Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Brian Funk (22-918-957), Andrin Gasser (21-923-594),
Silvan Metzker (22-915-508)

Elevating Efficiency

Project Report

Complex Social Systems: Modelling Agents, Learning and Games

Supervision

Prof. Dr. Dirk Helbing
Dr. Dino Carpentras
Dr. Damian Dailisan

Abstract

In our research, we conduct simulations of multicar vertical transportation systems, systematically comparing their policies to identify the most optimal strategy. We implemented policies also used in disk scheduling, including SCAN, LOOK, FCFS and SSTF. Furthermore, we present our proprietary parametrized policy (PWDP) additionally improved by standard optimisation techniques. Through comprehensive analysis of different metrics, we found the more simple approaches, such as SCAN or LOOK, to be performant. Moreover, PWDP performed overall better than the other policies. However, we concluded that further optimisation of parameters not to improve the performance drastically either, but with a clear trend, indicating the importance of parameters.

December 11, 2023

Contents

1 Introduction

An efficient elevator policy is vital to customer satisfaction of an elevator system. Not only does a good policy improve overall waiting times, but it also decreases time spent in and waiting for the elevator, thus allowing to allocate more time for other pursuits and activities.

That's one of the reasons we set out to find the best policy by running agent-based simulations. We modelled realistic scenarios and compared the performance of different metrics. Mainly comparing average waiting time (AWT), the average time until a passenger enters the elevator, and average time to destination (ATTD), which is the average total time it takes for the passenger to arrive at its destination. In addition, we measure the average crowdedness (ACE), more precisely the average amount of people per elevator.
We will in closer detail compare SCAN, LOOK, FCFS and SSTF. These policies are based on algorithms also used in disk scheduling, a field of science dedicating itself to finding an algorithm which minimizes disk arm movement while maximising data flow. The two subjects appear to share similarities, leading to the interchangeability of policies in certain instances. (Chen et al., 2021; Celis et al., 2014; Wang and Jiang, 2021)
Additionally, we propose our own algorithm called PWDP, which uses parameters to determine the best floor to move to. While also providing an attempt at improving these parameters using a random search algorithm.
Lastly, we implemented a visual representation and a plotter, which displays the simulation in real-time. This helps to verify the correctness of the algorithms and understand how the final analysis correlates with the actual situation in the simulation. These parts will not be covered in this report. However, in appendix A.1 readers can check out our project in item 1 and our extensive documentation of the whole project in item 2.

2 Model

2.1 Overview

2.1.1 Introduction

In this section, we will outline how our project is structured and more generally demonstrate the functionality of our simulation. Additionally, we will detail the constants we defined for our analysis.

2.1.2 Simulation

Everything concerning the simulation is stored inside the simulation class. It's being initialised from the main file alongside our visualisation class and live-plotting class. Alternatively, we initialise another instance of the simulation class when plotting from within the plotter class. The simulation contains most crucially, a building which is responsible for running the elevators and spawning passengers. Additionally, the simulation contains some delegate classes and a simulation statistics class tasked with gathering diverse data. The delegates will be called if any important events occur in the building, whilst the simulation-statistics class monitors these calls using listeners on the delegate classes.
Another essential part of the simulation is the time variable since our simulation is based on a step-by-step basis. The time variable defines at which timepoint in seconds, we currently

are. One second refers to one timestep. A simulation can be started by calling `run()` on the simulation object. As parameters, the runtime of the simulation must be specified either in seconds, minutes, hours, days or a combination of these. Once the simulation has started, it will iteratively call `step()` on the building and update the time variable until the end of the simulation is reached.

2.1.3 Building

The building incorporates classes for floors, elevators and a distribution. Floors are initialized on creation of the building. The only factor in this role is the floor amount. It's one of the constants we set for our analysis since we want to be able to compare results across different scenarios. If the floor amount would differ, it would lead us to no decisive conclusion about the strengths of different policies. That's why we keep our amount of floors constant at 10.
Similarly for the elevators, we decided to have a constant amount of two elevators. They will be initialized beforehand and passed as parameters to the building via the simulation. This is to simplify the choice of which policy the elevator should use. On initialisation, each elevator needs its own policy as an argument. This is because each elevator should be able to keep a running count of all the metrics it uses for decision-making. Lastly, the elevator also needs capacity as a parameter. According to various sources, the weight limit of a standard elevator ranges from 454 up to 2'722 kg. With elevators of low-rise buildings ranging from 907 to 1'134 kg. Thus by considering one person around 75 kg and a low- to mid-rise building, we arrive at a capacity of around 15 people. This is also the capacity we used for our simulation. (Bianco, 2022; TKElevator, 2022; Brown, 2023; Elevator Wiki, 2023)
This leads us to the only dynamic component, apart from the policies in the elevators, the distribution. Its purpose is to emulate realistic scenarios in buildings. One provides a time and it will return a list of passengers which should spawn at this time. For each passenger, we get additional information, about where it should spawn and which floor its target will be. This will be looked at in more detail, in section 2.1.5 and 2.2.

2.1.4 Execution Flow

The execution flow (see fig. 2.1) works as previously explained on a step-by-step basis. More precisely once the `run()` function was called, we iteratively called the `step()` function on the building, which will again trigger the `step()` functions of the elevators *one-by-one*. This is an important decision to make since we could also have opted to call all elevators at the same time while giving them only the common previous state. This however would complicate things, due to the elevators not being able to know, if a passenger already entered another elevator since the previous state.
In each step the building first calls `get_passengers_to_spawn()`, which returns a list of passengers with corresponding spawn and target floor. Consequently, these passengers will be spawned on the floor using the `spawn_passenger()` function. Lastly, the elevator `step()` function will be called, as stated before, on a *one-by-one* basis. Each elevator will then call its own policy and get a decision. If the elevator previously decided to open the doors, `remove_passenger()` will be called.

2.1.5 Distribution

Before moving on to the scenarios, it's important to understand how they work. Upon initialising a distribution one has to supply, as part of the parameters, a list of tuples of a time

