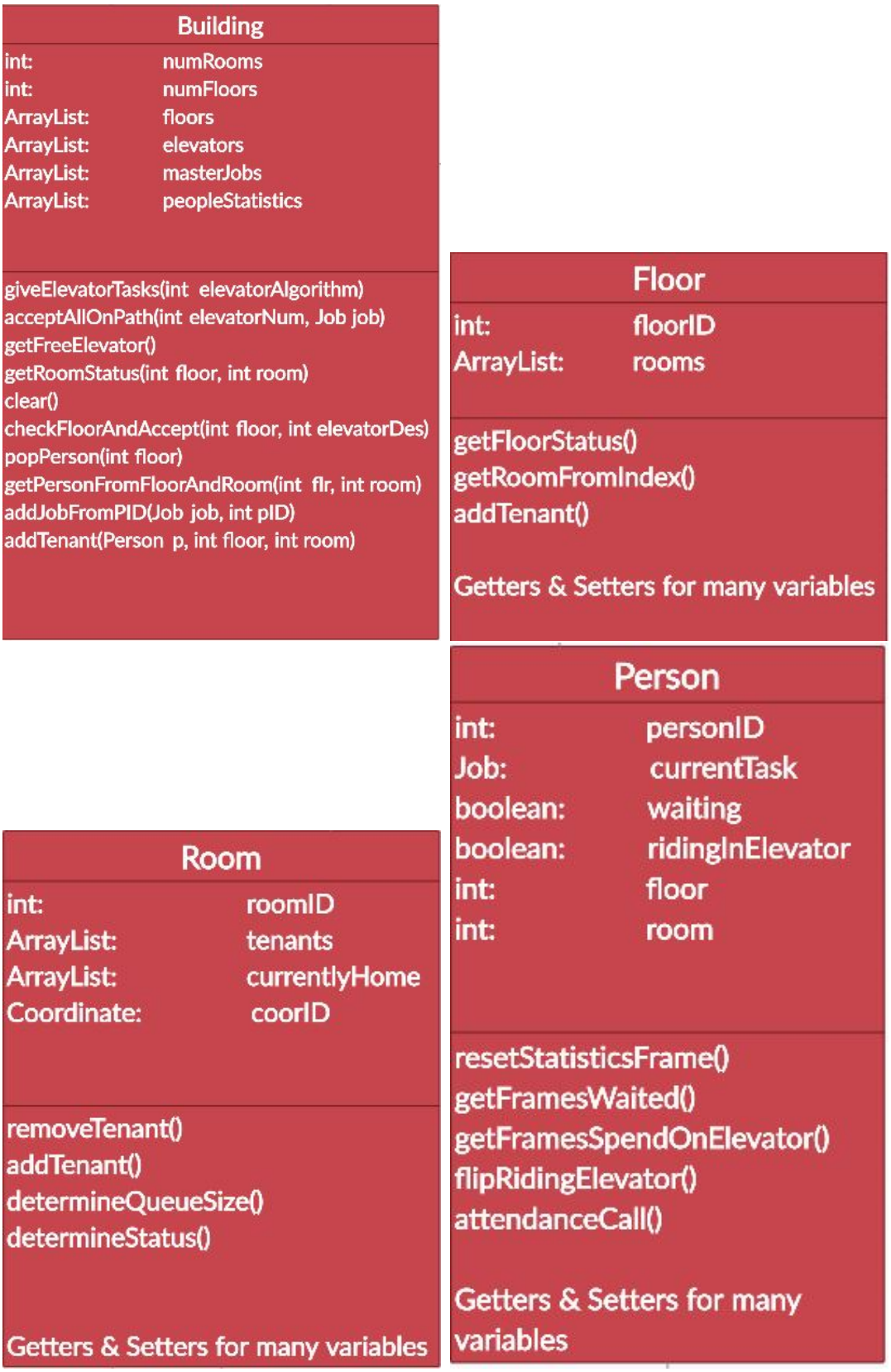
Events may also come in the form of GUI events, this could be from mouse presses from the person using the simulation, but it could be also from directly interacting with the many GUI elements like the sliders or buttons.

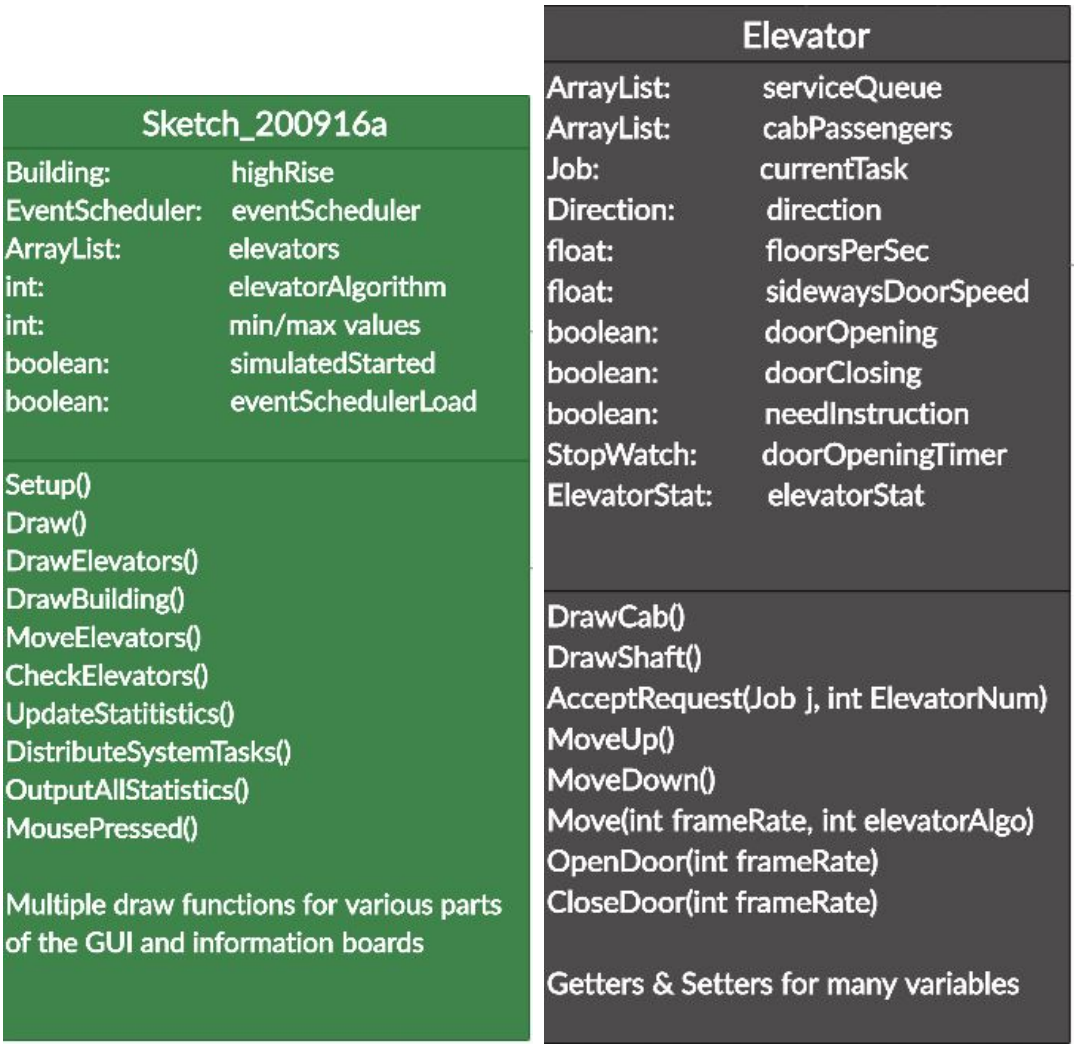
Events are processed in the Draw method which is run every single frame. It contains important subroutines that are also run which may or may not create even further events based on the simulation state. Events are also processed within many different small functions that are being conceptualized by the Event Listeners, Update Presentation and Update Simulation State boxes. Together these events, combine into a single simulation state, which is eventually finished ultimately outputting two important files containing the statistics of the past run with help from File IO on both the beginning and end of the program flow.



