Lift Control Simulation

Design Document by Isaac Cheng

Table of Contents

Development Log

Week 1

- Started development for the lift control project.
- Started designing user interface in Qt Creator.

- Converted the user interface file from a .ui file to a .py file using pyuic5.
- Created the main driver Python file which will be used to run the program.
- Added ability to edit the configuration of the simulation after it passes validation against null inputs.

Week 2

- Started to add full docstrings for all methods in the code for readability.
- Added feature to automatically generate a random set of starting positions for a given number of people.

- Stored people in a list of dictionaries, with an ID, starting floor, target floor, current floor, and status.
- Added validation against inputs of zero for editing the configuration.

- Added validation to prevent the target floor of each person being the same as their starting floor.
- Added in lift and direction as attributes for each person dictionary.
- Started implementing the lift moving as part of the simulation.
- Added validation to continue the simulation until all target floors have been reached.

- Added ability for more than one person to be in the lift at the same time (picking up extra people if they are on the way and there's space in the lift).
    - Revamped the system used to track people in the lift to implement this.
        - Removed in lift attribute for people.
        - Created a second list of dictionaries to track the people in the lift, and kept the first list for an overview of all people.

- Fixed a bug where the simulation would add people to the list of people in the lift twice.
- Fixed a bug where the simulation would fail to remove people from the list of people in the lift.
- Fixed a bug where the simulation would fail to move the lift in the correct direction.

- Improved docstrings to briefly summarise my algorithms in the code.
- Implemented the naïve mechanical simulation algorithm.
- Added some spacing for print statements to make simulations easier to track in the console.

Week 3

- Changed the generation of people from taking place when a simulation is run to taking place when the configuration is saved. This enables direct, fair comparisons

between the naïve and the improved simulation algorithms, as the same cases will be tested for each algorithm.
- Added a check in the improved algorithm to collect and deliver extra people if this is possible when the lift is on the way to collecting someone.

- Fixed some bugs occurring when the delivery of extra people on the way of collecting someone can fail, related to a logical error, causing the program to crash.

- Fixed an incorrect implementation of the improved algorithm which caused it to incorrectly take the same number of moves as the naïve algorithm.
- Cleaned up code to be more concise and efficient.
- Fixed bug causing people to be incorrectly marked as delivered.
- Changed ‘status’ key to ‘delivered’ for clearer representation.

- Fixed crashing caused when person to be collected is already on the current floor.
- Started implementing the user interface.
- Added refreshing of the user interface whenever a label is updated (to show number of moves etc).
- Added ability to generate a new simulation with the existing configuration settings.

Week 4

- Separated the main window into a window for the main menu and a window for the simulation, as the user interface for the simulation must change depending on the number of floors configured.
- Added white blocks to represent lift floors where the lift is elsewhere, and red blocks to represent lift floors when the lift is present.

- Added animations so the lift simulations can be visually tracked in real time.
    - The statistics on each floor (number of people waiting, and number of people delivered) and the coloured blocks change as the simulation runs, picks up, and delivers people.
- Added a status update message at the bottom of the simulation user interface to show when new people have been generated, or when someone has been delivered.

Description of Improved Lift Control System

Measuring Efficiency

There are many measures of efficiency in a lift control system, and it is important to establish the benchmark for how efficiency would be determined, as the most effective approach varies depending on what is being optimised. [1] For example, optimising the maximum wait time of a person would likely have a different heuristic algorithm to one which was optimising for the average wait time of a person.

To measure the efficiency of the lift control systems, I decided that the distance travelled by the lift would be the best benchmark, where it is measured in terms of number of floors travelled.

One reason for this is that it provides a universal measure, whereas other measures such as time taken for different operations may vary. For example, the time taken to stop on a floor may not be equivalent to the time taken to travel from one floor to another, and there are also other factors such as the time taken for the doors to open/close, which can all be related to the mechanical ability of the lift.

In addition, the distance travelled provides the best benchmark for measuring lift efficiency because distance will ultimately be the largest factor in delivering people to their respective floors the most quickly. The less the lift travels, the more quickly it can collect and deliver people to their floors, as it means less redundant time is spent with an empty lift travelling between floors.

Improvement from Changing Directions

The most obvious drawback of the naïve lift control system is that the direction of the lift can only change once the lift has reached either the top floor to change direction from going upwards to downwards, or the bottom floor to change direction from going downwards to upwards.

This is the primary cause of greater distance. In the worst-case scenario, it can result in an extremely large amount of additional distance travelled for each person.

The worst-case scenario occurs during the collection phase when the lift is only one floor away from the next person to be collected, and the person is located on either end of the building (either the top floor or the bottom floor), but the lift is travelling away from the next person. This means the lift would need to travel all the way to the other end first, before being able to turn around and travelling back up to the other end of the building to collect the person. The amount of distance wasted on the lift travelling to the opposite end of the building to the person needing to be collected would be (N – 2), where N is the number of floors in the building.

During the delivery phase, this worst-case scenario occurs when the person requests to go to the other end of the building (from bottom floor to top floor, or from top floor to bottom floor). This is not caused by any mechanical restrictions of the naïve lift control system, but it is the worst-case scenario because it is the maximum distance from their starting floor.

A simple example of this would occur in the case of a building with five floors (#0 to #4), where the lift is currently on floor #3 travelling downwards, and the next person in the

request queue needs to be collected from floor #4 (the top of the building) to be delivered to floor #0 (the bottom of the building). It would involve 7 units to collect the person (5 – 2 = 3 units unnecessarily spent on changing direction), and 4 units to deliver the person, summing to 11 units to handle the person's request.

My improved lift control system eradicates the amount of distance wasted on the lift travelling to change directions, as there are no mechanical restrictions on the lift being able to change directions. The lift cart can change directions as required.

For the example above, it would mean that only 1 unit (down from 7 unit previously) would be required to collect the person, and 4 units (unchanged from previously) would be required to deliver the person, summing to 5 units (down from 11 units previously) to handle the person's request.

Improvement from New Collection Algorithm

When I was running simulations for the naïve algorithm, I also noticed that a lot of distance was being wasted on travelling to collect each person from the queue individually, as there were many cases where the lift finished delivering the previous person on a floor which was far away from the starting floor of the next person.

I researched into existing lift control algorithms to get an idea of how existing algorithms deal with this. The elevator algorithm seems like the most commonly used algorithm for lifts currently; it works on the principle that the lift should continue travelling in the same direction for as long as possible, stopping as necessary. When the lift is idle and a request is made, the lift notes the direction which it needs to travel to fulfil that request, and it only services requests which are going in the same direction. The lift only considers changing direction when all requests in the current direction have been exhausted. [2]

The elevator algorithm is quite an efficient algorithm, but there are still some improvements which can be made.

Firstly, the algorithm has little consideration for the chronological order in which requests were made; if there were ten people who made requests, and the first person made a request going upwards, then the second person may end up being served last if the other eight people had a request going upwards (assuming lift capacity was not an issue).

Furthermore, there are a few cases when the algorithm can become very inefficient, some of which would feasibly occur in real life, as people are more likely to use a lift to make longer journeys (because shorter journeys can be easily taken using the stairs). For example, if there was a building with 100 floors, and the following requests were made:

Person #1: Floor #0 -> #1
Person #2: Floor #1 -> #0
Person #3: Floor #98 -> #99
Person #4: Floor #99 -> #98

In this scenario, the elevator algorithm would cause the lift to travel 198 floors, as it would fulfil orders from person #1 and #3 first (because person #1 made the first request, and person #3 is also going upwards), then person #2 and person #4.

A better approach for this configuration would be to stick within the bottom two floors to complete the requests from persons #1 and #2, then going to the top two floors to complete the requests from persons #3 and #4. This is not based on the chronological order of the requests, but rather based on the idea of fulfilling requests which are close together.

This suggestion of an alternative approach does conflict with the first point about having little consideration, though coincidentally, being less considerate of the chronological order coincidentally harms the efficiency of the algorithm in this case. There is also the idea that considering chronological order would make this less likely to occur; given three requests made in a row, it is statistically more likely than not that the requests will not all have the same direction (75%). Hence, having the lift change direction more often would help reduce the chances of this edge case occurring.

I wanted to ensure that the queue was being respected to a point, so I discovered that it would be much more efficient for the algorithm to collect and deliver people en route to the collection of the next person in the queue. This would deliver people from the queue at no extra cost in distance, yet there would still be some sort of priority queue to ensure that people who requested the lift first would get priority over those who requested the lift later, all else being equal.

My improved algorithm implemented this by looping through the list of people, and checking for the first person in the list who has not been marked as delivered. This person will be the next person to be collected, and they are added to a list of people pending to be collected. The direction which the lift will need to travel is calculated by the program if the starting floor of the next person is not the same floor as the lift is currently on.

In the process of moving up or down to collect the next person, the algorithm checks the list of people to see if there is anyone who can be delivered en route of the collection of the next person in the queue. To do this, it checks if the request meets several criteria. The person must be going in the same direction as the lift is, the lift must have space for this extra person, and the target floor must be closer to the lift than the starting floor of the next person in the queue (to check that they can be collected and delivered on the way). If the request meets the criteria, the person is added to the list of people pending to be collected.

