Redesign Staircases to Optimize Trains' Unloading Time

Team # 4671

November 16, 2014

Summary

During our investigation of this situation, our team aimed to build an optimization model to analyze and minimize the unloading time of trains. To achieve this goal, we put forward two models, the mathematical model based on queuing theory and the simulation model based on cellular automaton model. Two models proved each other very well.

We first built our mathematical model, in which the exiting process is divided into four time periods: alighting, walking on platforms, queuing, and walking on staircases. We used the knowledge of former research and d/d/s queuing model in order to calculate the time of the four periods. Using this model, we determined that when there are two staircases, each stair of which contains 2 passengers, the total unloading time for one train is around 261 seconds, while that for two trains is approximately 498 seconds.

In addition, we adopted a simulation approach based on cellular automaton model. Different from the mathematical model, the simulation model simulated several periods of unloading processes as a complete process, and considered the effects of congestion on the moving speed and moving rule of passengers. In addition, the model could visualize the simulation process. The simulation model gave the approximately same result as the mathematical model: the unloading time for one train is 251 seconds, while that for two train is 498 seconds. The results convinced us that both of our models are reliable and accurate.

To understand the effect of different factors, we changed variables such as the number of trains arriving, the location and number of staircases, and the capacity of each staircase to see their impact on unloading time. Running our simulation model, we found that: the most effective way to minimize the unloading time is to increase the number of staircases; the capacity of the staircase is also a very important factor; the location of staircases does not affect the unloading time very much.

Lastly, we wrote an letter to the director of transportation to recommend that our model be used to help design platforms' layout and schedule transit system. Although the context of our problem is simple, our simulation model can be applied to design of any complex transfer stations by slightly adjusting some parameters.

1 Introduction and Problem Restatement

1.1 Introduction

Nowadays, more and more people choose to travel by public transportation. As an important part of public transportation,the train station system is playing a key role in people's everyday life. Meanwhile, the popularity of traveling by train puts greater pressure on a station's operation and brings about many new problems such as emergency response to fires, explosions and terrorist attacks. Therefore, how to properly design a station's layout to minimize passengers' exiting time becomes a vital problem to the designers and operators of public transit systems. We wanted to help solve these problems. For this purpose we built a mathematical model and a simulation model to analyze the unloading time in different situations. We analyzed the relationships of different parameters in order to determine the key factors for unloading time. Our model can help operators of public transportation system to optimize their design and scheduling, provide better services and reduce accident rates.

1.2 Problem restatement

We wanted to build a model to analyze and optimize the time traveled from the train to the street level exit of the station. We addressed situations with one fully occupied trains and two full occupied trains. Also, we adjusted the moving speed of passengers and number of passengers to see the impact. We then took other factors such as the location of the staircases, the number of staircases and the capacity of the staircases into consideration.

The most important factor is the total time it takes to evacuate all the passengers.

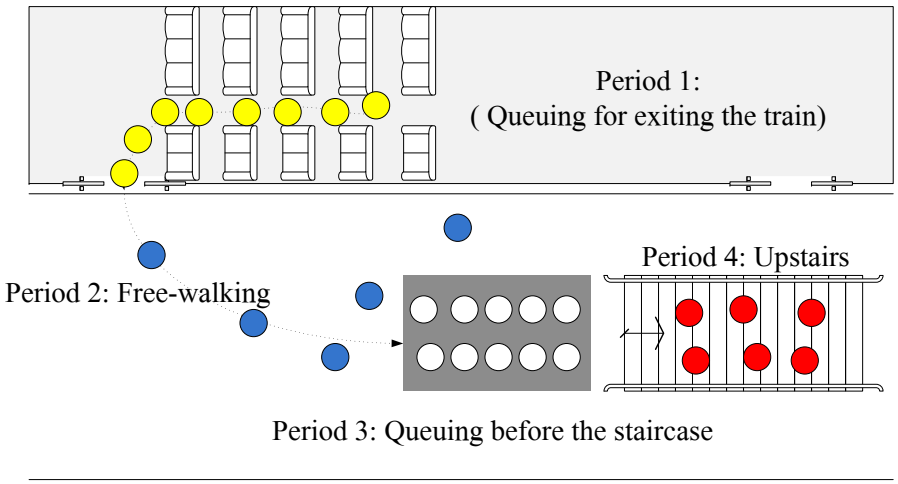


Figure 1: The exiting process

2 Definitions, Assumptions and Variables

2.1 Definitions

As shown in Fig.1, the process what a passenger exits the station includes four periods:

1. waits at the queue inside a car to exit the train;
2. walks towards the staircase;
3. waits in the queue to go upstairs;
4. moves upwards on a staircase.