Language based:

Processing is "Java, with added functionality for simple graphics". As such, OOP has been employed. Basic UML outlines follow for the various important components of the elevator simulator design follow. Components are color coded for general proximity in relationship. There are a few less important classes that are omitted here, for a look at the full UML diagram it can be found in resource files contained in the project repository. [6]


Building
int: numRooms
int: numFloors
ArrayList: floors
ArrayList: elevators
ArrayList: masterJobs
ArrayList: peopleStatistics
giveElevatorTasks(int elevatorAlgorithm)
acceptAllOnPath(int elevatorNum, Job job)
getFreeElevator()
getRoomStatus(int floor, int room)
clear()
checkFloorAndAccept(int floor, int elevatorDes)
popPerson(int floor)
getPersonFromFloorAndRoom(int flr, int room)
addJobFromPID(Job job, int pID)
addTenant(Person p, int floor, int room)
Floor
int: floorID
ArrayList: rooms
getFloorStatus()
getRoomFromIndex()
addTenant()
Getters & Setters for many variables
Person
int: personID
Job: currentTask
boolean: waiting
boolean: ridingInElevator
int: floor
int: room
resetStatisticsFrame()
getFramesWaited()
getFramesSpendOnElevator()
flipRidingElevator()
attendanceCall()
Getters & Setters for many variables
Room
int: roomID
ArrayList: tenants
ArrayList: currentlyHome
Coordinate: coorID
removeTenant()
addTenant()
determineQueueSize()
determineStatus()
Getters & Setters for many variables



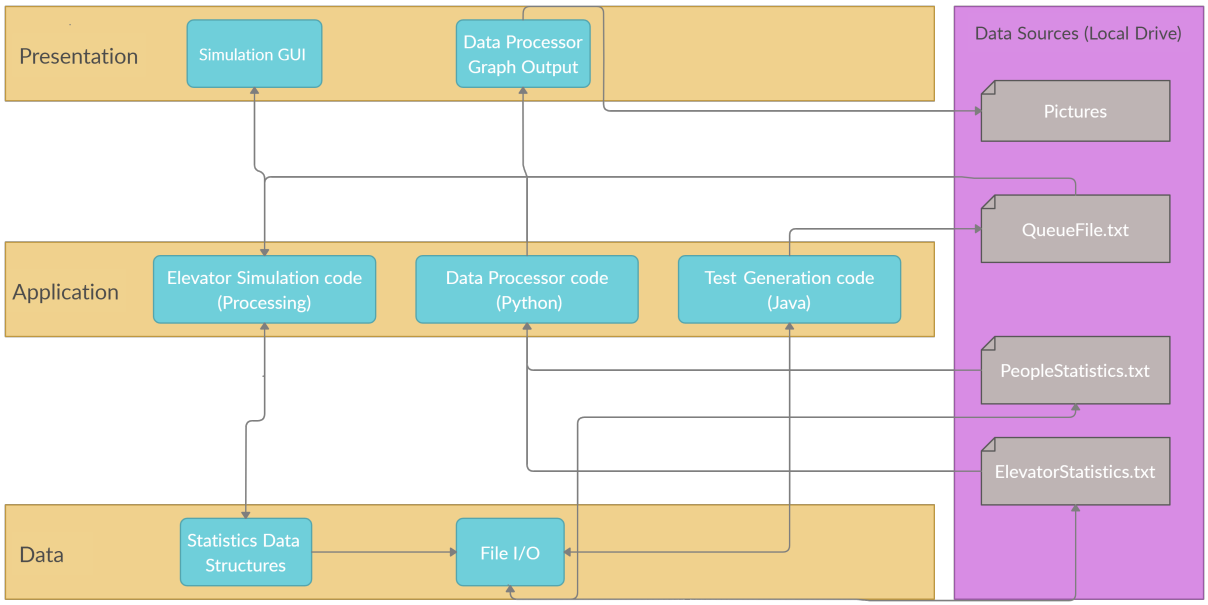
Main program and Subroutines design is employed as follows: Main program is Draw(), from within the main sketch file. The main program calls 3 important subroutines, moveElevators(), checkElevators(), and distributeSystemTasks(). Based on the developer options specified on the GUI each of these functions will work slightly differently in order to provide access to different elevator algorithm types.

The sub-routine moveElevators() calls the logic concerning updating the cab position or animating the doors opening and closing. Function checkElevators() contains the logic concerning people getting on and off the elevator

cabs. Lastly distributeSystemTasks() will ensure all elevators available are sharing tasks if needed, as well as picking up extra passengers along the way.

Layered Architecture:

Logically the program is separated into Presentation, Application and Data tier each doing their own tasks. Each frame, the presentation subroutines are run showing all the graphical components to the user. The application portion consists of the Main routine and all other subroutines that are not involved with graphics or file manipulation. Data in exists from files, and variables being adjusted on the GUI. Data coming out exists within exported text format, showing statistics concerning the population flow of the simulated run. This data can then be used as input to a provided python program that crunches this data and creates graphs. Given two data files that are the output statistic files from the simulation software, it will produce some useful and saveable graphs to a local drive if you wish.



6: Mathematical Definition

This section gives a better explanation as to why a more optimal elevator algorithm is needed given COVID-19 circumstances. What follows is declaration of variables in order to create equations that will be used to conduct proofs and reason about the runtime of different elevator algorithms. This section will attempt to demonstrate why standard elevator algorithms are a poor choice in a world under pandemic mainly due to wasting time unnecessarily opening its door when the passenger cab is already occupied.

P: The probability that the elevator cab is going to stop on a floor to attempt and fulfill a request, even if there is already one family unit in the cab (this is the situation that is currently installed in high rises). (VARIABLE) This is a variable that is essentially impossible to actually determine but we can reason about its value to make predictions. When the building is in a "busy" state, P will be larger than when than when the building is "slow".

D: The distance of the given elevator trip. (EG: someone going from their 10th floor apartment to ground floor, D=10). (CONSTANT) In the worst case the elevator is at ground floor, and must travel 2D.

R: The time that the door takes (in seconds) to open and close. (CONSTANT)

V: The speed of the elevator cab (in floors/second) (CONSTANT)

T.1: We can define the best case time of an elevator trip as:

$$2DV + R$$

T.2: We can define the expected amount of extra time spent opening the door:

$$RDP$$

T.3: We can define the Expected total trip time:

$$2DV + R + RDP$$

These equations will be used in the next section to argue about passenger trip time in various scenarios. In a hyper realistic simulation there are some other concerns such as acceleration/deceleration though section 4 explains why this is not that important for the simulation being provided and outlines alternate criteria that is not concerned with physics and energy consumption.

7: Mathematical Solution

This section will describe, analyze and prove an optimal Covid-19 elevator algorithm that successfully makes P = 0. In doing so we will obtain best case running time on every service request for a Covid-19 pandemic situation where only one family unit should be in a cab at any point.

Making P = 0: It is possible to make P=0 by designing an algorithm that does not open its doors for extra passengers if there is already one in the cab. In simulation this is easy; however in the real world it may be slightly harder to tell if a cab is truly empty or populated. Laser technology reading in/out passes through the door or having an AI reading feedback from the elevator camera to determine if there is a passenger or not are some solutions to this real world problem. I believe that the second idea is the most feasible, given the circumstances as well as the fact that cameras are already installed into most elevators, doing this would be a relatively easy upgrade in order to ensure efficiency of your elevator shafts.

We can ensure that P=0, by never opening the cab doors until the current Job is completed. In this way each elevator should only ever have one active Job. In this way we will be able to eliminate the second part of the run time equation **T.3:** ultimately saving RDP time. There are many other things that an elevator algorithm may wish to do, but since it has been established that there will only ever be one passenger in the cab this simplifies a great deal of other typical elevator concerns.