Each time the lift reaches a new floor, it checks for the list of people pending to see if the lift has reached any of their starting floors. If so, it collects them, adds them to the list of people in the lift, and removes them from the list of people pending. It also checks if there is anyone in the lift who can be delivered; these people would have been collected on a previous floor, and it would complete the process of delivering that person en route of collecting the next person in the queue.

An example of when this new collection algorithm would see a benefit to the distance travelled would be if there were five people; person #1, person #2, person #3, and person #4 who made requests in that order:

Person #1: Floor 4 -> 5
Person #2: Floor 2 -> 3
Person #3: Floor 5 -> 6
Person #4: Floor 3 -> 4
Person #5: Floor 1 -> 0

With my improved algorithm, the lift (which started at floor 0) would look at the queue, and see that person #1 was the first person who made a request for the lift. Then, it would search the list of people to see which people it can deliver en route.

Persons #2 and #4 would be eligible for this, so they would be added to the lift on the way of collecting person #1. Person #3 is not eligible for this because their target floor is further away than the starting floor of person #1, and because their starting floor cannot be reached on the way of collecting person #1. Person #5 is not eligible for this because the direction they would require the lift to travel to deliver the person is down, whereas the lift is moving upwards.

The lift would pick up person #2 on floor 2, and deliver them on floor 3. On floor 3, it would also pick up person #4, then deliver them on floor 4. At that point, person #1 would be collected, so the algorithm would move onto the delivery stage.

In contrast, the naïve algorithm would not consider other people; it would simply see that person #1 was the next person in the queue to be collected, and pick them up. It would ignore persons #2 and #4, which could otherwise have been collected and delivered on the way.

Implementation of Improved System

Traditionally, lift control has been built around the premise of people calling the lift with a two-button approach, one button to go up, and one button to go down.

This premise works well for the naïve, mechanical algorithm, as the lift only needs to know if a person is moving in the same direction, and not where they are going.

However, modern lift control systems such as mine require knowledge on which floor a person calling the lift wants to reach. As opposed to having a button each for going up or down, people call the lift by entering their destination floor number. This knowledge is used to determine whether it would be efficient to pick up a person at a given moment.

For example, the New York Marriott Marquis, a hotel located on Times Square, implements this system [3] as one of the first installations of destination dispatch, which is an optimisation technique used for multi-elevator installations to group passengers with the same destinations into the same elevators.

This optimisation technique claims to reduce trip times (which can be equated to distance in floors travelled for these simulations) by 25%, improving availability of lift capacity by 30%. The reduction is obtained by dynamically allocating people to elevators in a way which avoids excessive intermediate stops [4].

My improved system is a similar implementation to destination dispatch technique, although I am applying this for a one-elevator system, so the process of grouping passengers is different. Instead of grouping passengers with the same destinations, I am grouping passengers with *similar* destinations into the same elevator, which reduces distance travelled on collecting people, hence reducing total distance travelled.

Data Structures and Algorithms

Dictionaries

In my lift simulation, *person* dictionaries are used to store the information on each person generated, which includes information on their identification, their start floor, target destination floor, current floor, whether they have been delivered, and the direction they need to travel from their start floor to reach their target destination floor.

These dictionaries are added one by one to a *people_overview* list when people are being generated.

Lists

Lists are used similarly in the naïve and improved algorithms.

In both algorithms, a *people_overview* list is used to store dictionaries of people generated. This list is then used to copy people over to other lists.

Both algorithms use the *people_overview* list to copy people over to the *people_lift* to track the people in the lift.

Lists are also used to act as priority queues in *people_pending* and *people_lift*.

Priority Queues

Priority queues are paramount to the success of my improved lift control system, as they are how the lift decides who to pick up, and whether to pick someone up at any given point. Compared to traditional queues, this data structure enables greater flexibility in how elements are removed, based on which element has the highest priority at that point. They are used in both the naïve and the improved algorithms, but they have much greater significance in the improved algorithm.

Only my improved algorithm uses a priority queue for people who are going to be collected. When there is nobody to collect, it adds the first person in *people_overview* who has not been delivered to the *people_pending* priority queue. Then, the lift checks if it can collect (and deliver) people on the way of collecting this first person, adding people to *people_pending* if this is the case. This priority queue gives priority to the people who are closest to being collected; it does this by finding the minimum absolute difference between the floor the lift is on and the start floor of each person in the queue. Once the lift reaches the next person's start floor, it is moved from *people_pending* and into *people_lift*, as they are collected and enter the lift.

The naïve mechanical algorithm does not need to use *people_pending*, as it does not collect and deliver people on the way of collecting the first person; it only collects that first person.

Both algorithms use *people_lift* to track the people in the lift, with the exact same implementation. There are two cases when someone is added to this priority queue. Firstly, a person is added to *people_lift* as the lift transitions from the collection phase to the delivery phase. Secondly, a person is added to *people_lift* in the delivery phase if there is someone on the lift's floor who is going in the same direction as the lift, and there is space in the lift to fit them.

Iteration

Two forms of iteration are used in my lift simulation, *for* loops and *while* loops.

for loops are used to iterate over the elements of a list so that a linear search can be performed.

while loops are used to continuously loop over code until a condition no longer holds. It is used when generating each person to ensure that their target destination floor is different to their start floor.

It is also used in both algorithms to continue the simulation until all people have been delivered.

Furthermore, *while* loops are used in the naïve algorithm to continue moving the lift until the next person to be collected has been collected.

Finally, both algorithms use a *while* loop to continue moving the lift in the delivery phase when the lift is occupied; this ensures all people in the lift are delivered.

Linear Searches

Linear searches are used in conjunction with *for* loops to check if there are people who have not been delivered (so the algorithm knows to continue the simulation).

It also checks if people are suitable to be added to the priority queues by checking the dictionary keys of each person.

Selection

Selection is used throughout the program; it is essential in ensuring that the program does not crash by trying to load/change values when they are not applicable. To do this, it ensures that certain procedures are only carried out if they are relevant to the situation.

When opening the main window for the simulation, it ensures that the correct user interface is loaded. When statistics for each lift floor are loaded, selection is used to ensure that the program does not try to change values for a user interface element which does not exist (such as the number of people delivered for floor #4 when there are only five floors), which would cause a program crash.

Saving the simulation configuration also uses selection to validate against the user inputs, as all fields must be required for the simulate to work, and values must be input within certain ranges for a simulation to be possible.

In addition, selection is used during the simulations to ensure that people are only moved to priority queues when they are suitable. Only those who have not been delivered should be added to *people_pending*, and additional people should be added to *people_lift* if they are travelling in the same direction as the first person in the lift. When the lift is moving, it should move up if the *lift_direction* value is ‘Up’, or down if the *lift_direction* is ‘Down’. Without these validations, the simulations would not apply the algorithms as intended, often resulting in the simulations running indefinitely, as people would not all be delivered.

Performance Analysis

To test how effective my improved algorithm was in saving time compared to the naïve mechanical algorithm, I performed ten tests for numerous types of configured scenarios.

Within each test, people were randomly generated with an equal distribution for both the starting floor and the target floor. The only restriction I added was that people generated cannot have the same starting and target floor.

I performed the simulation with the naïve algorithm and the improved algorithm for each test, so the number of moves taken by the algorithms could be compared fairly.

With each configuration set, I performed ten tests with both the naïve and the improved algorithms, and recorded the distance travelled each time. Using these results, I also calculated the differential in the distance travelled for each test, as this shows how many units of distance were saved by using the improved algorithm over the naïve mechanical algorithm. The differential was recorded in raw distance terms, and in percentage decrease in units of distance from the naïve to improved algorithms.

From the ten tests, I calculated an average in the distance travelled for both the naïve and the improved algorithms, and calculated the differential for this too.

Using the test results, I created performance charts to provide a visual representation of the distance travelled for the naïve and improved algorithms, as well as the differential between these two algorithms.