Now we define some terms what will be used in the whole paper.

unloading time: the time when the last passenger walks onto the foot of a staircase. In this paper we are mainly researching unloading time.
evacuation time: the time when the last passenger leave a staircase to exit the station. We also call it exiting time in this paper.

2.2 Assumptions

In order to streamline our model, we have made several key assumptions.

Assumption: Each car of a train can contain 90 people.
Justification: There are 5 seats in each row in the given information. The seating plans[1] searched from the Internet show that one car of a train holds around 16 to 19 rows of seats, so we assume one car holds 18 rows in this paper.

Assumption: The staircases are distributed uniformly along the longitudinal direction of the platform.
Justification: The uniform distribution can make passengers' walking distance the shortest.

Assumption: When a train arrives at the station, all passengers on the train will unload.
Justification: From the given information, the station is a central station which is the destination or transfer station of most people.

Assumption: Each queue in cars waiting to exit the train is a single queue. Passengers get off the train by leaving the head of queue one by one in order.
Justification: There are 5 seats in each row, so the center aisles in cars are narrow.

Assumption: Passengers would choose only one from two exits of a car to leave. They are distributed equally to two queues of the car.
Justification: Passengers always select the nearest exit.

Assumption: There are no people waiting on the platform to get on the train when the train arrives.
Justification: No given information mentions the passengers waiting for the train.

Assumption: When a passenger disembarks from train, he chooses one staircase as his moving target based on crowding level and distance and

never changes the target.
Justification: Most people move in this way. Changing a moving target consumes both time and effort.

Assumption: We do not differentiate the moving speed of the passengers with different ages and genders.
Justification: Although this assumption simplifies the real world, it could still satisfy the requirement of the models which estimate the average evacuation time.

Assumption: Once a passenger reaches the upper level of the station, he exits the complex.
Justification: From the given information, to exit a station, passengers must exit the car, and then make their way to a staircase to get to the next level.

2.3 Variables

- N is the number of trains arriving at the same time. N is 1 or 2.
- n is the number of car to a train. A full train contains $90n$ passengers.
- d is the length of each car in meters.
- p is the length of the platform in meters. For simplicity we assume $p = d * n$.
- w is the width of the platform in meters.
- q the number of stairs in each staircase.
- k is the number of people that each staircase can accommodate. k is an integer greater than one.
- c is the number of passenger a single stair can accommodate. $c = k/q$. c is an integer.
- s is the number of staircases in the platform.

- v_p is the moving speed of the passengers on the platform in meters per second.
- λ is the input rate of the queue before each staircase in persons per second.
- μ is the output rate of each staircase in persons per second.
- t_0 is the time for all passengers to get off the train in seconds.
- t_1 is the time for a passenger on the platform to walk to the entrance queue of the stairway in seconds.
- t_2 is the maximum time for a passenger to wait at the queue before the stairway in seconds.
- t_3 is the time for a passenger to move on the stairway in seconds.
- t is the max time for a passenger to exit the station in seconds. We named it evacuation time. $t = t_0 + t_1 + t_2 + t_3$.
- $t^{'}$ is the max time for a passenger to walk on a staircase in seconds. We named it unloading time. $t^{'} = t_0 + t_1 + t_2$.

3 Mathematical Model

3.1 Modeling Alighting Time

We modeled the alighting time first. Existing works have done research on alighting and boarding time of the rail traffic passengers. The research[2] completed by Hong Kong Polytechnic University has built a mathematical model to estimate the dwelling time of trains in relation to the crowding conditions at the stations, which is given as follows:

$$DT = C_0 + C_1 * A_l + C_2 * B_o$$

where:
DT is dwelling time of train(s);
A_l is the number of alighting passengers per train, and
B_o is the number of boarding passengers per train;
C_0 is a constant (s), C_1 and C_2 are coefficients.

Since there is no given information of the number of boarding passengers for this paper, we simplify the model to research the relation between alighting time and the number of alighting passengers. The research[3] completed by Shouhua Cao has shown that there is a linear relation between alighting time and the number of alighting passengers in Beijing rail transit as shown in Fig. 2. The data what we observed from Shanghai subway stations shows the similar tendency.