We will show the code for this algorithm soon, but for now we can simply assume that a smart algorithm exists that can make P = 0. We will make this assumption to simply continue with the math, and see if this change in implementation results in a faster expected passenger trip time. If we were to implement an elevator algorithm with P = 0, expected total time of the trip using this new algorithm is now:

$$2DV + R + RDP$$

$$2DV + R + RD * 0$$

$$2DV + R$$

This runtime outcome is great because it is shown in Section 6 that this new runtime is actually equal to the best case run time! **T.1:** . Therefore an algorithm that does not waste any time opening its door unneccessarily will always run in best case time. Since the "assumed" algorithm doesn't waste anytime opening its doors unless it needs to, the runtime will always best case. Now it has been shown that if an algorithm exists and is able to make P = 0, the elevator system will save **T.2:** time. I will present pseudo code that provides this solution, after discussing some other typical elevator concerns.

Making judgements about job distribution order: An elevator that operates in such a fashion described above does not need to make judgements about what way is the fastest to claim tasks – because it will not have the opportunity to pick up extra passengers and save on up/down time by claiming a high floor request and then subsequently grabbing everyone else

on the way down. The regular elevator algorithm benefits from grabbing the higher level requests and then subsequently grabbing the rest of the waiting tenants on the way down. This will not be the case when the Covid-19 algorithm is in use and is detailed below.

We have shown that the total run time for any single task is a constant 2DV + R. **T.1:** For this algorithm to be fair it must operate its service queue in a FIFO manner, otherwise there is a chance that one passenger may have to wait unreasonably much longer than the others. The Covid-19 elevator algorithm will not allow multiple passengers, and thus does not benefit from grabbing higher level requests first.

Making judgements about extra passenger pickups: An elevator that operates regularily should not travel upwards to grab more passengers if the current destination is in the downward direction. Therefore, the standard algorithm should only be picking up extra passengers if the passenger request is going in the same direction as the passengers that are currently in the cab. In the simulation provided, a standard elevator implementation will attempt to grab extra passengers when it is travelling in the downwards direction, after grabbing the highest request possible. The COVID-19 implementation does not have to concern itself with this either, it will just simply never make a stop if it already has a passenger and has yet to fulfill the passengers job.

8: Pseudocode

Distribution Algorithm: The distribution algorithm will give Jobs to any elevator that is currently not doing anything arbitrarily, and allow the other elevators to ignore this request. If all elevators are busy, it will continue to look at the elevators until there is one not doing anything. At that point it will immediately give that free elevator the next request in FIFO order. The distribution algorithm will be responsible for maintaining the FIFO order of the passenger requests, and will continue giving requests, until there are no more in the building. This algorithm blocks until it senses that there are more requests in the building.

Covid-19 Job Distribution elevator algorithm Pseudo-Code – runs every frame of the simulation:

Input:
ArrayList of Elevator objects: Elevators. Contains all elevators objects in the building
ArrayList of Job Objects: Queue. Contains all jobs currently in the building
Job Object: Job. Contains either a default job to signify no new job added this frame, or the actual Job that has been added this frame.

i = 0

1. Queue.addAtEnd(Job)
2. While(Queue.size() $\geq$ 1):
3. i = i + 1
4. For each elevator in Elevators:
5. IF: this elevator doesn't have a job
6. give it the first Job in the Queue, remove Job from Queue
7. break to outer loop in order to recheck if Queue.size() $\geq$ 1
8. ELSE: continue

Output: Void

Consequences: After running this algorithm, anywhere from 0 to the total number of elevators will have potentially had a job added to their service queue.

Cab movement algorithm: The cab movement algorithm will call the appropriate movement functions based on the status of its 1 current active task. By reading and issuing updates to this Job's attributes, the elevator will be to orient itself and call appropriate methods in order to do its doors and pick/drop passengers.

Input:

Elevator object: Elevator.

Job Object: Job. The job that the elevator is currently satisfying

Covid-19 Cab Movement elevator algorithm Pseudo-Code – runs every frame of the simulation:

1. IF (there is an active Job):
2. IF (the active job is not picked up yet):
3. IF (current floor > active job pickup floor):
4. moveDown()
5. ELSE IF (current floor < active job pickup floor):
6. moveUp()
7. ELSE IF (current floor == active job pickup floor):
8. openAndCloseDoors()
9. pickupPassenger()
10. ELSE IF (the active job is picked up):
11. IF(current floor > active job destination floor):
12. moveDown()
13. ELSE IF(current floor < active job destination floor):
14. moveUp()
15. ELSE IF (current floor == active job destination floor):
16. openAndCloseDoors()
17. dropPassenger() - completes job (removes active Job and terminates)

Output: Void

Consequences: The elevator will either move up, move down or is somewhere in the process of opening/closing its doors or picking up or dropping off a passenger

MoveDown() – floor = floor - 1

MoveUp() – floor = floor + 1

9: Proofs

This section presents two proofs for the algorithms presented above. This section will prove by induction that the distribution algorithm for the covid-19 elevator algorithm correctly implements a FIFO queue scheduling pattern. This section also provides a proof by cases that the cab movement algorithm will correctly complete a job for one single passenger. In this way, once both of these have been proved, the correctness and fact that they terminate will be able to prove the overall correctness of the proposed Covid-19 elevator algorithm. [4][5]

I will prove the Job Distribution Algorithm correctly implements a FIFO queue scheduling pattern.

I will do a proof by Induction on N, which is the initial size of Job Queue
Base case: N = 0, 1 are trivial as any output of a Queue with size 0,1 will be a FIFO output no matter what.

Base case is N = 2.
Given J = [x0, x1] we expect that the order the Jobs are given to elevators is equal to the proper FIFO output set O = [x0, x1].

i = 1
We can see Job x0 is given to the first free elevator on line 5, reducing J to [x1] (ADD x0 to the output set)

i = 2
We can see Job x1 is then given to the next free elevator on line 5, reducing J to [ ] (ADD x1 to the output set)
We can see that this algorithm has now terminated and the output set is = [x0, x1], with i = 2, the size of the input set. Therefore this is a FIFO pattern.

Induction Hypothesis: Assume that the Distribution algorithm implements a FIFO pattern for all sizes of set J from [0, 1 ... K-1], and our indicator variable i is equal to the size of the input set when the algorithm terminates.

Induction Step: Want to prove that if the Distribution algorithm implements a FIFO pattern for all sizes of J from [0, 1 ... K-1] then it must also implement a FIFO pattern for a set J of size K.

N = K.
Set J is [x0, x1, x2 ... xN]. Lets reduce this set by one element by removing Xn. Since the size of J is now K-1, we can apply the inductive hypothesis. From the inductive hypothesis we know that the set with element Xn removed is returned in a FIFO pattern.

Since J [x0, x1 ..... xN-1] is returned as a proper FIFO output set. When this algorithm stops on a set with size N-1, i will be equal to N-1 as well. By returning the element [Xn] to the set J, we know that the first N-1 elements will be output correctly in FIFO pattern.

On line 6 the Queue is reduced by size 1 each step, and since we know i = N-1 (our step counter), we know that the size of the Queue is now 1, which contains only the element [Xn] that was on the end. (We know this because [x0, x1, x2 ... xn-1] / [x0, x1, x2 .... Xn] = [Xn]).

The output set is currently a proper FIFO set, and we can see on the next iteration i = N, the element Xn is appended to output set on the end, which is still in proper FIFO order.

We can now see that since i = the size of J this algorithm now terminates.
Therefore the distribution algorithm correctly implements a FIFO queue for all sizes of input set J.

I will prove the cab movement algorithm correctly completes a single job for an elevator cab.
Proof by cases
Case 1: s Given a Job with pickup X, and destination Y that starts with not having been picked up. We know currentFloor, X, Y are elements of the positive whole numbers. [0, 1, 2, 3 ,...]

. **Case 1 A:** If we enter line 3: we know currentFloor > X. We know that moveDown() has made currentFloor closer in value to X, by reducing currentFloor by 1.
. We will then run line 3 again, depending on the value of X, we may

or may not run moveDown() again, but we can be certain that line 5 will never be run on subsequent iterations. Since moveDown() is correctly making currentFloor closer to X, we know that line 7 will eventually be true, which correctly picks up the passengers – then terminates

. **Case 1 B:** If we enter line 5: we know currentFloor $<$ X. We know that moveUp() has made currentFloor closer in value to X, by increasing currentFloor by 1.