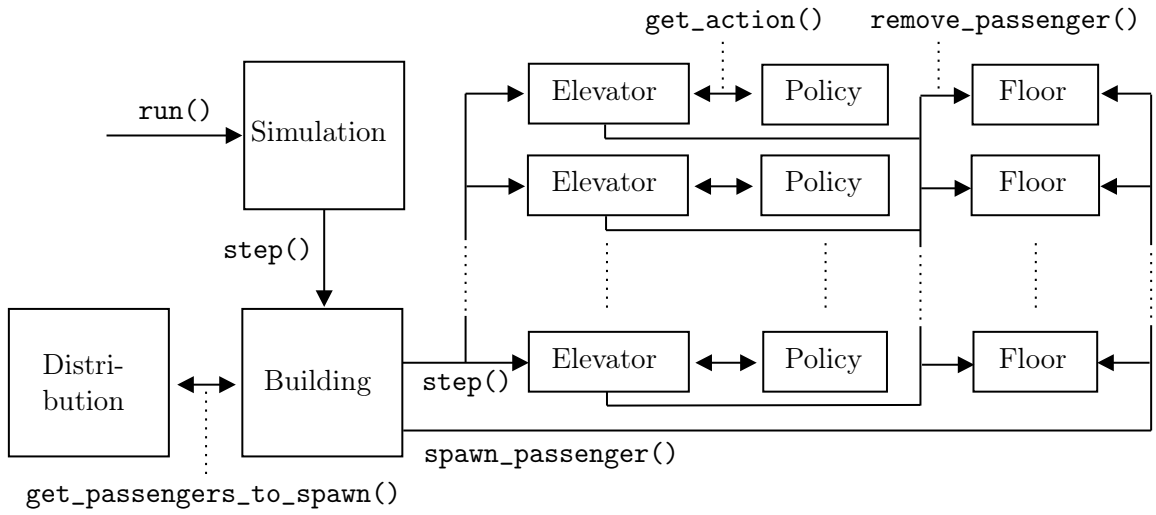


Figure 2.1: Execution Flow of Simulation

and a `FloorDistribution` object. These `FloorDistribution` objects take a list of weights as arguments, whereas each weight represents how frequently the floor of the same index should be chosen. This list should of course have the same length as the amount of floors in the building. So in other words, one defines the importance of each floor at some given time point. Additionally, one has to specify a passenger distribution of class `TimeDistribution`, which takes as arguments a list of tuples of time and amount of people which should spawn at the corresponding time.
After initialising, the distribution will precalculate all interpolated values between the defined times to improve performance. In the end, one can simply call the function `get_passengers_to_spawn()` to retrieve the locations of where passengers should spawn and which target they will persue.

2.2 Scenarios

2.2.1 Shopping Mall

The shopping mall scenario is the most simple scenario. The chance of a passenger spawning on a certain floor is equal for every floor. Thus the only thing that varies is the amount of people which spawn over time.
More notably, we try to emulate a shopping mall which opens at 8 a.m., peaks at lunchtime and closes again at midnight. This can be seen more clearly in Fig. 2.2a, which displays the passenger distribution, thus as described in section 2.1.5 the number of people spawning at a certain time.

2.2.2 Residential Building

The residential building has more complexity, the weights now differ per floor. Additionally, the spawn and target distribution are now also distinct.
This scenario should reflect the typical use of such a residential building. More precisely we expect people to leave the building for work at 6 a.m. This can be seen in fig 2.2b, where

more people, relatively to other floors, will spawn on floors 1 to 9. On the other hand, floor 0 will be chosen mostly as the target floor (see fig. 2.2c) to exit the building. An influx of people arriving at floor 0 and going towards the other floors, can again be seen at around 12:00. This should represent the lunch break, as people are coming back from work to eat. Then at 13:00, most people will start leaving again. Finally, peaking at 19:00 most people will return from work. It can also be seen, that similar to the shopping mall, there is also some activity through the night.

2.2.3 Rooftop Bar

Lastly, we have the rooftop bar scenario. The idea of this scenario is to test the policy's capabilities to handle a large number of people arriving at one floor and having mostly one floor as a target, the rooftop bar (or vice-versa).
So as one can see in Fig. 2.2d most people start spawning at 10 o'clock, peaking at 20:00. Similarly, it can be seen in Fig. 2.2e that these people will mostly go towards floor 9, our rooftop bar. At 22:00, supposedly when the rooftop bar closes, the situation flips and most people will spawn on the top floor and head towards the exit at the bottom floor. On random occasions, some people also seem to be heading towards the middle floors 1-8.

f

2.3 Policies

2.3.1 Introduction

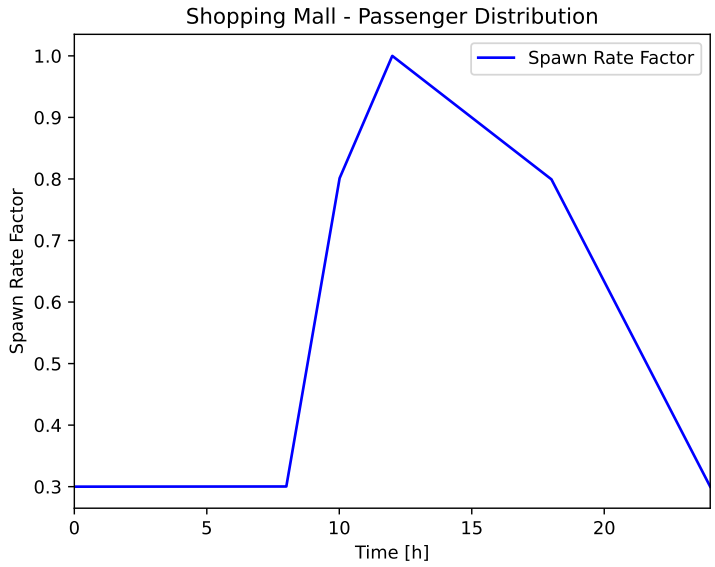
The actions of an elevator are decided by the policy class. Depending on the situation, the policy can choose one of five actions:

- `Up`: This signals the elevator that it should move up.
- `Down`: Indicates to the elevator that it should move down.
- `Wait`: Signals the elevator that it is currently waiting. Thus no passenger can enter.
- `WaitOpen`: Tells the elevator that it is currently waiting on a floor, with no advertised direction. Then, any passenger, regardless of their target floor, can enter the elevator.
- `WaitUp`: Indicates to the elevator that it is waiting while the advertised direction is up. Thus, any passenger headed in the same direction can enter.
- `WaitDown`: Indicates to the elevator that it is waiting while the advertised direction is down. Thus, any passenger headed in the same direction can enter.

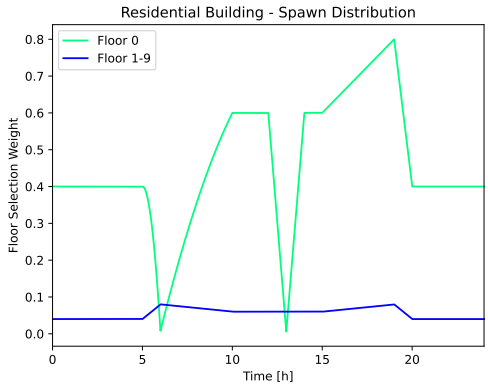
The decision is taken depending on various factors, which will presented in the following paragraphs.

2.3.2 SCAN Policy

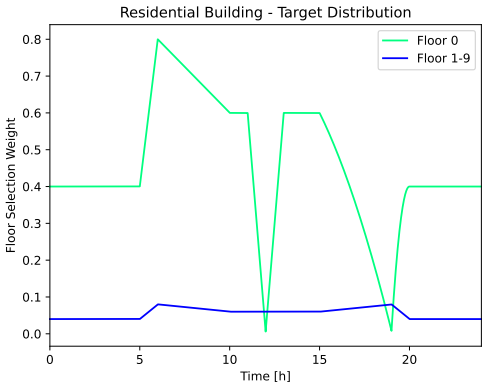
The SCAN Policy fully traverses the whole building until it reaches the highest or lowest floor. In which case the policy decides the change the direction from up to down or vice versa. It only stops when a passenger wants to exit the elevator on the current floor, or a passenger on the floor is also headed in the same direction. In those cases, the policy indicates to the elevator with `WaitUp` or `WaitDown` respectively, that passengers are allowed to enter or leave.