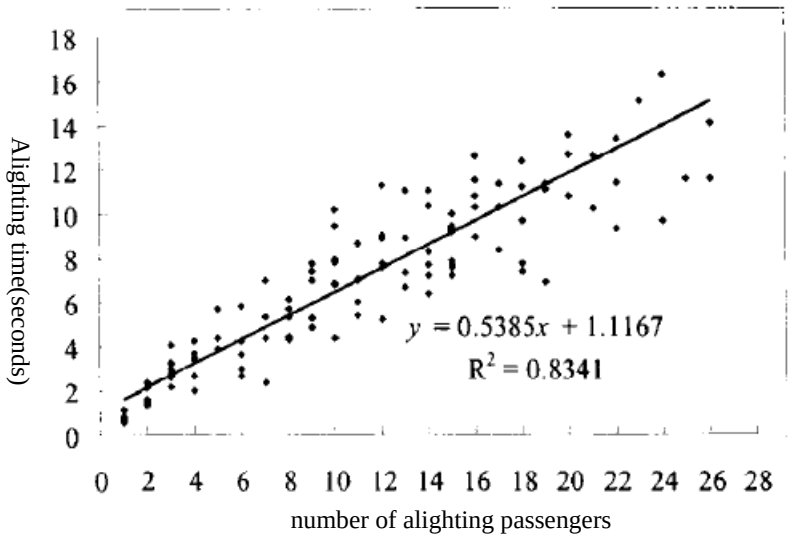


Figure 2: Fitted curve for alighting time and number of alighting passengers in Beijing Rail Transit. Cited from[3]

According to the formula, the alighting time of 45 passengers (half passengers in one car)would be:

$$t_0 = 0.5385 * 45 + 1.1167 \approx 26(seconds) \tag{1}$$

So the max time for a passenger to get off the train is 26 seconds.

3.2 Modeling moving on platform

In this subsection we calculate t_1, time a passenger walks from the exit of the train to the queue before the landing of stairway.

$$t_1 = (\frac{p}{2 \times s})/v_p = \frac{5 \times d}{s \times v_p} \tag{2}$$

where p is the length of the platform, s is the number of stairways and v_p is the moving speed of the passengers on the platform. From Fig.3 we could see the average walking distance for one passenger is $p/(2 * s)$, because the platform is separated into $(2 * s)$ parts by s stairways. So the key to this model is to calculate v_p, the average walking speed of passengers on the platform.

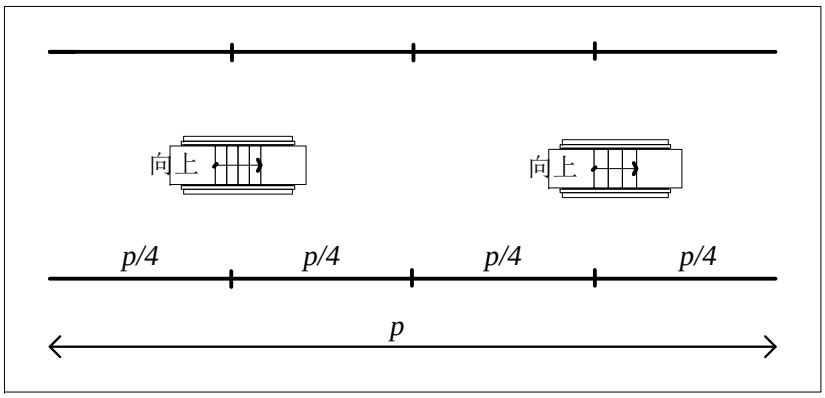


Figure 3: Four partitions of a platform with 2 stairways

Many research works have studied people's walking speeds under different circumstances. Qingmei Hu gave a summary[4] on these studies. The conclusions were:

1. If the crowd density $\leq 1.5\ persons/m^2$, people can walk nonblockingly.
2. People walk at an average speed of $1.34m/s$ in a nonblocking environment.
3. If the crowd density $\geq 4\ persons/m^2$, people stagnate and walk at an average speed of $0.25m/s$.
4. When the crowd density ranges from $1.5\ persons/m^2$ to $4\ persons/m^2$, the moving speed and crowd density have an approximate linear relation. Fig.4 is the summarized density-speed relationships for underground stations.

In this paper we use the result [3] observed from Beijing's rail transit directly, which is a linear relation between density and speed as shown in

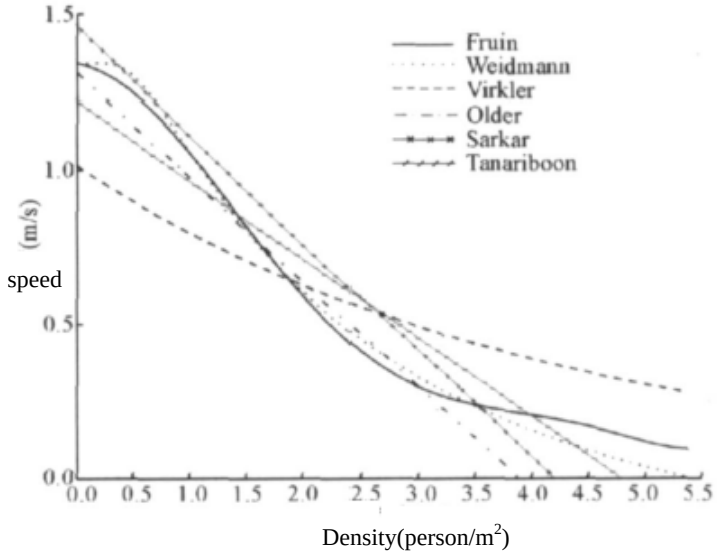


Figure 4: Density-speed relationship for underground stations.
Cited from [4]