. We will then run line 5 again, depending on the value of X, we may or may not run moveUp() again, but we can be certain that line 3 will never be run on subsequent iterations. Since moveUp() is correctly making currentFloor closer to X, we know that line 7 will eventually be true, which correctly picks up the passengers – then terminates.

. **Case 1 C:** If we enter line 7: we know currentFloor == X, thus we will immediately do the doors and pickup the passengers – then terminates.

Case 2: Given a Job with pickup X, and destination Y that starts having been picked up. We know currentFloor, X, Y are elements of the positive whole numbers. [0, 1, 2, 3 ...]

. **Case 2 A:** If we enter line 11: we know currentFloor $>$ Y. We know that moveDown() has made currentFloor closer in value to Y, by reducing currentFloor by 1.

. We will then run line 11 again, depending on the value of Y, we may or may not run moveDown() again, but we can be certain that line 13 will never be run on subsequent iterations. Since moveDown() is correctly mak-

ing currentFloor closer to Y, we know that line 15 will eventually be true, which correctly drops off the passengers – then terminates.

. **Case 2 B:** If we enter line 13: we know currentFloor $<$ Y. We know that moveUp() has made currentFloor closer in value to Y, by increasing currentFloor by 1.

. We will then run line 13 again, depending on the value of Y, we may or may not run moveUp() again, but we can be certain that line 11 will never be run on subsequent iterations. Since moveUp() is correctly making currentFloor closer to Y, we know that line 15 will eventually be true, which correctly picks up the passengers – then terminates.

. **Case 2 C:** If we enter line 15: we know currentFloor == Y, thus we will immediately do the doors and pickup the passengers – then terminates.

In all of Case 1 – The Job not being picked up yet - we can see that eventually line 7 will be run, correctly picking up the passengers. Note that the state the elevator and Job is left in after this case – is exactly where Case 2 picks up. We can see that in all of Case 2 – line 15 is eventually run which correctly drops off the passengers and terminates. We can see that all Cases 1, after picking up the passengers, the Job is now in a state described by Case 2 and all Case 2 jobs are completed successfully. Therefore the entire system is correct and terminates.

Correctness of the proposed Covid-19 Elevator Algorithm:

This algorithm is correct as long as all passengers are picked up and dropped off on the appropriate floor in FIFO order, while the number of times opened/closed the doors = number of Jobs completed. In this way, we know that the algorithm has run in best case time 100 percent of the time, solving the given problem.

The Covid-19 Elevator Algorithm correctness follows from the correctness of the implementation of FIFO from within the Distribution algorithm as well as the correctness of the Cab Movement algorithm.
Since the distribution algorithm has been shown to correctly distribute a set J of jobs of any size N to E elevators, and the Cab movement algorithm has been shown to successfully complete a Job Ji for elevator Ei as long as elevator Ei has a job – it follows that together these algorithms correctly distribute and complete all Jobs properly for any input set J.

10: Results

To re-iterate what was discusssed: First there was equations established to calculate passenger trip time. Next, an algorithm that performs perfectly in a pandemic (one family per elevator cab situation) was presented to make a promise that there would be no wasted time opening and closing doors unnecessarily. These passenger trip time equations were then manipulated by using that fact that P = 0 in this perfect case to eliminate the variable portion of the equation.

After making these changes, the runtime became a calculable constant and shown to be running in best case time **T 1.1** , on every single passenger request. No matter how busy the building is, and how many stops a regular elevator algorithm would make, the COVID-19 algorithm successfully runs in constant best case time for every single passenger request.

Test queue generation files will be created using the TestGenerator program provided by the simulation. There will be a "slow" version, and a "fast" version of queue file created for each building size of 10 floors x 10 rooms, 15 floors x 15 rooms, and 25 floors x 25 rooms. These exact queue files are available in the repository [6].

Output graphs shown in the report have been created by the Graph Generator program provided by the simulation. A directory of these graphs are available on the repository landing page as well.
The results section will compare 3 different scenarios. The same queue files

for each building size will be tested against:

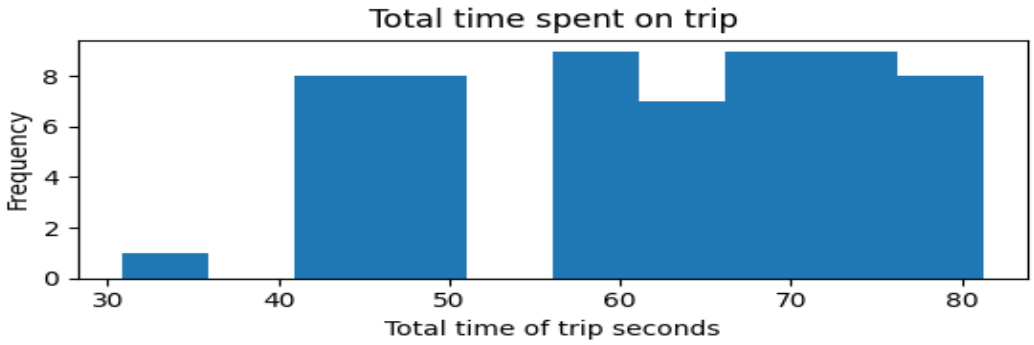
- COVID-19 Algorithm - does not make any extra stops, one passenger at a time
- Regular Algorithm - makes extra stops but does not pick the passengers up - this is our current state of affairs (pandemic situation)
- Standard Algorithm - makes extra stops and picks the extra passengers up - (non-pandemic situation)

Testing all 3 of these scenarios is important because it will allow us to determine whether or not the COVID-19 algorithm is performing better than the Regular algorithm. If we are able to see that the COVID-19 algorithm is in fact an improvement to the current situation, we then need to determine how much slower it is performing than the Standard algorithm. It is presumed that the Standard algorithm will be the most efficient, because this algorithm runs in a situation where there is no pandemic, and can pick up multiple passengers without worry. The extra set of statitistics will help to determine this, but also therefore we know that when returning to regular circumstances and additional health measures are no longer of concern it should be stated that picking up the extra passengers is of course more efficient than passing by these requests. [3]

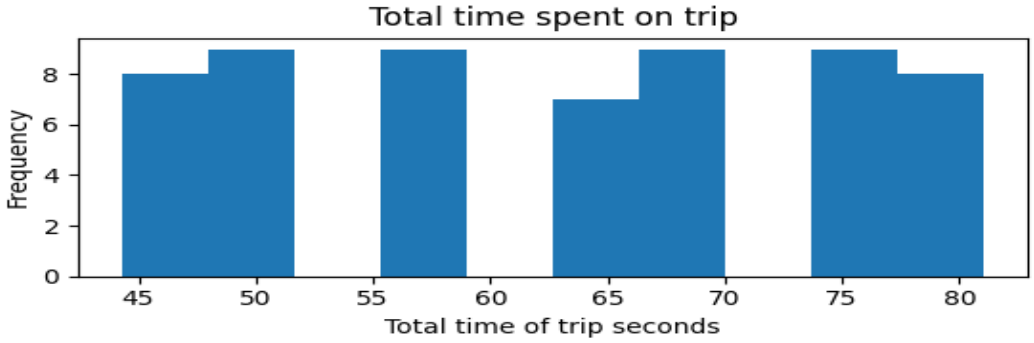
We will be examining the various algorithms underneath different workflows. It is hypothesized that during a "slow" off peak hours algorithms will operate in similar average trip times. The main problem is when the building is "busy", and there creates a point in which a lot of time is wasted to door openings.