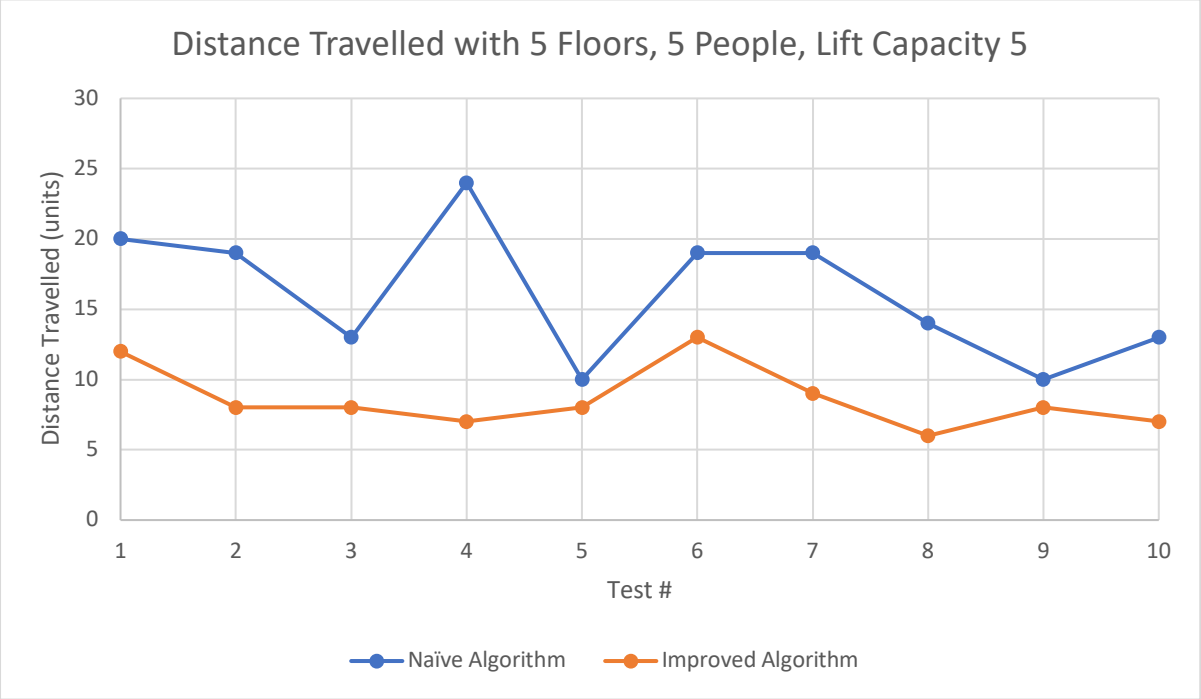
Configuration #1: 5 / 5 / 5

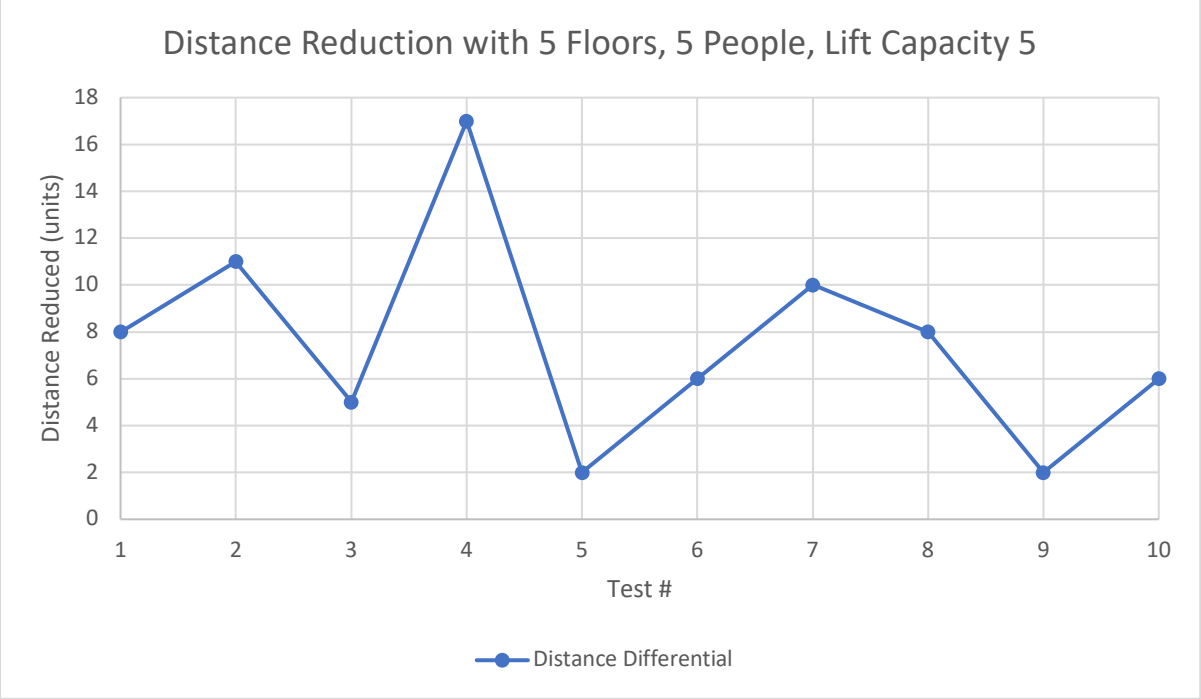
Number of Floors: 5
Number of People: 5
Lift Capacity: 5

Test Results

Test	**Distance Travelled (Naïve)**	**Distance Travelled (Improved)**	**Differential**
#1	20	12	8 (40.0%)
#2	19	8	11 (57.9%)
#3	13	8	5 (38.5%)
#4	24	7	17 (70.8%)
#5	10	8	2 (20.0%)
#6	19	13	6 (31.6%)
#7	19	9	10 (52.6%)
#8	14	6	8 (57.1%)
#9	10	8	2 (20.0%)
#10	13	7	6 (46.2%)
Avg	*16.1*	*8.6*	*7.5 (46.6%)*

Performance Charts





Commentary

I started by testing with a very basic case. This configuration would simulate a small building which expects low footfall. For example, many small businesses may have lifts to provide disabled access to employees and/or customers.

From the test results shown here, my improved algorithm has clearly reduced the distance travelled by the lift by a large margin overall; on average, it caused a reduction in distance travelled of 46.6% from the naïve algorithm. There were no cases in which the naïve algorithm led to the lift travelling less distance than my improved algorithm, which is a good sign. However, there is some variance in how much the distance is reduced by.

It can be concluded that my lift algorithm is an improvement over the naïve, mechanical lift algorithm even for small buildings.

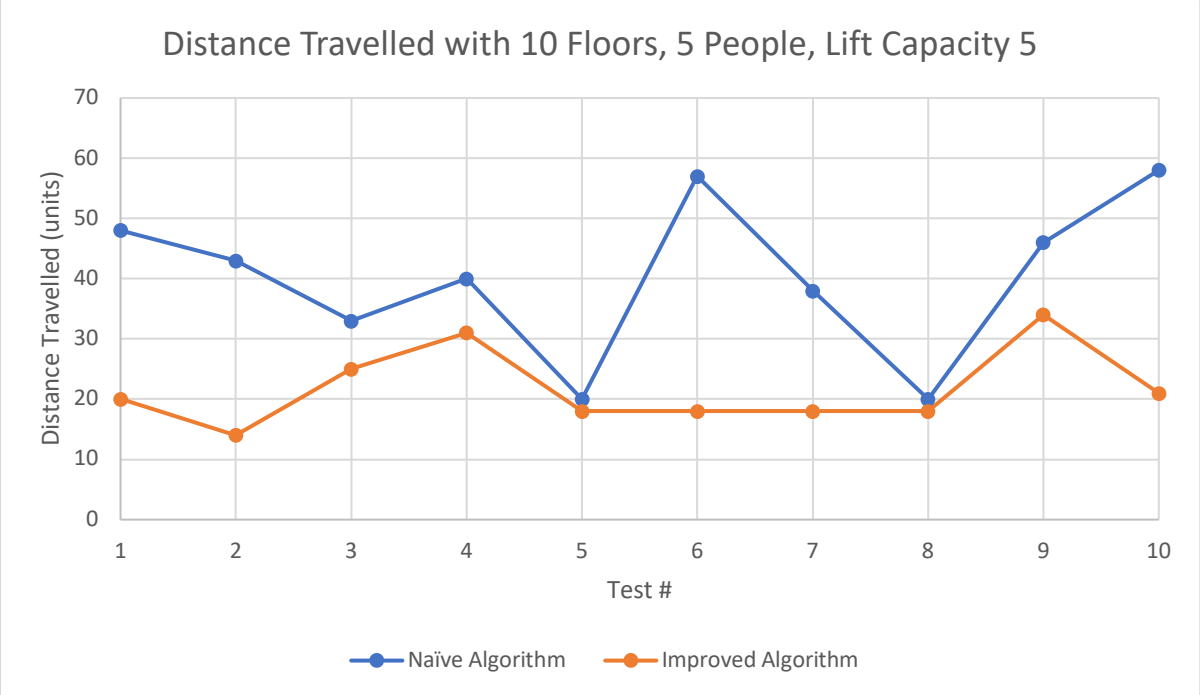
Configuration #2: 10 / 5 / 5

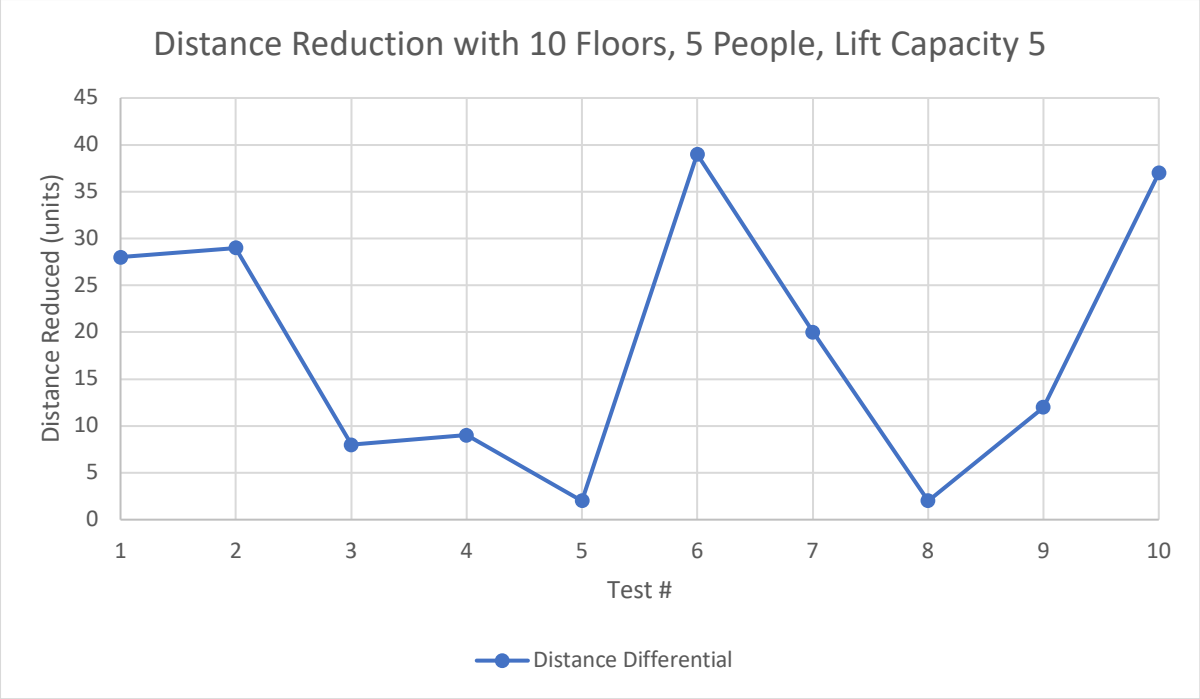
Number of Floors: 10
Number of People: 5
Lift Capacity: 5