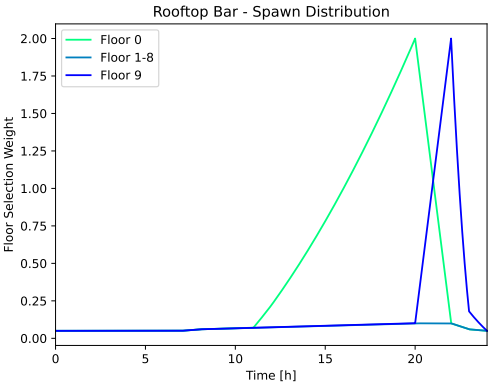
(a) Shopping Mall, Passenger Distribution



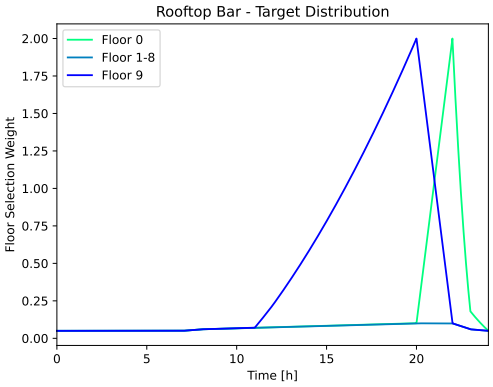
(b) Residential Building, Spawn Distr.



(c) Residential Building, Target Distr.



(d) Rooftop Bar, Spawn Distribution



(e) Rooftop Bar, Target Distribution

Figure 2.2: Distributions of Shopping Mall, Residential Building and Rooftop Bar

2.3.3 LOOK Policy

The LOOK Policy uses a similar mechanism as the SCAN Policy but can decide to change directions, without having to visit the highest or lowest floor. (Wang and Jiang, 2021) As before, the policy respects the movement of the elevator. Thus the algorithm will return up or down respectively, until a passenger wants to exit the elevator on the current floor, or a passenger on the floor is also headed in the same direction. In those cases, the policy indicates to the elevator with `WaitUp` or `WaitDown` respectively, that passengers are allowed to enter or leave. The policy now decides to switch from moving up to down, when the elevator has reached the topmost request. Giving the chance to not unnecessarily visit the highest floor. Similarly, when the elevator is currently moving downwards and the lowest request is reached, it decides to change direction to up, if there are still requests.

2.3.4 FCFS Policy

The first-come-first-serve (FCFS) policy, addresses the request within the elevator in the order they were raised. It will go directly to the targeted floor by whichever passenger turn it currently is, by returning the according direction up or down. When the desired floor is reached, it signals with `WaitUp` or `WaitDown`, if there are still requests in the queue, otherwise it will signal `WaitOpen`. This procedure will be repeated until the elevator is empty. In that case, the policy decides to head directly to the request, which is closest to the current floor of the elevator. The whole algorithm follows the idea of the traditional FCFS algorithm (Javed and Khan, 2000), only deviating in the scenario when the elevator is empty.

2.3.5 SSTF Policy

The shortest-seek-time-first works similarly to the FCFS policy. It addresses the request within the elevator in the order of increasing distance. The elevator will always go to the floor, that is the closest to the current floor and has a request inbound. (Javed and Khan, 2000) Before arriving at the desired floor, it will evaluate the next target and indicate the direction with `WaitUp` or `WaitDown` respectively. Thus, potential passengers heading in the same direction can enter the elevator. This procedure will be done until the elevator is empty. In this case, the policy decides to head directly to the request, which is closest to the current floor of the elevator.

2.3.6 PWDP Policy

The parameterized weighted decision policy (PWDP Policy) is our own attempt to tackle the uncovered flaws of the previous policies (for more, see section 4).
When an elevator needs to make its next decision, it calculates a score for each floor in the building. The floor with the maximal reward will then be set as the next target to travel to. For each floor $i \in \{0, \dots, \#\text{floors} - 1\}$,

$$\textbf{Score}[i] = \frac{s_1 + s_2 + s_3 + s_4}{\max\{1, s_5 + s_6\}}.$$

The score function consists of multiple components $s_1, ..., s_6$. Every part addresses a different aspect to optimize for. They all also possess a parameterized weight to change the relative impact the parameter has towards the final score.

$$s_1 = \textbf{flButtonW} \cdot \text{flButtonPressed}[i]$$

Component s_1 is a constant mask, on which floors passengers are waiting for an elevator. It should encourage elevators to move towards waiting passengers.

$$s_2 = \mathbf{flButtonW} \cdot \mathbf{flButtonTimeW} \cdot \text{flButtonPressed}[i] \cdot \frac{\text{flButtonTime}[i]}{\max\{1, \text{maxFlButtonTime}\}}$$

Component s_2 is the normalized time of how long passengers have been waiting for an elevator to arrive. Passengers waiting for a longer amount of time receive a higher score. It should encourage elevators to move towards passengers who have waited for a long time.

$$s_3 = \mathbf{elButtonW} \cdot \text{elButtonPressed}[i]$$

Component s_3 is a constant mask of the target floors from the passengers currently in the elevator. It should encourage elevators to move towards destination floors.

$$s_4 = \mathbf{elButtonW} \cdot \mathbf{elButtonTimeW} \cdot \text{elButtonPressed}[i] \cdot \frac{\text{elButtonTime}[i]}{\max\{1, \text{maxElButtonTime}\}}$$

Component s_4 is the normalized time of how long passengers have been inside the elevator. Passengers waiting for a longer amount of time receive a higher score. It should encourage elevators to move towards floors that passengers have targeted for a long time.

$$s_5 = \mathbf{competitorW} \cdot \sum_{j \neq \text{elIndex}} \#\text{floors - distToOtherElevator}[j]$$

Component s_5 measures the distance between the elevators compared to the height of the building. A higher value represents the elevators being closer together. Because s_5 is in the score function in the denominator, minimizing s_5 yields a higher score. It should encourage elevators to choose different targets and avoid spontaneous alignment.

$$s_6 = (\mathbf{distanceW})^{\mathbf{distanceExponent}} \cdot |\text{currentFloor} - i|$$

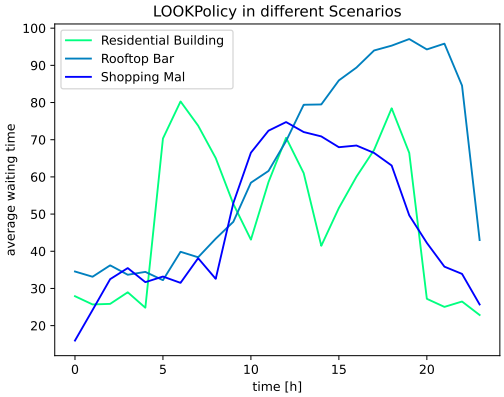
Component s_6 is the distance between the elevator and the floor. s_6 is in the score function in the denominator, minimizing s_6 yields a higher score. It should encourage elevators to prefer closer targets.