Small Building - 10 floors x 10 rooms - Slow building state

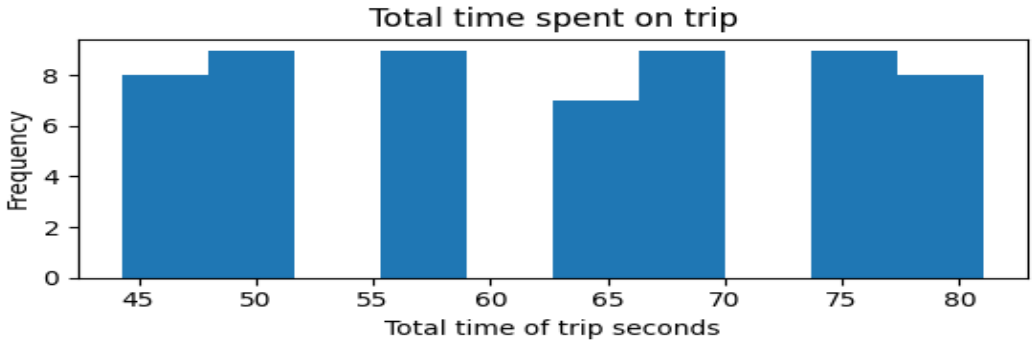
COVID-19



Regular



Standard



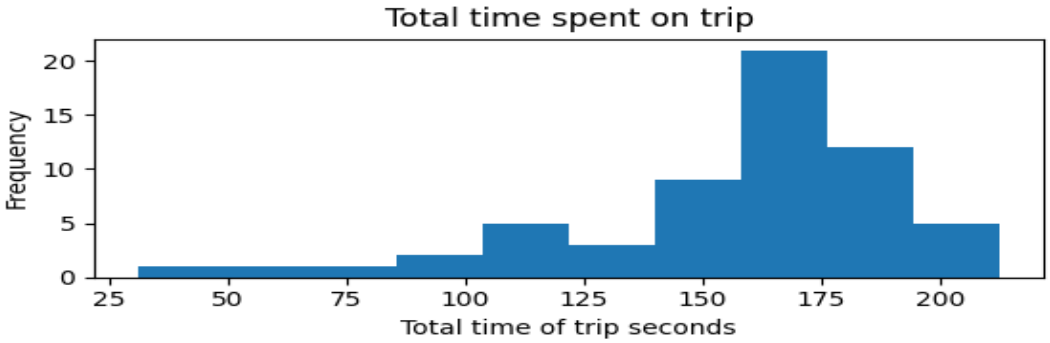
It is hypothesized that the COVID-19 algorithm may not have too large of an effect in smaller buildings, and during off peak time periods. The COVID-19's elevator algorithm proficiency comes from the fact that it doesn't stop unneccessarily, but in a situation where the building is small and slow, then the probability P (as discussed in section 6) is low and perhaps even resulting in zero extra openings if there was never a busy floor inbetween the original pickup and destination. As well, since both Regular and Standard algorithms work identical, except for the fact that the Regular does not pick up the extra passengers when it makes a stop (current pandemic situation), it can also be noted that if P is low that the output graphs for total passenger time should be very similar, or identical in the case where $P = 0$.

Here we can see what is expected to be happenning. Since this is a "slow" case, and the building is small all algorithms seem to be working to about the same efficiency. Since Standard and Regular output graphs are identical, it is clear that this queuing schedule provided for this case did not create the scenario in which doors are unneccessarily opened.

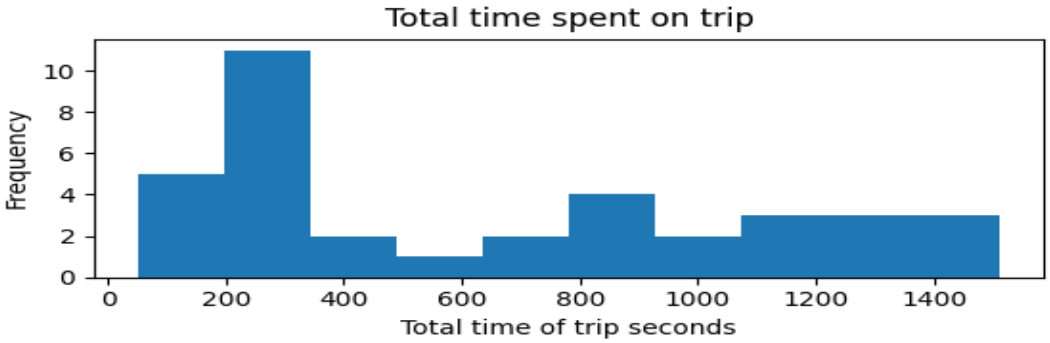
Based on these graphs so far, the COVID-19 algorithm can successfully keep up with the Standard and Regular implementations for small and slow buildings. Though it does not offer any significant time advantages to the Regular algorithm, it is not any slower. To determine if the COVID-19 algorithm is benefical to small buildings it is important to continue and test the same building size under a heavier workload.

Small Building - 10 floors x 10 rooms - Busy building state

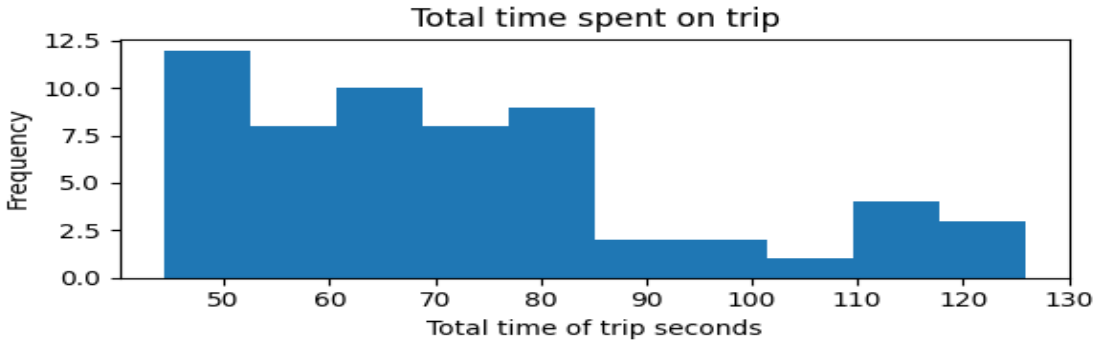
COVID-19



Regular



Standard



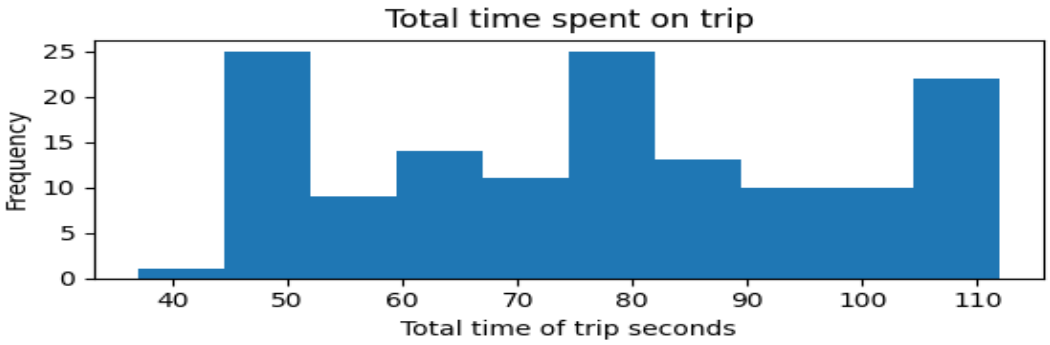
This information provided by this 10x10 busy simulation is much more interesting than the slow case! Here we can begin to see where the Regular algorithm completely fails to satisfy its basic requirements, and we can begin to see that the Standard algorithm is in fact the fastest (due to the fact that it is operating under the assumption there is no pandemic). The queue file provided for this simulation was quite aggressive, and would be an extreme in a realistic setting. However, since the COVID-19 algorithm can handle it, I wanted to show some data where the Regular implementation completely fails its objective of responding to all requests in a timely manner.

When the building is busy, P (as describe in section 6), is extremely high. This results in **T.2** RPD time being wasted on every single job that the elevator is doing, not only does this increase the wait time of the passenger in the cab, but it also increases the wait time of all tenants currently waiting in the building because its going to take longer for the elevator to be free and come pick them up.