Test Results

Test	Distance Travelled (Naïve)	Distance Travelled (Improved)	Differential
#1	48	20	28 (58.3%)
#2	43	14	29 (67.4%)
#3	33	25	8 (24.2%)
#4	40	31	9 (20.5%)
#5	20	18	2 (10.0%)
#6	57	18	39 (68.4%)
#7	38	18	20 (52.6%)
#8	20	18	2 (10.0%)
#9	46	34	12 (26.1%)
#10	58	21	37 (63.8%)
Avg	*40.3*	*21.7*	*18.6 (46.2%)*

Performance Charts





Commentary

Following the test results from the previous configuration, I wanted to test how changing the number of floors would impact the performance of both the naïve and the improved algorithm for a building with low footfall. This would simulate a building like the first configuration, but with some floors which have no people on them; this may be the case where there are floors used for storage, for example.

With an increased number of floors yet a low number of people, results proved to be very similar to configuration #1, but with larger distances travelled by the lift as expected for both algorithms due to the increase in the number of floors. The average reduction in distance travelled with the improved algorithm was 46.2%; very marginally than the previous configuration, which can be attributed to the limited number of tests performed. My lift algorithm was equally as effective in reducing the distance travelled with this configuration.

There was an improvement in distance travelled using my algorithm over the naïve algorithm, but the extent of improvement varied a lot by test. Like previously, this can be attributed to having a small number of people generated.

Testing this configuration demonstrated that my algorithm improves upon the naïve algorithm for small buildings, even when the number of floors is increased.

Configuration #3: 10 / 25 / 5

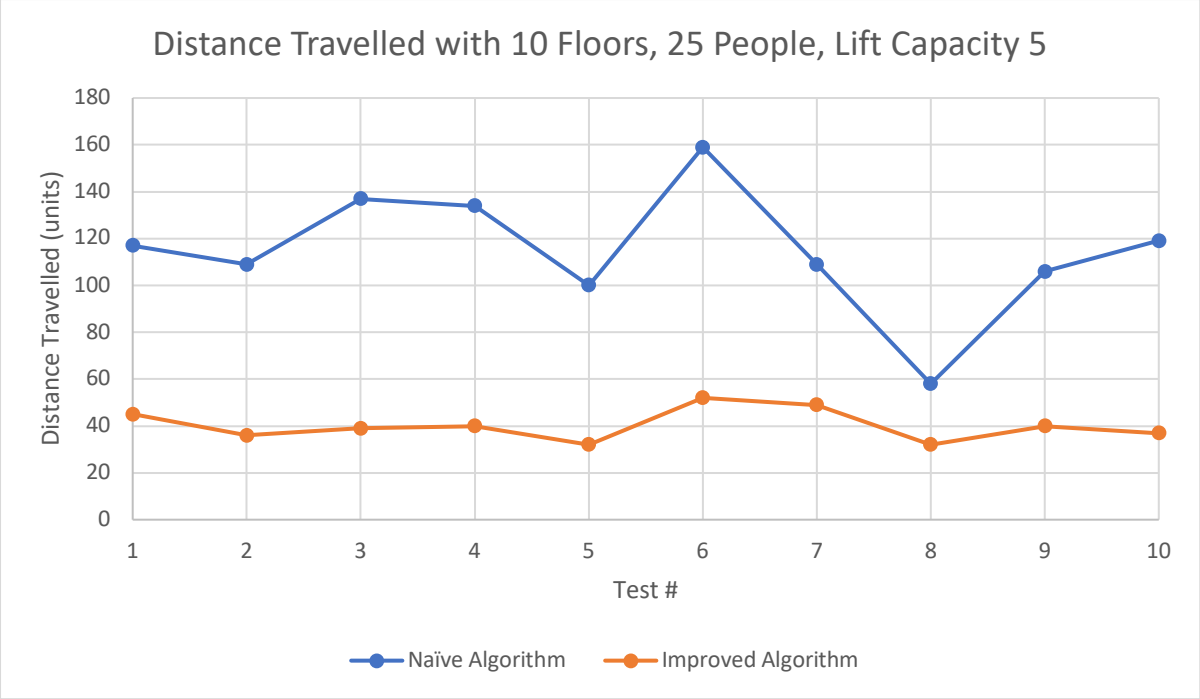
Number of Floors: 10
Number of People: 25
Lift Capacity: 5

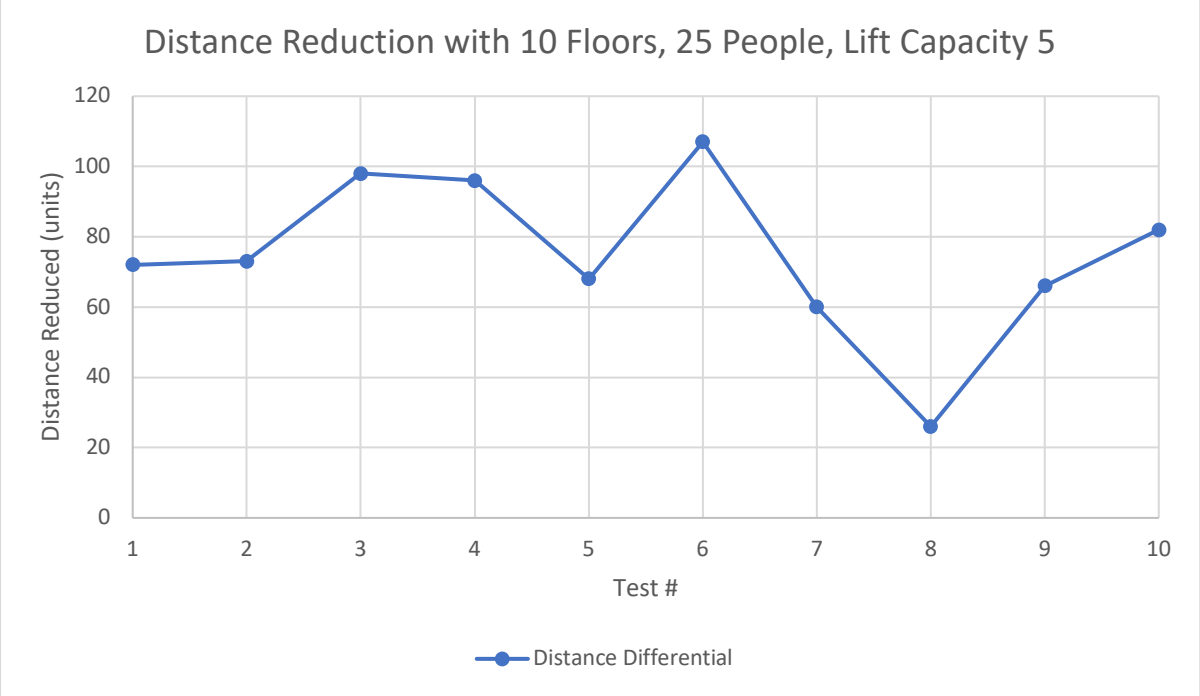
Test Results

Test	Distance Travelled (Naïve)	Distance Travelled (Improved)	Differential
#1	117	45	72 (61.5%)
#2	109	36	73 (67.0%)
#3	137	39	98 (71.5%)
#4	134	40	96 (71.6%)

#5	100	32	68 (68.0%)
#6	159	52	107 (67.3%)
#7	109	49	60 (55.0%)
#8	58	32	26 (44.8%)
#9	106	40	66 (62.3%)
#10	119	37	82 (68.9%)
Avg	*114.8*	*40.2*	*74.6 (65.0%)*

Performance Charts





Commentary

In the previous two test configurations, it was clear that at least some of the variation in the distance differentials was caused by the small number of people being generated for the simulations. Therefore, I decided to increase the number of people for this configuration.

This configuration could represent a medium sized building. For example, it may represent a building which is used for meetings between departments within a small business. It is important to remember that while 25 people does not seem like a lot, this is only the number of people making lift requests, rather than the total number of people in the building.