3 Results

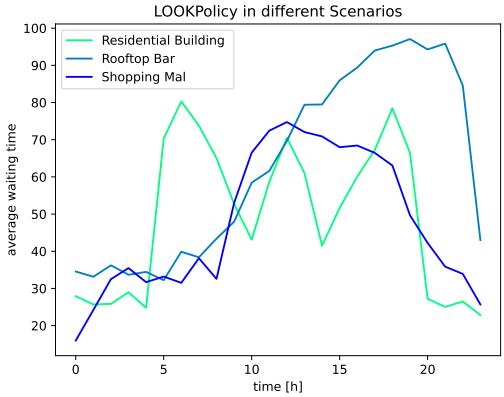
3.1 Policies

In this subchapter, we are focusing on some inefficiencies of already existing policies, in the hope that we can learn from them. For that, we are going to benchmark the four introduced policies, in the three scenarios with the two metrics average waiting time (AWT) and average time to destination (ATTD). The average crowdedness (ACE) is a metric, which is also looked at but in less detail. The curves defined by the three metrics - AWT (Fig. 3.1), ATTD (Fig. A.3) and ACE (Fig. A.4) - follow the graph of the total people spawned in the entire building (Fig. A.1) quite accurately, only deviating strongly in some special circumstances. The introduced AWT and policies yield the graphs presented in figure 3.1.

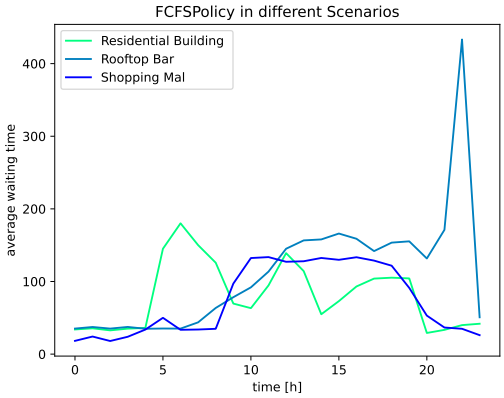
The table 3.1 displays the values that achieve the maximum in each policy and scenario, as indicated by the corresponding figure 3.1. It holds for all metrics (Fig. A.1 and Fig. A.2)



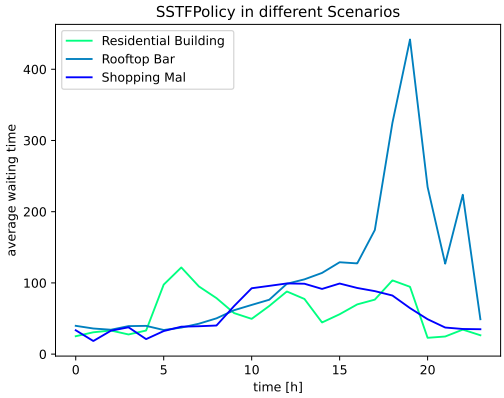
(a) SCAN Policy



(b) LOOK Policy



(c) FCFS Policy



(d) SSTF Policy

Figure 3.1: Different policies in various scenarios, Average Waiting Time per hour

that the maximum is attained at almost the same time. In the shopping mall scenario, the maximum is attained in $[11, 12]$, whereas the residence building produces its maxima in $[6, 7]$. The only scenario where the extreme points are more distributed is in the rooftop bar scenario where they can range from 18 to 22.

Policy	shopping mal		residence building		rooftop bar	
	max value	time	max	time	max	time
LOOK	74.75	12	80.29	6	97.07	19
SCAN	80.2	12	89.39	7	96.48	18
SSTF	99.24	12	121.72	6	441.48	19
FCFS	133.59	11	180.12	6	433.26	22
PWDP	46.52	11	57.46	7	48.94	21
PWDP (optimised)	47.19	11	54.47	7	48.51	21

Table 3.1: Collection of all maximal AWT values in the scenarios

The SCAN Policy (Fig. 3.1a and Fig. A.3a) and both PWDP Policies (Fig. A.2, Fig. A.3e and Fig. A.3f) are the only algorithms which show strong volatilities. They are found

in the shopping mall scenario from midnight to 5 a.m.

Figure 3.2 showcases the performance of the policies relative to each other. In the shopping mall scenario, the SCAN Policy has the worst efficiency, when it comes to low-intensity situations. However, when the intensity rises, it remains relatively stable, only to be outperformed by LOOK and PWDP. SCAN has the same tendencies in the rooftop bar example. The FCFS policy gets outperformed by all other algorithms when the intensity peaks in the shopping mall. However, in the rooftop bar scenario, the worst performance in the high intensity is delivered by the SSTF Policy. Figure 3.3 showcases the overall improvements in the optimisation of the PWDP Policy. The said changes are only seen in the high-intensity scenario.

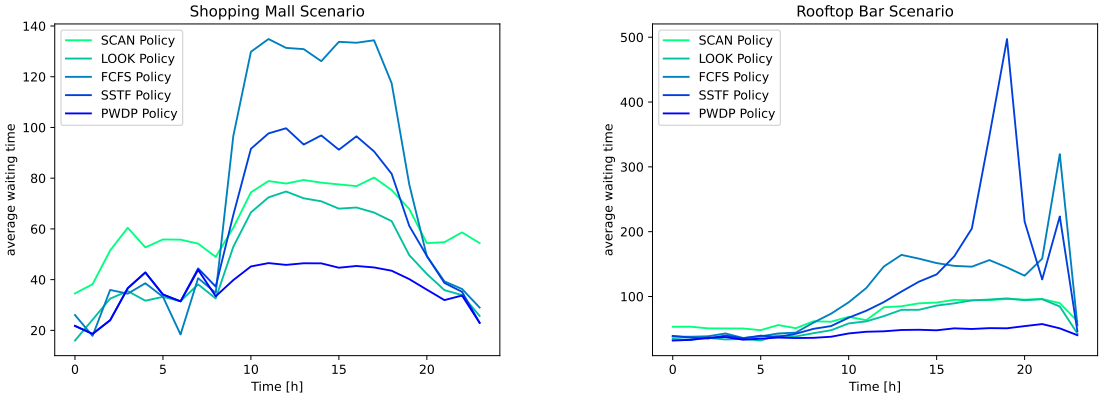


Figure 3.2: AWT over time of each policy. Shopping mall scenario (left) and Rooftop bar scenario (right).

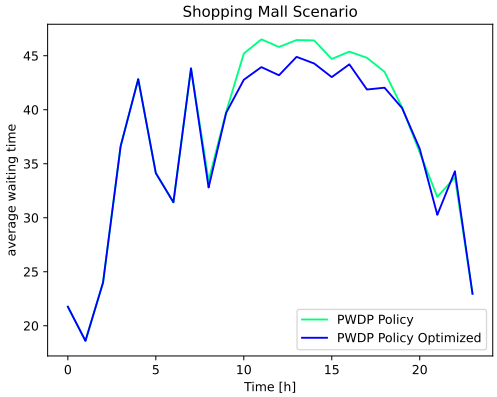


Figure 3.3: Comparison between Initial PWDP Policy and the optimized version in the shopping mall scenario.