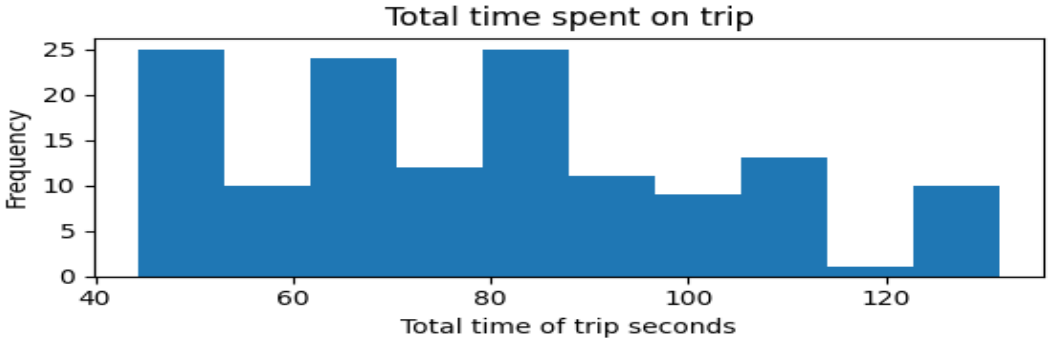
Taking a look at the output graph for the Regular algorithm, you can see that because of the pandemic situation, this has created a scenario where tenants can be waiting over 10 minutes for an elevator to come get them. In a real-world scenario, this is obviously ridiculous and it is most likely that the majority of people that have to wait this long would end up taking the stairs. Having your tenants take the stairs is a solution to the problem, but not a very good one when the COVID-19 implementation can successfully deal with this same workflow in run time very similar to a the most optimal algorithm running without any extra constraints.

Medium Building - 15 floors x 15 rooms - Slow building state

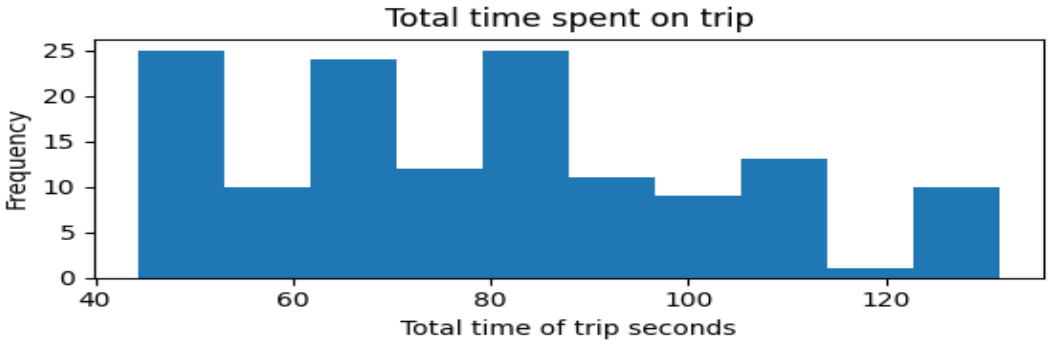
COVID-19



Regular



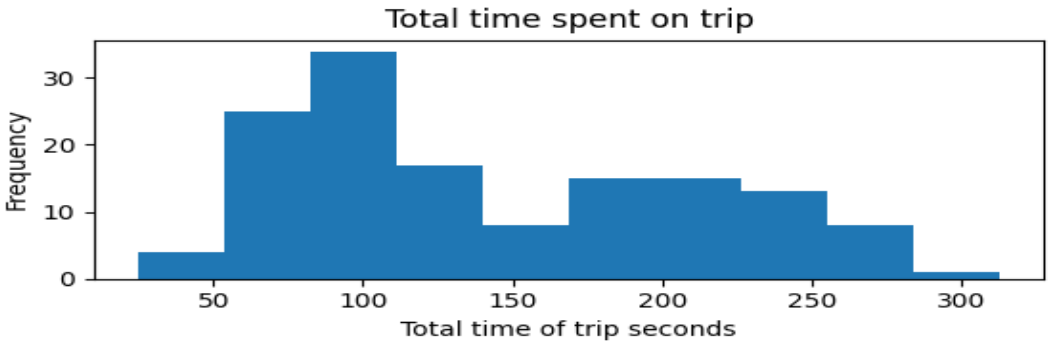
Standard



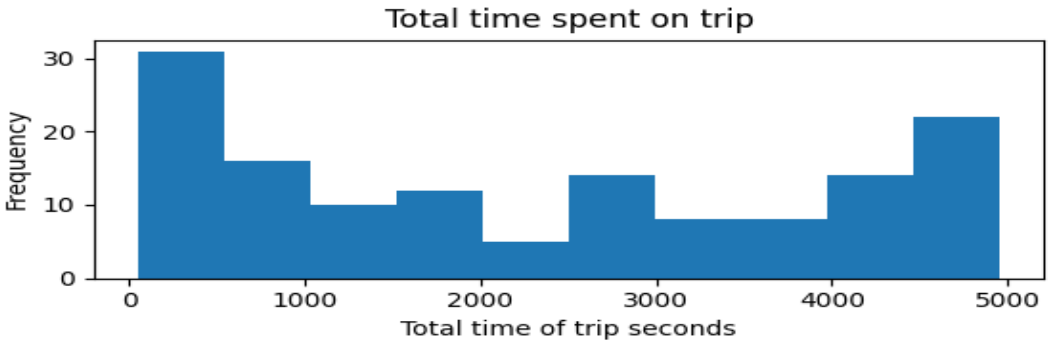
Here we have another slow state, but the building is slightly larger and hence busier. As the building grows, typically P (as described in section 6) grows as well, because there is a higher expected trip distance, which results in a higher probability of being stopped along the way. Despite the building being larger, we are still in off-peak hours, and thus variation between the run times of the algorithms is pretty non-substantia or non-existant. However, having this result is also important because it reminds us that the COVID-19 algorithm is no less efficient than the most optimal one running without constraints. In the same fashion as discussed previously for the 10x10 building, here in the 15x15 situation the output graphs for all 3 algorithms are similar or identical for this because of P being close to or equal to 0. When this is the case, these algorithms will run in very similar time, and this will most likely continue to be the trend during off-peak hours in different building sizes.

Medium Building - 15 floors x 15 rooms - Busy building state

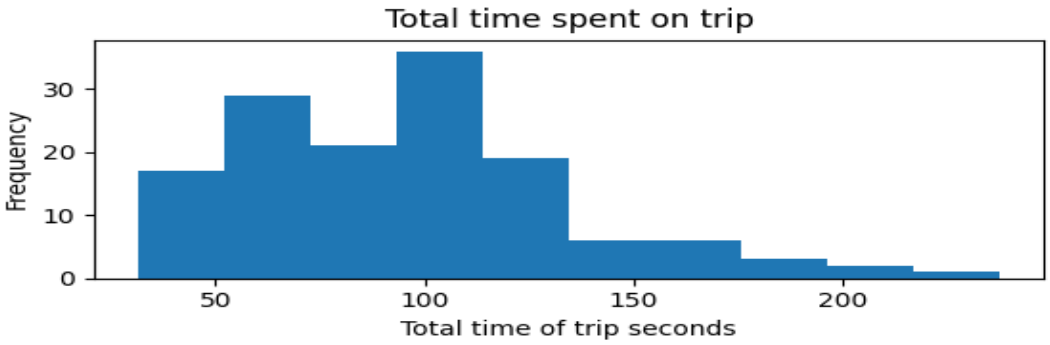
COVID-19



Regular



Standard



Once again the "busy" simulation provides a lot more information of note for this study. First the most obvious observation is that the Regular algorithm (our current pandemic situation with wasted door openings) has completly failed resulting in absurb passenger trip times. Again, in the real world this would result in many tenants using the stairs if they are able, which completely defeats the purpose of having elevators to begin with. Since the building is so busy, P is large and stopping multiple times on a single trip is the common scenario. As the average trip time starts to become slower due to the building becoming more and more backed up, resulting in more door openings, and resulting in higher wait times for everyone else, the algorithm slows to a point where it is completly unusable.

Here we can begin to see the true proficiency and power of the Standard elevator algorithm. Since it is operating in a non-pandemic situation, it can benefit from picking up extra passengers on the way down. Since this building is so busy, it can benefit from this fact many times throughout the simulation and should have a much lower run time than other algorithms which it does.