The test results for this configuration backed up the suggestions that the variations may have been largely caused by the smaller number of people in the first two test configurations. The distance differentials between the naïve and improved algorithms were much more consistent throughout the ten tests, and there was an even greater percentage distance reduction with my improved algorithm, increasing to an average of 65% reduction in units for this test configuration.

Testing from this configuration suggests that the effectiveness of my algorithm may become more apparent the more demanding the simulation is, with more floors and more people likely to increase the distance differentials.

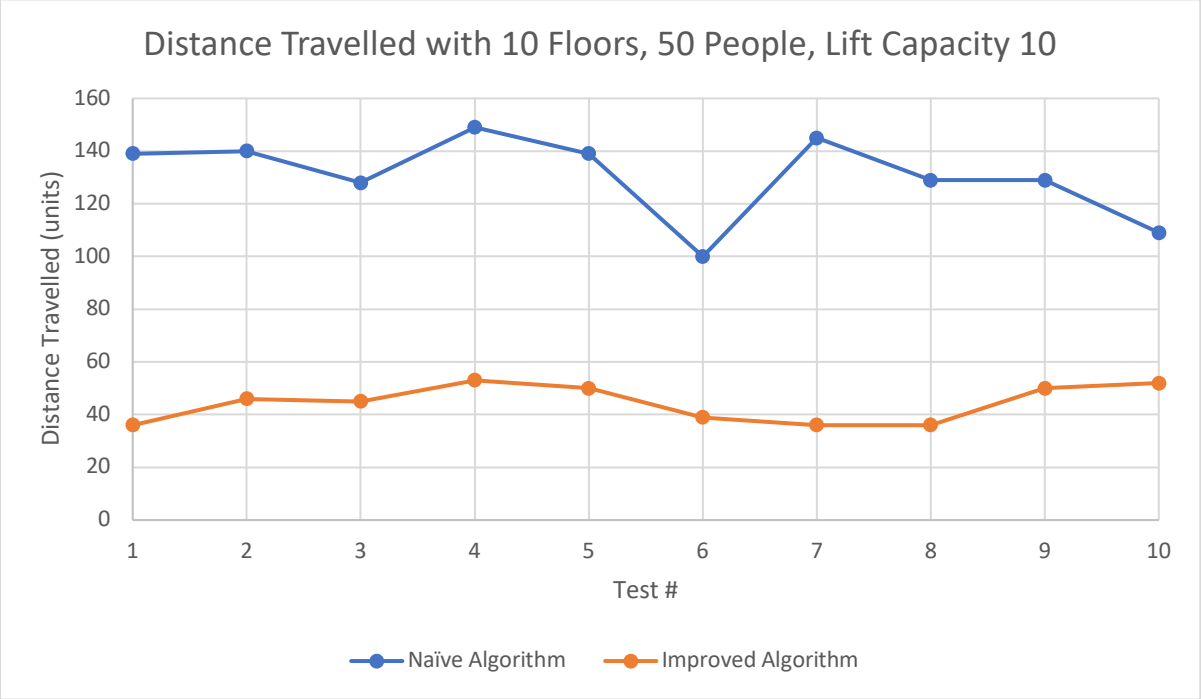
Configuration #4: 10 / 50 / 10

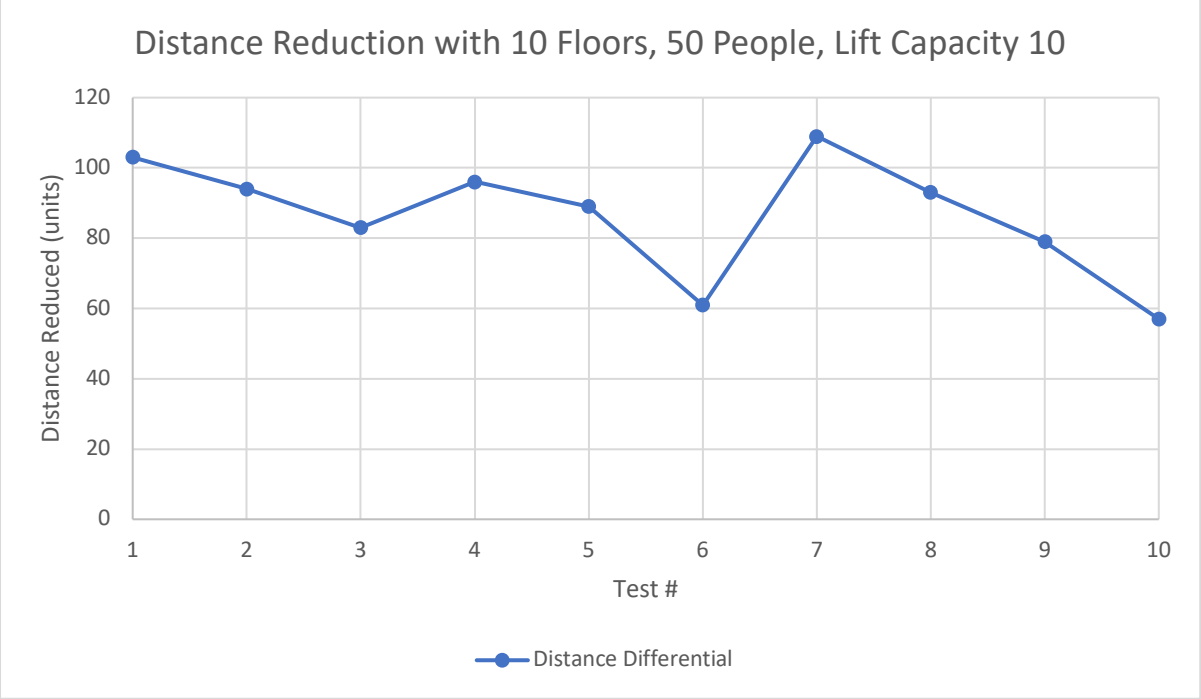
Number of Floors: 10
Number of People: 50
Lift Capacity: 10

Test Results

Test	**Distance Travelled (Naïve)**	**Distance Travelled (Improved)**	**Differential**
#1	139	36	103 (74.1%)
#2	140	46	94 (67.1%)
#3	128	45	83 (64.8%)
#4	149	53	96 (64.4%)
#5	139	50	89 (64.0%)
#6	100	39	61 (61.0%)
#7	145	36	109 (75.2%)
#8	129	36	93 (72.1%)
#9	129	50	79 (61.2%)
#10	109	52	57 (52.3%)
Avg	*130.7*	*44.3*	*86.4 (66.1%)*

Performance Charts





Commentary

For this test configuration, I wanted to see how my lift algorithm would perform in a larger building to simulate a medium sized building. This may include a building which is used for business meetings between companies, as more people would be attending, and a bigger lift with a greater capacity would likely be used in these buildings.

The results for this test configuration were interesting; there was very little increase in the distance differential percentage. As we have previously seen that increasing the number of people causes the differential to increase percentagewise, this suggests that the increased lift capacity may have cancelled out the effects of increasing the number of people, which

had led to this smaller increase. This is because while there were more people to collect and deliver in the first place, more people would have been able to be transported at once.

Consistency in distance reduced by my algorithm improved a lot with this test configuration. This suggests that the less constrained the algorithm is by the lift capacity, the more it can be taken advantage of to collect and deliver multiple people at once. As aforementioned, the increase in the number of people would have also greatly contributed to this increased consistency.

Observing the distance travelled for both algorithms helps form the likely explanation that my algorithm becomes more effective directly based on the distance travelled, as opposed to being directly affected by the constraints set in the simulation. Increasing the number of people while also increasing the lift capacity meant that the distance travelled by the lift changed very little, and the small increase in distance differential also followed this.