3.2 PWDP Optimisation

The PWDP Policy with all weights set to 1 already dwarfs all other analyzed policies, as shown in Figure 3.2. While there isn't much of a notable difference during low traffic, the PWDP Policy inversely stays stable in high traffic.

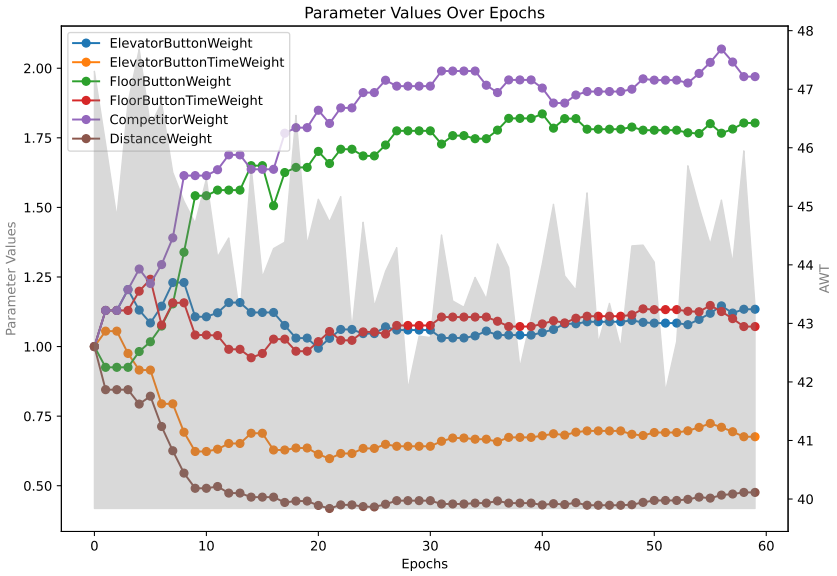


Figure 3.4: Change of the parameters (left y-axis) of the model over 60 iterations (x-axis). The greyed-out area represents the AWT (right y-axis) of the validation set.

We optimized the model parameters using a random search algorithm with AWT as the parameter to minimize. Over many iterations, we randomly perturbed the weights of the model and the average waiting time of the new model was then compared to the current model. The current model is replaced if a performance increase is recognized.

The model was initialized with all parameters set to 1. In Figure 3.4 we can see a downward trend of the AWT in the first few iterations. The model then stabilizes at around 5 to 10 Percent improvement compared to the initial model (see gray area in Fig. 3.4).

3.2.1 Distance Weights

The distance parameter shows a reduction in weight of around 50%. This pattern is also reproduced in the phase diagrams (Figure 3.5) of the Distance Weight compared with other weights. They too set the optimal Distance Weight at 0.5, much less than any other parameter. There is also an unequivocal degradation of the model for higher values.

3.2.2 Competitor Weights

The Competitor Weight parameter shows a clear increase in its base value, as seen in Figure 3.6. Increasing the weight by a large amount (Figure 3.6b) does not have a large negative impact on the AWT.

3.2.3 Elevator Time Weight

Figure 3.4 also shows a decrease in the Elevator Time Weight, while the Floor Button Weight inversely increases. This behavior is also reflected in their phase diagram (Figure 3.7a), where too small values of the Floor Button Weight will break the policy but the Elevator Button Time Weight stays relatively unaffected. In Figure 3.7b, the extreme values are cut off and

we also see that the optimum correlation between these two parameters is off the diagonal, biased towards the side of the Floor Button Weight.

4 Discussion

Generally, what is true for all comparisons, regardless of the policy or scenario, is that ATTD is always higher than AWT. This is given by the fact that the time waiting, is contained in the total travel time. Furthermore, the metric ACE also follows in almost all the cases the distribution of the ATTD. To compare the ATTD and ACE please refer to the section A.2.3 and A.2.4 in the appendix. The only case where this does not hold is in the low-intensity scenario, where a certain volatility comes into play. Since in the low-intensity scenario, the average crowdedness is either zero, when the elevator is empty, or one, when a passenger is transported, one can conclude that this difference in the initial low-intensity scenario comes from the changing AWT and ATTD.

4.1 SCAN Policy

The SCAN policy is the simplest policy implemented in this project. Besides its simplicity, it performs reasonably well, only being outperformed in high-intensity by its improved version, in our project called LOOK policy, or the parametrized policy. However, the policy also introduces volatility in low-intensity cases. For both metrics, the limited influx of passengers in these regions renders the entire system reliant upon the arbitrary placement of individuals, a factor that exerts a significantly greater influence in comparison. Consequently, a passenger may emerge near the elevator, expediting the pickup process. Conversely, an unfavourable scenario may transpire, in which the elevator must traverse the entire building before ultimately reaching the passenger. Since ATTD has an inherently larger baseline - namely the distance from spawn- to the target floor - it relativises the fluctuations, resulting in lower volatility.

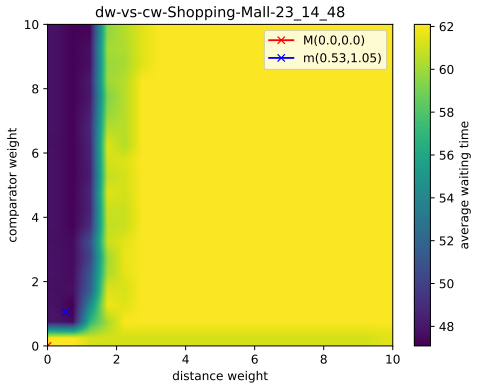
4.2 LOOK Policy

Compared to the SCAN Policy, the LOOK Policy is a bit harder to implement but comes with some efficiency improvements. Besides the parametrised policy, it is the most capable policy and most resilient, especially in high-intensity situations.

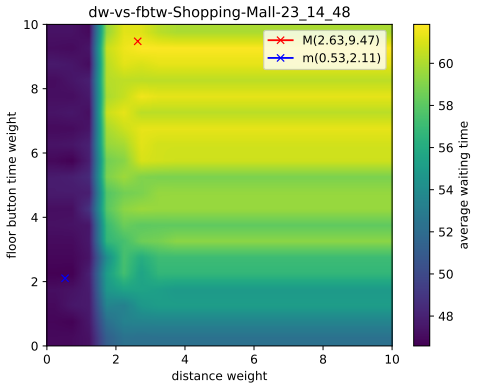
One small improvement comes in the low-intensity situation, where the policy can decide to switch directions. This makes the algorithm less prone to random placements of passengers and their targets, reducing the volatility. It slightly outperforms SCAN in high-intensity,which showcases that changing directions looses significance with the increase of total amount of passengers in a building.

4.3 FCFS Policy

The FCFS Policy follows a more complex algorithm and has the upper hand when it comes to low-intensity scenarios. However, in the rooftop bar example, it more than quadruples the average waiting time achieved by the LOOK Policy (Tab. 3.1). However, the average time to destination is only tripled (Tab. A.1).