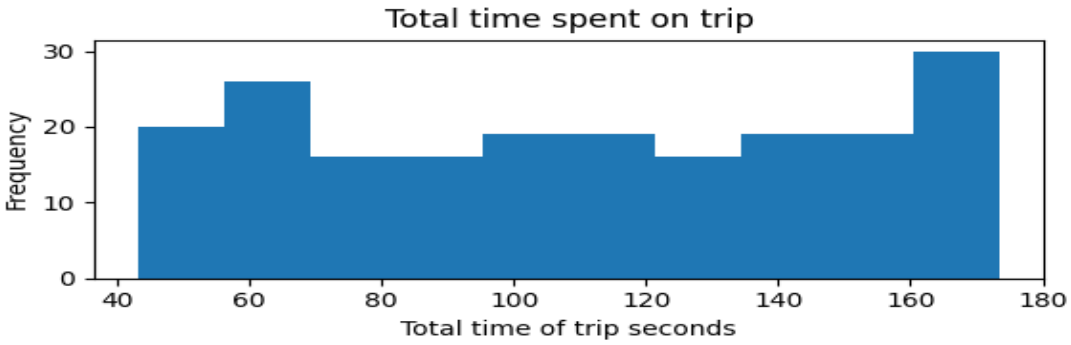
It has been noted that the Standard algorithm is very efficient and operates under best case constraints (multiple passengers), so any algorithm that can come close to its performance must be pretty decent. The COVID-19 algorithm is operating under extra constraints (one passenger at at time), and is still able to produce wait time distributions only slightly slower than that of Standard.

The Regular algorithm is clearly not a good choice of implementation during a pandemic, though it is the one in use. Even in small buildings, such as the 10x10 and 15x15 buildings analyzed above when the workflow becomes busy it will quickly clog up the entire building. If it were possible to switch elevator algorithms during the pandemic, the COVID-19 algorithm is a clear step above what is currently being used.

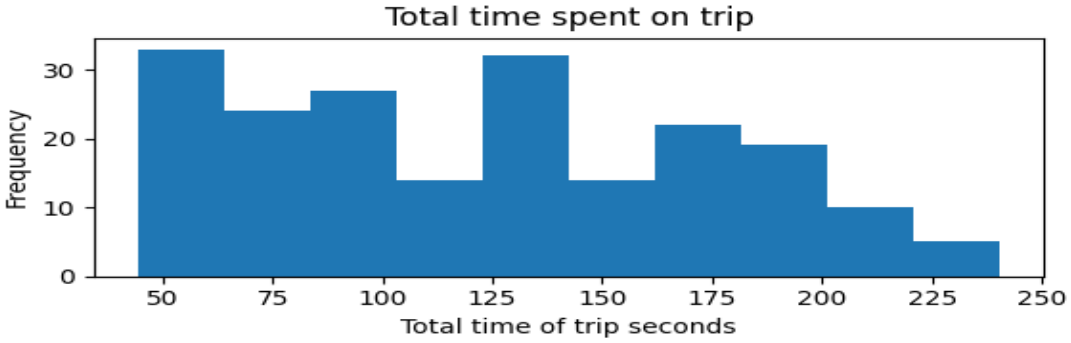
To be more certain of the results shown above, a study of a larger 25x25 building follows in both a slow and busy state.

Large Building - 25 floors x 25 rooms - Slow building state

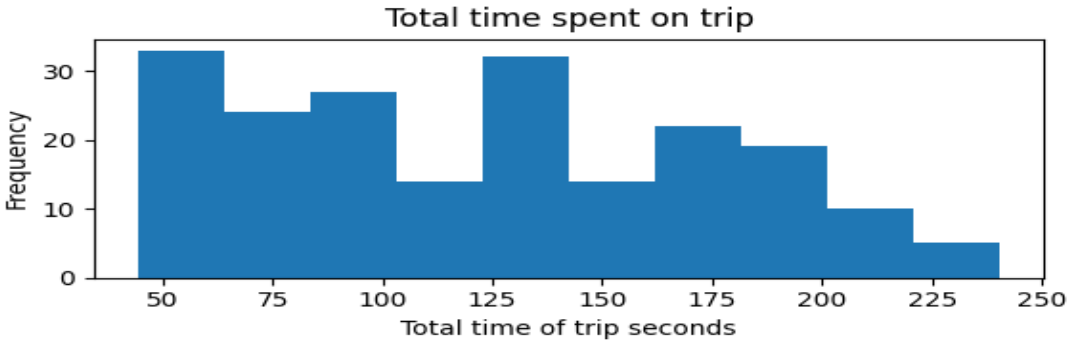
COVID-19



Regular



Standard

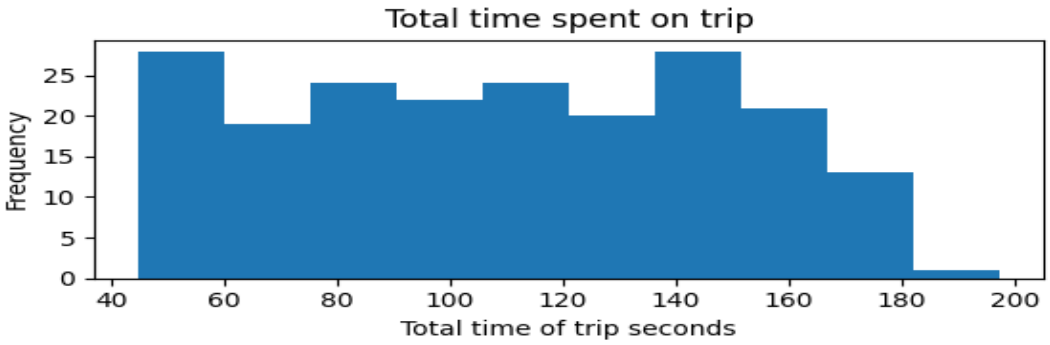


Off peak hours never seem to reveal too much information about the differences between implementations. Even though the building is large, there is still not enough cross interaction amongst the elevators to create a large amount of unneccessary door openings. All algorithms are operating in similar time here, just as the small 10x10 and medium 15x15 buildings did as well.

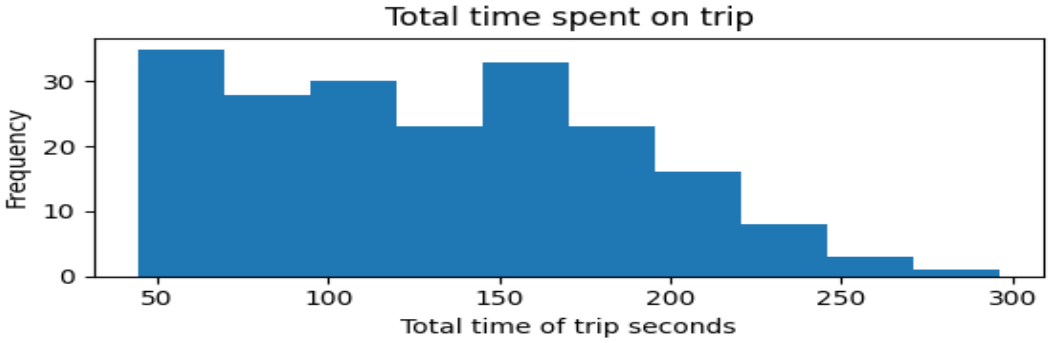
After checking a small, medium and large building in a slow state, it is clear that the COVID-19 algorithm is not more efficient, nor any slower than other algorithms provided. Now, it is time to run a busy on-peak hours simulation against this large 25x25 building.