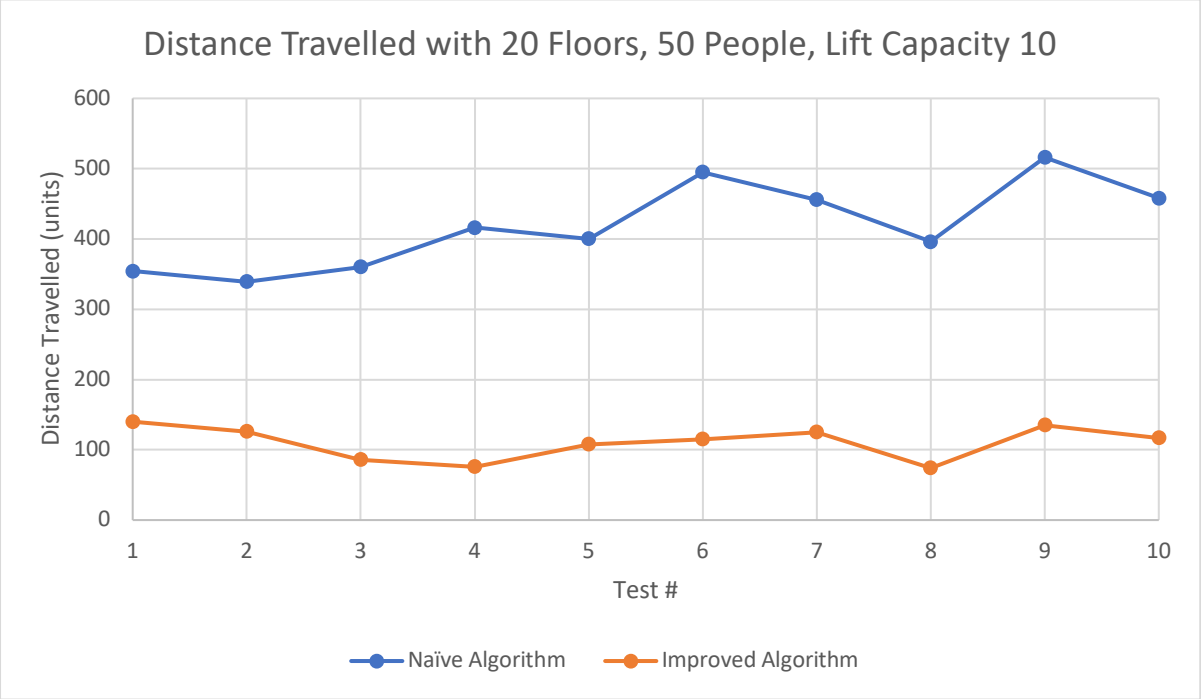
Configuration #5: 20 / 50 / 10

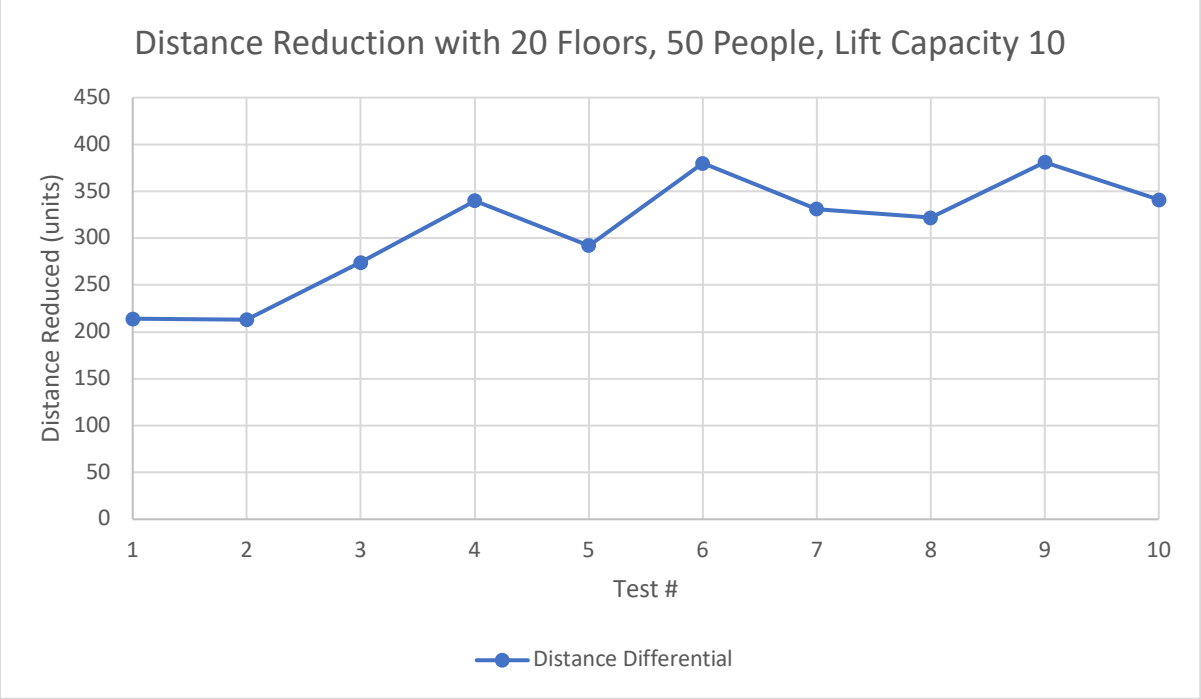
Number of Floors: 20
Number of People: 50
Lift Capacity: 10

Test Results

Test	**Distance Travelled (Naïve)**	**Distance Travelled (Improved)**	**Differential**
#1	354	140	214 (60.5%)
#2	339	126	213 (62.8%)
#3	360	86	274 (76.1%)
#4	416	76	340 (81.7%)
#5	400	108	292 (73.0%)
#6	495	115	380 (76.7%)
#7	456	125	331 (72.6%)
#8	396	74	322 (81.3%)
#9	516	135	381 (73.8%)
#10	458	117	341 (74.5%)
Avg	*419*	*110.2*	*308.8 (73.7%)*

Performance Charts





Commentary

As the previous test configuration suggested that the number of people was a factor leading to inconsistent distance differentials in other configurations, I wanted to test if the number of floors was also a factor. I performed the same test configuration, but with a larger number of floors.

This would represent a similar building to the previous configuration, but a higher number of floors means that people traffic would be reduced throughout the building. The lift would have to spend more time travelling between floors to collect and deliver people.

The results showed a small increase in the distance differentials between the naïve algorithm and my algorithm, but not as significant of a change compared to increasing the number of people.

Consistency was also affected in a similar way; there was a small increase, but not a major one. This is because more floors would make it more important to have an algorithm which transports more people at once, but the number of people would ultimately be a much more significant factor, as it causes a greater increase in the total distance travelled with both algorithms.

This set of tests demonstrated that the number of people is the most significant factor in distance differential for my algorithm by far. My algorithm is most effectively used in busier buildings.

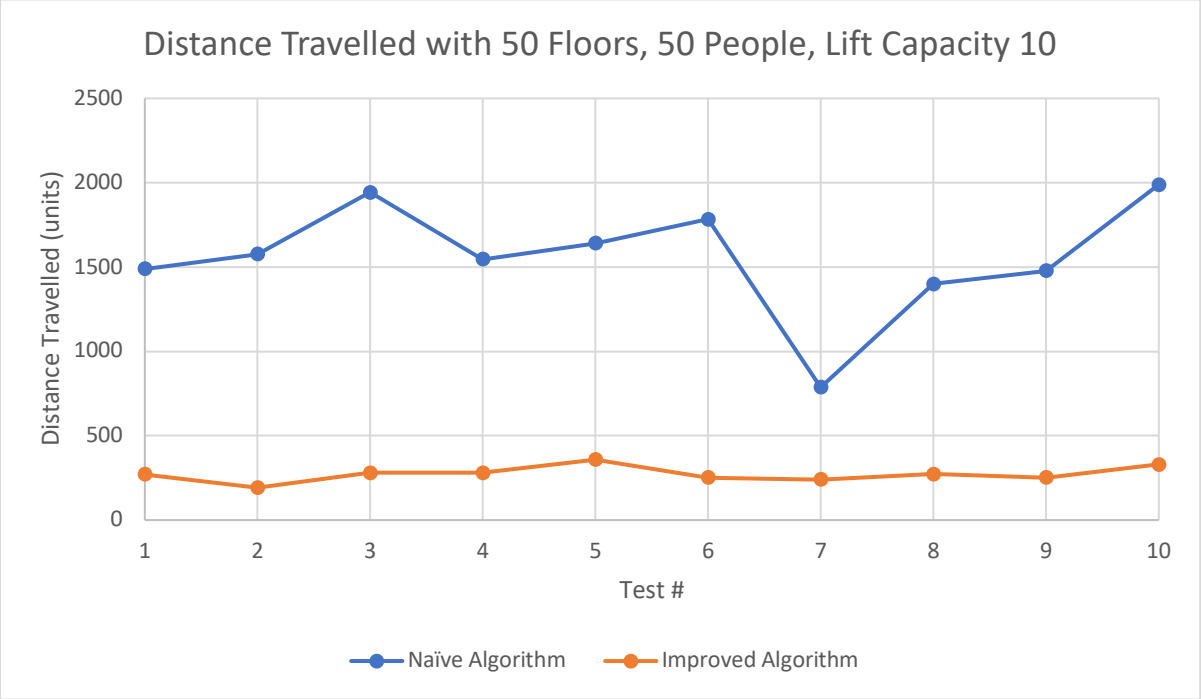
Configuration #6: 50 / 50 / 10

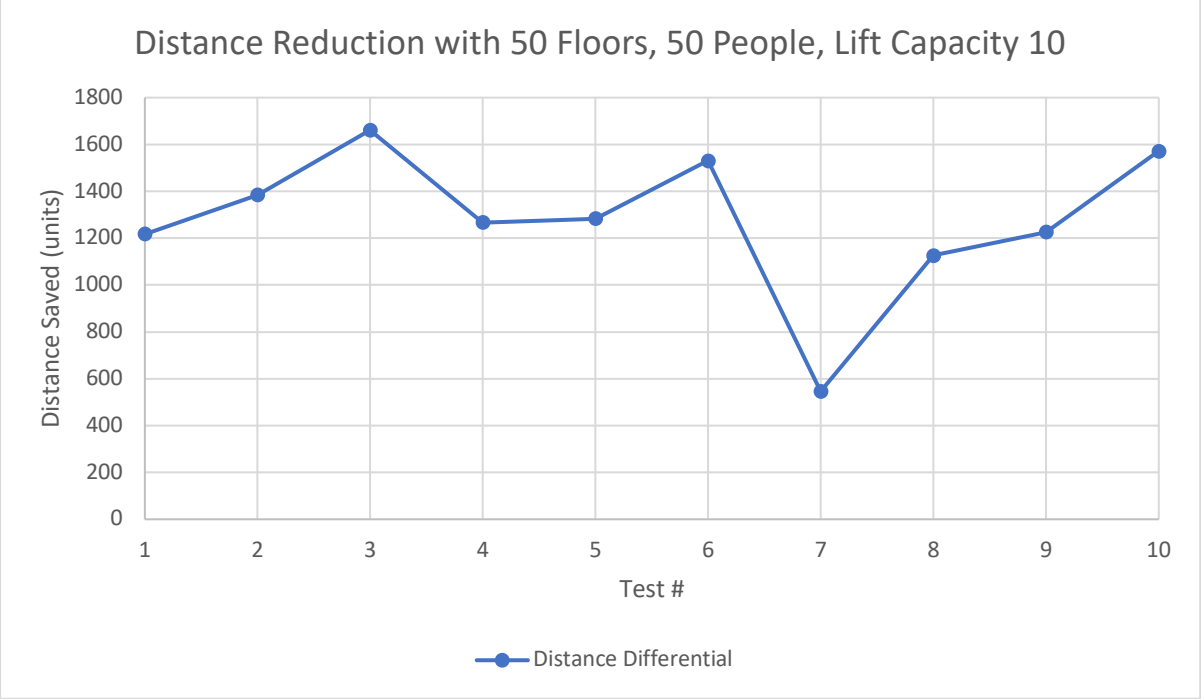
Number of Floors: 50
Number of People: 50
Lift Capacity: 10