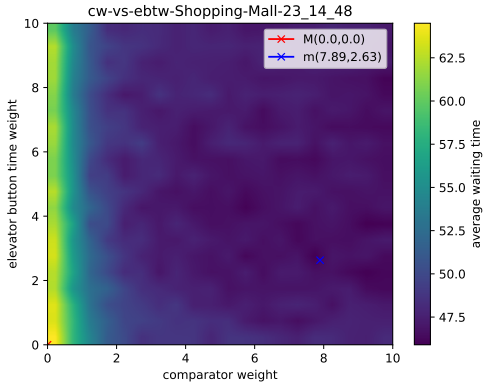


(a) Comparison of Distance Weight (x-axis) to Comparator Weight (y-axis).

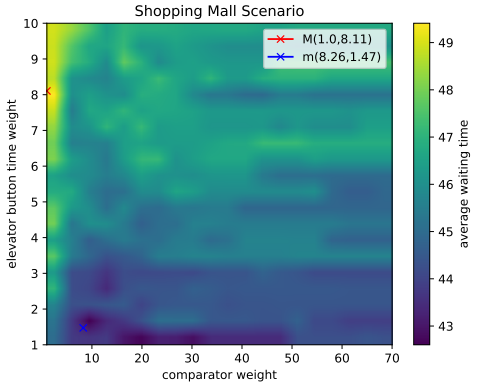


(b) Comparison of Distance Weight (x-axis) to Floor Button Time Weight (y-axis).

Figure 3.5: Phase diagrams of Distance Weight compared to other weights.

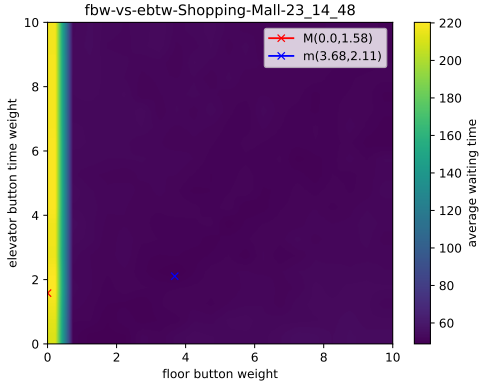


(a) Competitor Weight interval from [0,10]

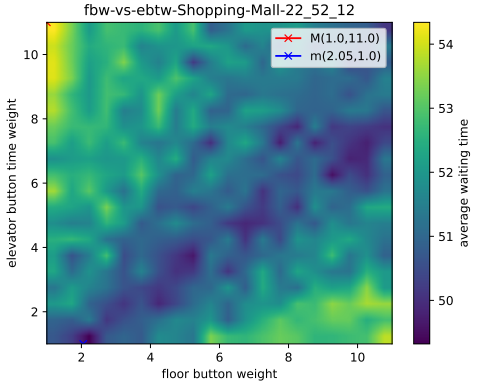


(b) Competitor Weight interval from [0,70]

Figure 3.6: Phase diagram comparing Comparator Weight (x-axis) to Elevator Button Time Weight (y-axis).



(a) Weight interval from [0,10]



(b) Weight interval from [1,11]

Figure 3.7: Phase diagram comparing Floor Button Weight (x-axis) to Elevator Button Time Weight (y-axis).

This inefficiency in the rooftop bar at 8 p.m., can be credited to the search pattern by an empty elevator, which always looks for the closest floor. In this scenario and time, most passengers occur on the upper floor, heading to the exit at the ground level. Thus an elevator going from the top floor will be emptied when it reaches the lowest floor. Then, one other passenger will be prioritized over the possibly many other passengers exiting the rooftop bar. This can keep the elevator on the lower levels until at some point no other individuals are close by or it gradually ascends, floor by floor. In both cases, the AWT, as well as ATTD, will drastically increase.

4.4 SSTF Policy

This policy now fully schedules levels by prioritizing the target closest by. This approach has shown to be quite stable in the shopping mall scenario as well as residence building (Fig. 3.1d), only closely missing the achieved metrics set by the LOOK, seen in figure 3.1b and SCAN Policy, seen in figure 3.1a. However, the distance-avoiding approach has proven to be counterproductive in the case of the rooftop bar, yielding an AWT of over 400 (Tab. 3.1).

The inefficiency in the rooftop bar at 8 p.m. can be credited to the search pattern by an empty elevator, which always looks for the closest floor. In this scenario and time, most passengers occur on the upper floor, heading to the exit at the ground level. This effect is intensified by the new possibility of additional passengers, who can enter at any floor and have higher precedence if they are headed to a floor close by. The FCFS Policy would address requests by passengers with increasing time of demand, giving it an edge over the SSTF Policy.

4.4.1 Spontaneous Synchronous Behaviour

A phenomenon, that can occur in all introduced policies is spontaneous alignment. With a reasonably chosen number of elevators, they can align at some point and then stay either partially or fully aligned for a longer time. This effect jeopardizes the fine granularity offered by multiple elevators, effectively reducing the system's overall adaptivity. Since this project only analysed systems in which all elevators have the same policy, it is a phenomenon, which occurred with various scenarios and policies. Given that policy makes a deterministic choice, for a given environment, the same policies will make the same decision. Thus, two elevators on the same level will inevitably decide on the same action. This situation can occur, when one elevator is already picking up passengers from one floor, while others are approaching, effectively decreasing the distance between both elevators and thus increasing the chance of taking the same action.

4.5 PWDP Policy

While the PWDP Policy repeatedly outperforms the other policies during high traffic, there isn't much of a notable difference during low traffic (Figure 3.2). The noisier data at off-peak times can be explained through the higher amount of randomness involved. If there are next to no travellers, it has a much higher impact on where a passenger spawns in relation to a waiting elevator. Additionally, with only a single assignment the actual behaviour of the elevator is an easily solved problem, without the need for any complicated policy.

4.6 PWDP Optimisation

According to Figure 3.3, the optimization gives a slight edge over the initial PWDP Policy. As seen in Figure 3.4 the main contributors towards this change are Distance Weights, Elevator Time Weights and Competitor Weights.

4.6.1 Distance Weights

One explanation for the sharp decline in Distance Weights is that the elevators may get stuck in their local environment and won't be able to listen to requests from passengers far away. Therefore, some floors would only seldomly be visited and passengers can stack up there.

4.6.2 Competitor Weights

The reason for the upsurge in the weight may be that increasing the Competitor Weight inversely decreases the impact of Distance Weight, which has already been shown to be detrimental. Additionally, elevators trying to maximize their distance from each other counteract the spontaneous synchronous behaviour mentioned in subsection 4.4.1.

4.6.3 Elevator Time Weight

One contribution to this pattern is the goal we optimized for. As we tried to minimize the waiting time instead of the total time, it naturally became much more important to pick up passengers waiting on a floor than to bring them to their destination. The Elevator Button Time Weight parameter only indirectly impacts the model when the elevator gets too crowded and no passengers fit in anymore.