Large Building - 25 floors x 25 rooms - Busy building state

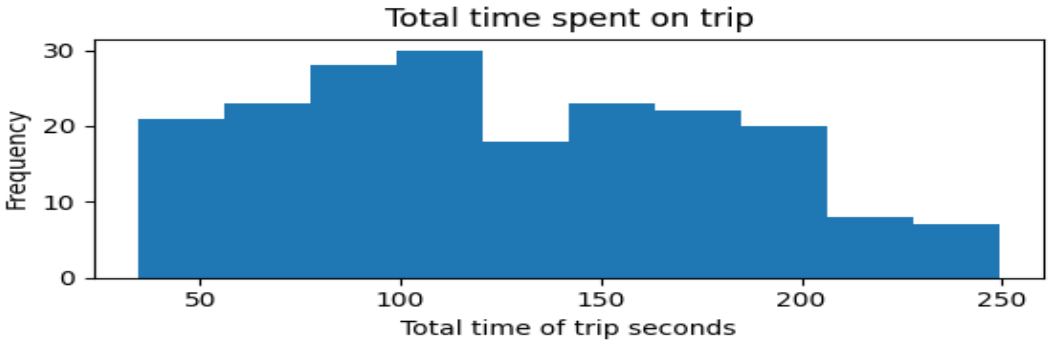
COVID-19



Regular



Standard



Surprisingly, unlike other scenarios there is not a huge difference between the COVID-19 and Regular implementations. This is because the queue file used is a little more reasonable, and this building had access to 6 elevators. Despite this, you can see that the Regular implementation produces much more 200+ second trip times than that of the COVID-19 algorithm.

It is also interesting to note here that the Standard algorithm is also producing more 200+ second requests than the COVID-19 implementation. This is also a very surprising finding because I did not believe that it was possible to be faster than the Standard implementation, but clearly given the right circumstances taking extra pick ups may not be the best choice for the overall average, as you are then more likely to produce data points on the extremes of possible values from the data set, such as the 200+ second request data points in this situation.

11: Discussion

The problem was clear, Regular implementations of elevator algorithms employed across large cities are not functioning properly during the COVID-19 pandemic. Not only is this frustrating tenants in apartments, it was also affecting large multi level offices from having all their employees seated on time from the morning rush. When people are being forced to take the stairs because wait times are too high, it is clear that something is not functioning as it is supposed to and needs to be changed.

It was theorized that if an algorithm never opened its doors unneccessarily, while maintaining the promise of only ever having one passenger that it would run optimally in a pandemic constraint situation. Calculating run time equation **T.3** , and then manipulating P=0 produces a best case running time for every single passenger delivery **T.1** .

Pseudo-code for multiple algorithms was then presented. Since the distribution algorithm was been shown to correctly distribute a set J of jobs of any size N to E elevators, and the Cab movement algorithm had been shown to successfully complete a Job Ji for elevator Ei as long as elevator Ei has a job – it follows that together these algorithms correctly distribute and complete all Jobs properly for any input set J.

Once the COVID-19 algorithm had been proven to be needed (fixes a problem), pheasible, and efficient it was time to verify this claim in simulation. An elevator simulator was created in order to simulate a down rush against different elevator implementations which outputs statistics that are then able to be turned into histograms or other visual data for consumption. In

order to show that the COVID-19 algorithm is effective, not only must we show that the algorithm is faster than the Regular implementation, but we must also show that the COVID-19 algorithm is not too much slower than the Standard algorithm operating without constraints. If the simulation can show this fact, then it is clear that the most appropriate choice for elevator algorithm in modern high rises during a pandemic is the proposed COVID-19 algorithm.

Through simulation analysis of small, medium and large buildings all underneath different types of workflow, namely slow and busy this above fact seems to hold true. Output statistics for all 3 elevator algorithms were very similar when the buildings were slow. This is a good finding, because it shows that the COVID-19 algorithm is not only efficient during the pandemic and when it is busy, but it is also an efficient elevator algorithm in general.

More interesting findings came when busy state simulations were run against the same building sizes. Given the same queue files, the Regular algorithm was simply unable to complete the jobs in a reasonable amount of time whereas the COVID-19 and Standard implementations showed no weakness and completed almost all jobs in under 4 minutes though most much quicker. The COVID-19 implementation was able to keep up with the Standard implementation operating with no extra constraints. They both produced very similar histograms, though the COVID-19 implementation often had more passengers in the larger end of the total trip time distribution, probably because of these extra constraints that the COVID-19 implementation was dealing with.

12: Conclusion

The COVID-19 algorithm is useful in the fact that it operates perfectly during a pandemic. The benefits in small buildings with only a few elevators is pronounced, as this is when the Regular algorithm fails the most. The COVID-19 algorithm maintains its average trip time speeds very close to the Standard algorithm, which operates without pandemic constraints and is known to be effective.

There is very little time consequences to using the COVID-19 algorithm during normal circumstances, with huge benefits being shown for when there is a pandemic. It could also be said that in the real world a trip on an elevator that promises it doesnt stop and start until you get off might actually be a more enjoyable experience than the way elevators are currently installed.

In a situation where there is a pandemic for the forseeable future, or where you want to offer a personal elevator experience to your tenants making the choice to install the COVID-19 algorithm to your elevator system is a beneficial one for many reasons that have been explored.

The main issue in the real world is that elevator systems are typically closed source, and once installed their algorithms are not interchangeable. Though theoretically, the proposed COVID-19 algorithm is beneficial, realistically it would be hard to "update" all current elevator systems to the improve COVID-19 version, as well as reverting back to the Standard implementation after the pandemic status is lifted.

Because of these real world practical aspects, having a COVID-19 algorithm or another similar algorithm for running in optimal conditions during a pandemic installed would be difficult. The fact that elevators have no interface in able to swap algorithms is a clear lack of foresight. COVID-19 has brought many unprecedented situations, and thus would have been hard to foresee but new elevators that are being installed should keep this in mind.

Ideally, new elevators that are being installed can keep this in mind as a priority. If there was a simple way to swap scheduling patterns inside elevators this could offer many other advantages not just for operating optimally during a pandemic. Algorithms faster at clearing people can work in the morning, and algorithms faster at bringing people back to their floor from ground level can work in the evening.

Potential limitations of these findings do exist. The pheasability of the COVID-19 algorithm has been verified through simulation, and as such the effectiveness of these claims is only as effective as the simulation. Simulation code is available [6], so this can be verified by the interested reader. Queue files generated from the TestGenerator that were used for the Results section have been kept and are stored in the github repository. As such as interested reader may also take the provided queue files, and re-create the graphs in the result section using the DataProcessor for further verification of simulation accuracy.

Another potential limitation is in the programming of the Standard and Regular Algorithm. In the simulation provided, these algorithms mimic that of what you would expect from a typical elevator system in the real world. Based on the constraints given in section 4 the Standard algorithm is indeed efficient and operates well, but there may be some ultra-efficient elevator algorithm that is not being used here.

As elevators are currently installed with innefficient algorithms for handling the pandemic, the need for the COVID-19 algorithm is clear. The correctness and guaranteed termination of the proposed algorithms were shown, and this can be verified through software simulation. The same simulation was used to compare various algorithms and the COVID-19 algorithm performs almost identical to that of the Standard algorithm which operates under less constraints. If there was a simple way to update the algorithm being used inside an elevator system, this paper proposes that the COVID-19 algorithm is the best available implementation.

13: Bibliography

1-5 are outsourced cited information

6-8 is information, data, or code that has been collected for use in this study.

- **1:** Tan Kok Khiang, Marzuki Khalid, Rubiyah Yusof 1999 Intelligent elevator control by ordinal structure fuzzy logic algorithm Link. (August 2020) - Used for elevator system criteria
- **2:** Frederick Ceder, Alexandra Nordin 2013 Elevator Control Strategies simulating different algorithms to find the most efficient strategy Link.(August 2020) - Used for elevator system criteria
- **3:** Viktor Bjorkholm, Jesper Brann 2015 Scheduling of Modern Elevators A description of modern elevators and a comparison of heuristics used for scheduling them Link. (August 2020) - Used for results and discussion section
- **4:** Anil Maheshwari and Michiel Smid Introduction to Theory of Computation Link.(August 2020) - Used for gerneral probability and algorithm proofs
- **5:** Jeff Erickson Algorithms Link.(August 2020) - Used for algorithm proofs
- **6:** Github Source Code for the Elevator Algorithm Simulation: Ethann Yakabuski 2020 Link.
- **7:** This data can be found at the provided github repository. [6]. This data was collected at random choice amongst available Ottawa apartments. This photo shows some details concerning floors/second speed and door time in seconds of various elevator shafts. This data was collected by Ethann Yakabuski between Oct 01 2020 and Nov 10 2020.
- **8:** This photo was taken of the elevator simulation being run. [6] Ethann Yakabuski

Ethann Yakabuski

100961945