Test Results

Test	**Distance Travelled (Naïve)**	**Distance Travelled (Improved)**	**Differential**
#1	1489	271	1218 (81.8%)
#2	1577	192	1385 (87.8%)
#3	1942	281	1661 (85.5%)
#4	1547	280	1267 (81.9%)
#5	1641	358	1283 (78.2%)
#6	1783	252	1531 (85.9%)
#7	787	241	546 (69.4%)
#8	1399	273	1126 (80.5%)
#9	1477	251	1226 (83.0%)
#10	1989	330	1659 (83.4%)
Avg	*1563.1*	*272.9*	*1290.2 (82.5%)*

Performance Charts





Commentary

To further test the conclusions from test configuration #5, I increased the number of floors even more. This would simulate a larger building which is very loosely packed with people; it may mostly consist of floors used to host servers/used for storage, with smaller teams working on each of the other floors.

When disregarding the anomaly of test #7, the consistency of distance differential seemed to improve slightly over the previous test configuration. This was likely due to the same number of people being generated for the simulation.

The percentage in distance reduction had a larger increase, but this can be attributed to the much larger increase in the number of floors in these simulations compared to the previous configuration; the number of floors went up by 10 from configuration #4 to configuration #5, whereas it went up by 30 from configuration #5 to configuration #6.

This test configuration showed that my algorithm is very well suited to large buildings, as the high number of floors makes it more important for the lift to move efficiently.

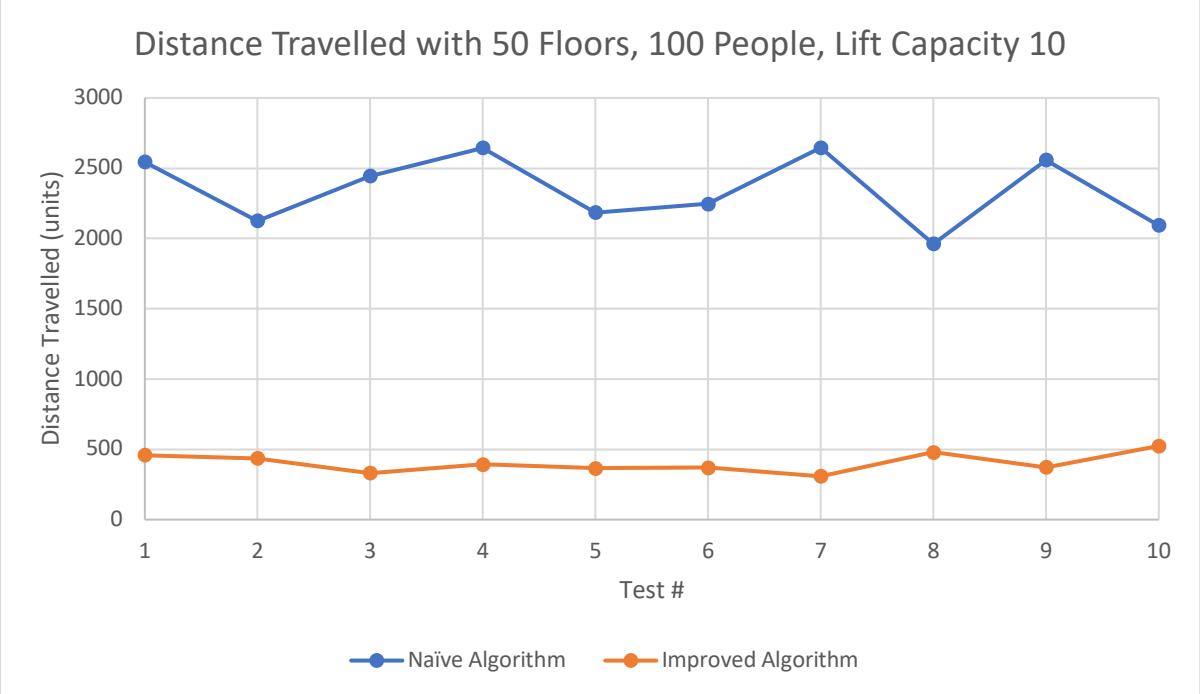
Configuration #7: 50 / 100 / 10

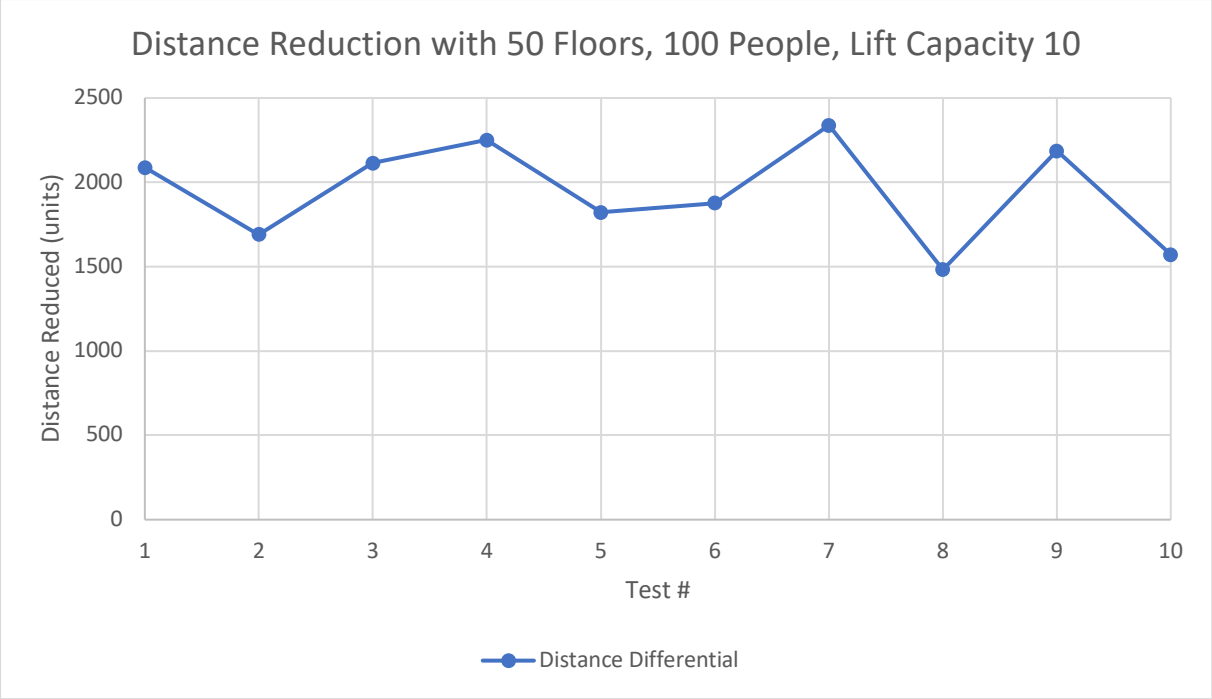
Number of Floors: 50
Number of People: 100
Lift Capacity: 10

Test Results

Test	Distance Travelled (Naïve)	Distance Travelled (Improved)	Differential
#1	2545	458	2087 (82.0%)
#2	2126	436	1690 (79.5%)
#3	2445	331	2114 (86.5%)
#4	2644	393	2251 (85.1%)
#5	2186	365	1821 (83.3%)
#6	2246	369	1877 (83.6%)
#7	2646	308	2338 (88.4%)
#8	1961	479	1482 (75.6%)
#9	2558	372	2186 (85.5%)
#10	2095	524	1571 (75.0%)
Avg	*2345.2*	*403.5*	*1941.7 (82.8%)*

Performance Charts





Commentary

For the final test configuration, I wanted to simulate a large building which was more densely packed than the previous configuration, as this would more realistically simulate a large building being used by a branch of a large corporation. The only change I made was doubling the number of people generated.

From the last set of configurations, something which was immediately noticeable was that the number of people in the lift at any given time was much larger with the improved algorithm than with the naïve algorithm.