5 Conclusion and Outlook

Amongst SCAN, LOOK, FCFS and SSTF. Both LOOK and SCAN are performing the best in all metrics, despite being simpler to implement. Whereas LOOK is performing slightly better. Amongst all policies, PWDP is performing better than the other policies in all scenarios. Although further improvements using the random search algorithm have been rather marginal, we found certain parameters to be more influential than others. Mainly the competitor weight and floor button weight. Others like elevator button time weight and, more extremely, the distance weight have demonstrated diminished influence.
In future work, one could try to find more optimal parameters using other techniques less prone to local minima. Moreover one could add the ability for passengers to indicate a target floor, further improving the capabilities of the policies. Another change one could make is the addition of random perturbations, simulating situations such as too many people blocking the doorway or service personnel with specialised equipment occupying a greater portion of the elevator capacity. Since our model is limited by a local approach, one can also implement a global controller, which can orchestrate a set of elevators individually.

A Appendix

A.1 Link to Project and Documentation

1. Project: `https://github.com/Silvan-M/ElevatingEfficiency`
2. Documentation: `https://silvan-m.github.io/ElevatingEfficiency/index.html`

A.2 Additional Plots

A.2.1 Absolute Spawning Curve

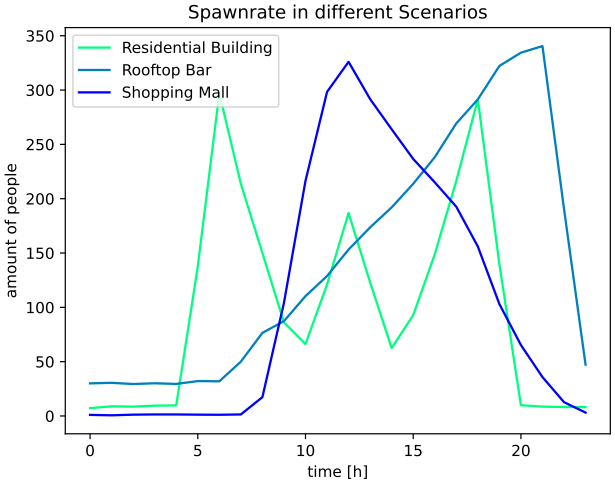


Figure A.1: Absolute amount of passengers spawning in the entire building over time

A.2.2 Average Waiting Time

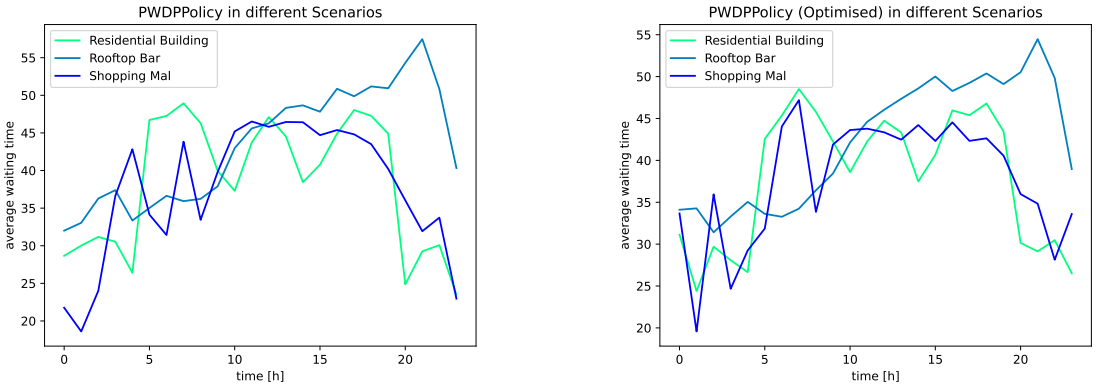
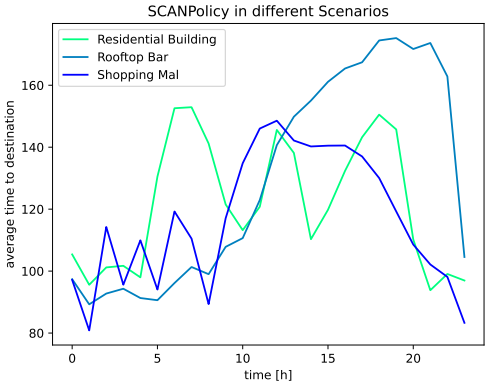
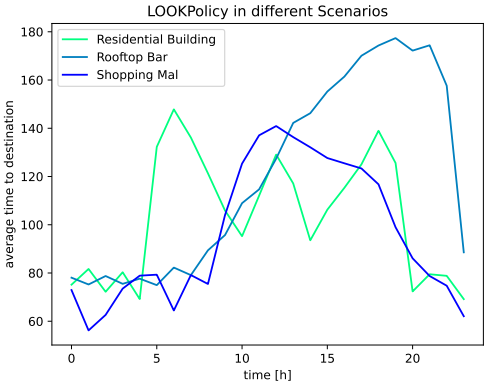


Figure A.2: AWT over time of the initial (left) and optimised (right) PWDP Policy.