This highlights the improved efficiency of my algorithm, because more people being in the lift at the same time means that multiple people are being collected and/or delivered while the lift is covering the same effective distance. An effective lift control system aims to maximise the number of people in the lift at any given time, as this minimises the distance travelled per person. This is the case with my improved lift algorithm, as demonstrated from this set of test results.

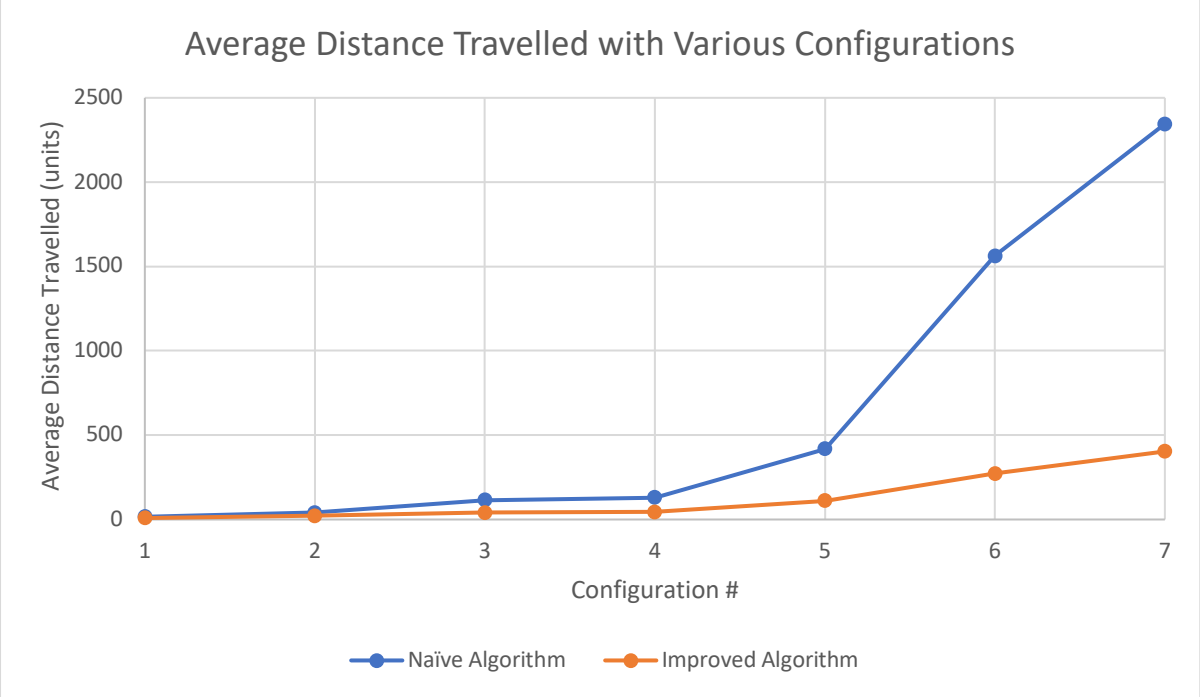
A large increase in the number of people led to the consistency of the distance differentials greatly improving as expected.

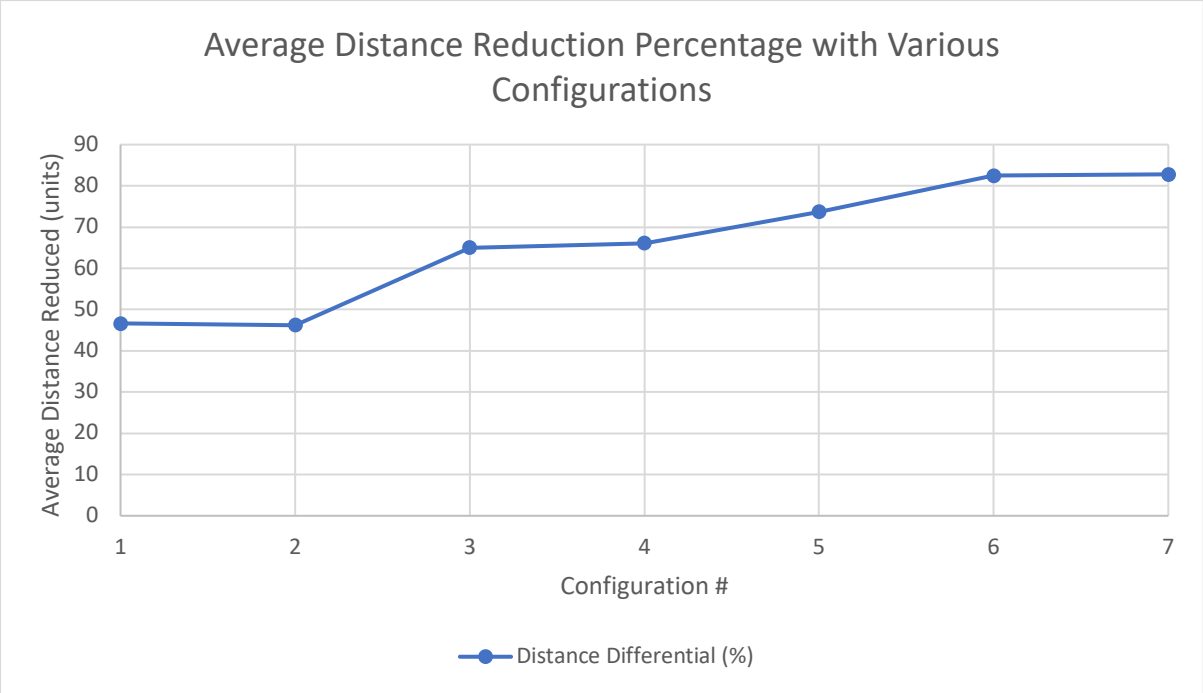
However, the average distance reduced percentagewise increased very marginally, and any changes could be explained by the limited number of tests performed within a single configuration set. This suggests that a large building with many people is where my improved algorithm starts to show its full effect; any further increases to the constraints would see a similar percentage reduction of around 82.5%.

This test configuration demonstrated that my algorithm is extremely well suited to being used in large buildings by large corporations, as it saved almost 2,000 units of distance on average during this set of tests.

Summary

Performance Charts





Commentary

By analysing the averages of data between the various configurations and considering the context behind that data, several conclusions can be drawn.

As I increased the difficulty of the simulation, the naïve algorithm quickly showed its inefficiencies; it required the lift to travel much further distances, whereas my improved algorithm showed a more gradual increase in distance travelled. This is extremely good, because it means that my algorithm has a more predictable performance, and it can be better scaled up for bigger and busier buildings.

The largest increase in average distance reduction percentage occurred when I increased the number of people generated from 5 to 25. The first two configurations had very few generated people, and the differences between the naïve algorithm and my algorithm were much less profound during these tests, which implies that the algorithm for controlling the lift has far less significance when there are fewer people to transport.

This makes sense, as the fewer the people there are, the fewer situations there are in which multiple people can be transported at the same time, meaning more efficient algorithms have less of an impact.

Despite this, my algorithm for lift control demonstrated consistently better performance than the naïve mechanical algorithm. Out of 70 tests performed in total with various configurations, there was not a single case in which the naïve algorithm caused the lift to travel less distance than my algorithm did. Therefore, my algorithm is a reliable improvement over the naïve algorithm, even if the degree of improvement varies depending on the building and people density of that building.

Smaller increases occurred when I continued to increase the number of people, and when I increased the number of floors, but these were much less significant increases. Hence, it can be concluded that my algorithm becomes more effective the larger and the busier the building is, but only up to a certain point. There was no difference in the distance reduced between configuration #6 and configuration #7 despite increasing the number of people by a large amount.

It can be expected that my algorithm will reduce the distance travelled by the lift by up to around 82.5% compared to the naïve mechanical algorithm. This dwarfs the claimed 25% reduction in trip times (which translates to distance travelled in these simulations) of the destination dispatch technique [4]. Despite this, it should be considered that my improved algorithm has been designed for a one-elevator installation, whereas the destination dispatch technique is designed for a multi-elevator installation.

Overall, my algorithm is an extremely effective solution to improve upon the naïve mechanical algorithm. Performance analysis has showed that the distance travelled by the lift is consistently lower using my proposed algorithm, meaning that it is suitable for all cases. In addition, it scales up very effectively for larger and busier buildings, which shows that it has been optimised for the general case, making it useful for large corporations who would benefit the most from an efficient lift control algorithm.

References

[1] kilotesla, “What's the most efficient elevator algorithm for a single elevator serving a building with 100 floors?,” 8 February 2020. [Online]. Available: https://www.reddit.com/r/math/comments/f0wfv6/whats_the_most_efficient_elevator_algorithm_for_a/fgz6wnr/. [Accessed 22 April 2020].

[2] J. Dunietz, “The Hidden Science of Elevators,” Popular Mechanics, 2 September 2019. [Online]. Available: https://www.popularmechanics.com/technology/infrastructure/a20986/the-hidden-science-of-elevators/. [Accessed 22 April 2020].

[3] cplai, “The Smartest Elevators with destination dispatch,” 5 November 2010. [Online]. Available: https://www.youtube.com/watch?v=T6gzm_ifzg8. [Accessed 14 April 2020].

[4] Wikipedia, “Destination dispatch,” 23 November 2008. [Online]. Available: https://en.wikipedia.org/wiki/Destination_dispatch. [Accessed 14 April 2020].