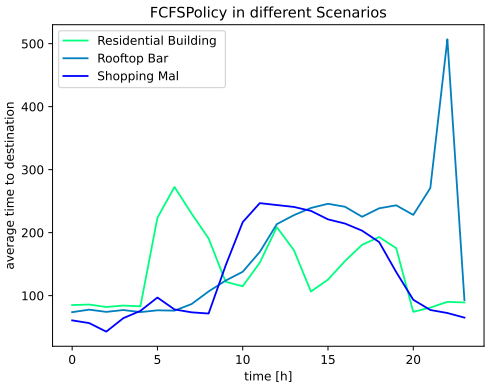
A.2.3 Average Time To Destination



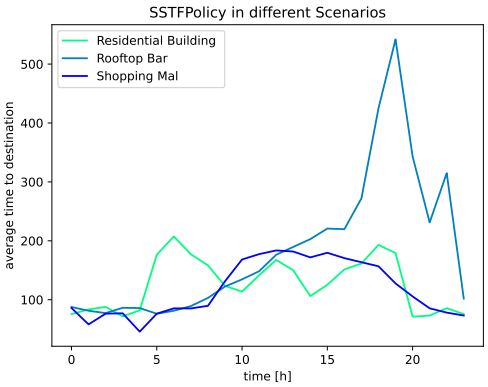
(a) SCAN Policy



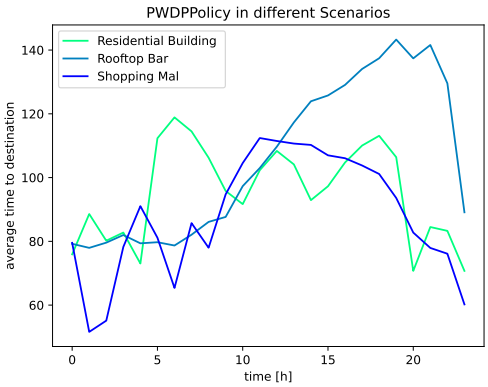
(b) LOOK Policy



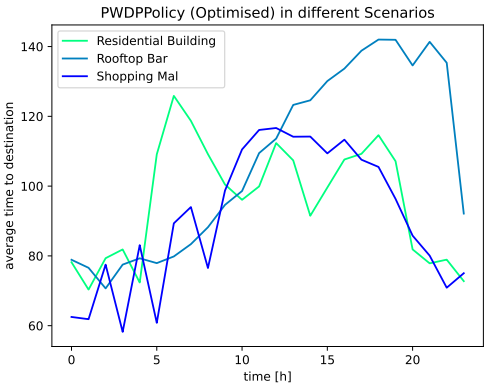
(c) FCFS Policy



(d) SSTF Policy



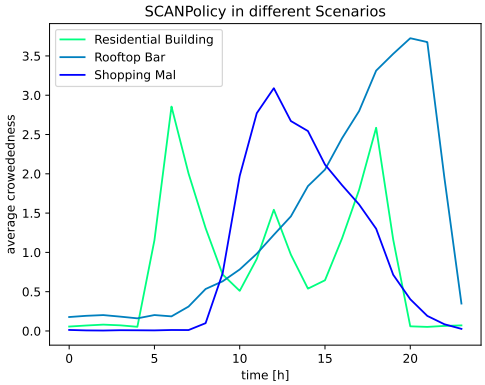
(e) PWDP Policy



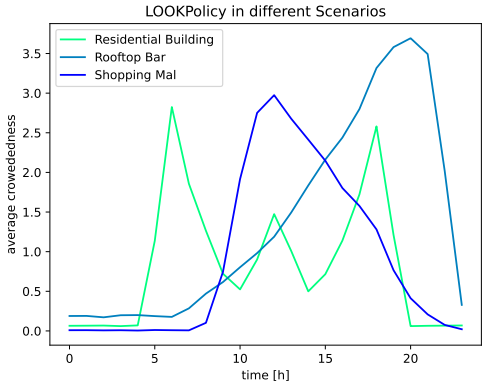
(f) Optimised PWDP Policy

Figure A.3: Different policies in various scenarios, Average Time To Destination per hour

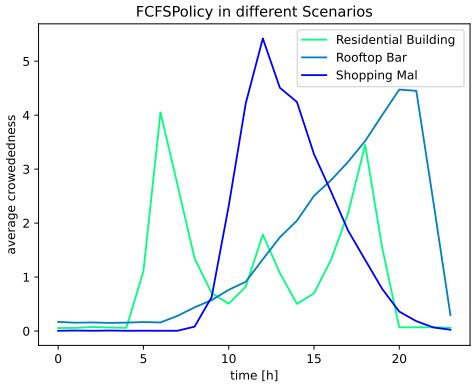
A.2.4 Average Crowdedness



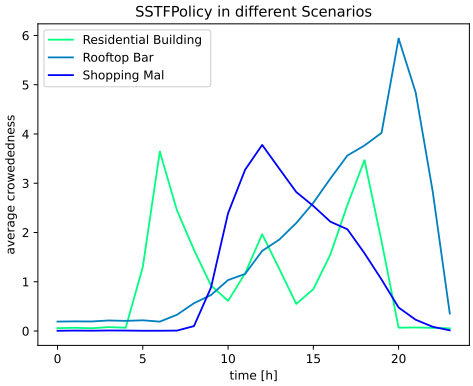
(a) SCAN Policy



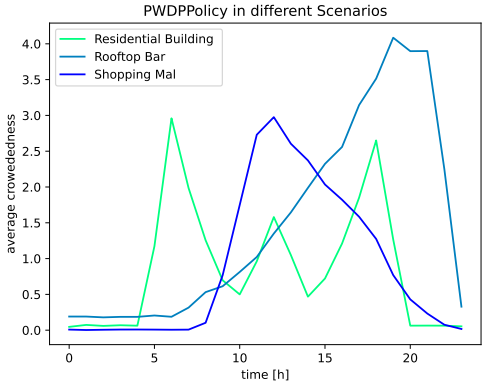
(b) LOOK Policy



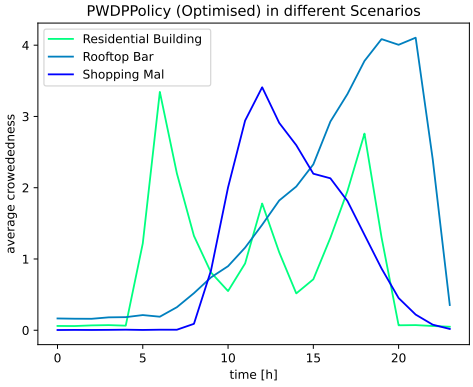
(c) FCFS Policy



(d) SSTF Policy



(e) Initial PWDP Policy



(f) optimised PWDP Policy

Figure A.4: Different policies in various scenarios, Average Crowdedness per hour

A.3 Additional Tables

A.3.1 Average Time to Destination

Policy	shopping mal		residence building		rooftop bar	
	max value	time	max	time	max	time
LOOK	140.91	12	147.85	6	177.38	19
SCAN	148.56	12	175.23	7	175.23	18
SSTF	183.66	12	207.41	6	542.03	19
FCFS	245.83	11	272.42	6	506.87	22
PWDP	112.41	11	118.89	6	143.29	19
PWDP (optimised)	116.65	12	125.87	6	142.0	18

Table A.1: Collection of all maximal ATTD values in the scenarios

A.3.2 Average Crowdedness

Policy	shopping mal		residence building		rooftop bar	
	max value	time	max	time	max	time
LOOK	2.97	12	2.82	6	3.96	20
SCAN	3.09	12	2.86	6	3.73	20
SSTF	3.78	12	3.64	6	5.94	20
FCFS	5.42	12	4.05	6	4.47	20
PWDP	2.98	12	2.96	6	4.09	19
PWDP (optimised)	3.41	12	3.34	6	4.11	21

Table A.2: Collection of all maximal ACE values in the scenarios

Bibliography

Bianco, A. (2022). How much weight can a elevator hold? [comprehensive answer]. https://www.cgaa.org/article/how-much-weight-can-a-elevator-hold.

Brown, E. (2023). How much weight can a standard elevator hold? https://www.aboutmechanics.com/how-much-weight-can-a-standard-elevator-hold.htm.

Celis, J. R., Gonzales, D., Lagda, E., and Rutaquio Jr, L. (2014). A comprehensive review for disk scheduling algorithms. *International Journal of Computer Science Issues (IJCSI)*, 11(1):74.

Chen, W., Zheng, B., Liu, J., Li, L., and Ren, X. (2021). A real-time matrix iterative optimization algorithm of booking elevator group and numerical simulation formed by multi-sensor combination. *Electronics*, 10(24).

Elevator Wiki (2023). Capacity. https://elevation.fandom.com/wiki/Capacity.

Javed, M. and Khan, I. (2000). Simulation and performance comparison of four disk scheduling algorithms. In *2000 TENCON Proceedings. Intelligent Systems and Technologies for the New Millennium (Cat. No.00CH37119)*, volume 2, pages 10–15 vol.2.

TKElevator (2022). How is elevator capacity calculated? https://www.tkelevator.com/us-en/company/insights/how-is-elevator-capacity-calculated.html.

Wang, Y. and Jiang, T. (2021). Design and plc realization of elevator control based on look algorithm. *Journal of Physics: Conference Series*, 1754(1):012082.