Fig.5 :

$$v_p = -0.4778 * density + 1.759 \tag{3}$$

For example, if the effective width of the platform is $w = 6m$ (although the width of the platform may larger than $6m$, people tend to walk in the center area on the platform, which is narrower than $6m$), length of a car is $d = 20m$, 900 passengers held by a fully occupied train with 10 cars get off from the train. The length of the platform can be calculated as $20 * 10 = 200m$ approximately. The density on the platform would be $900/(200 * 6) = 0.75person/m^2$. The average walking speed of passenger is $(-0.4778 * 0.75 + 1.759) = 1.40m/s$. However, if two fully occupied trains arrive at the station at the same time, the crowd density would be $1800/(6 * 200) = 1.5person/m^2$. The average walking speed of passenger becomes $-0.4778 * 1.5 + 1.759 = 1.04m/s$.

According to the equation(2), when there are $s = 2$ stairways on the platform, the average moving time on the platform of a passenger who gets off one train is:

$$t_1 = (\frac{200}{2 * 2})/1.40 \approx 36(seconds)$$

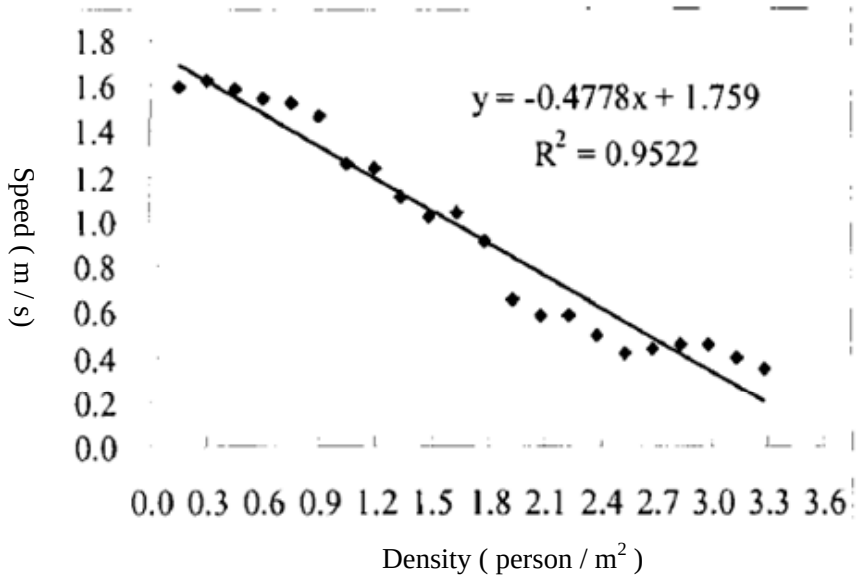


Figure 5: Fitted Linear Density-speed relationship observed at Beijing rail transit. Cited from [3]

The average moving time on the platform of a passenger on the occasion when two trains arrive together is:

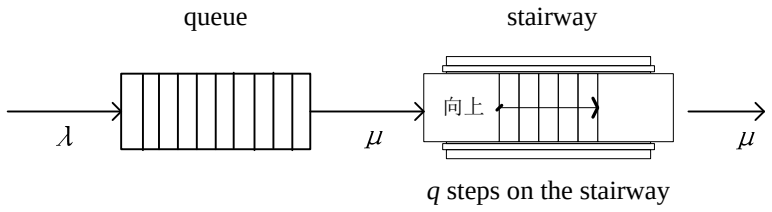
$$t_1 = (\frac{200}{2 * 2})/1.04 \approx 48(seconds)$$

3.3 Modeling queuing

When passengers get off the train, they walk through the stairways to get to the next level of the station. When there is a large number of passengers, some of them would gather before the stairways in a short time, and then wait in a queue to walk up the stairs.

We can regard the passenger flow attracted by a stairway as a queuing system. As shown in Fig.6, the queuing system includes three processes:

1. **Inputting**: Passengers walk to the nearest stairway for service. λ is the input rate of the queue(unit: persons/second), which is the number of passengers that arrive at the queue in unit time. For



λ: input rate of queue
μ: output rate of queue and stairway

Figure 6: d/d/s queuing model of the stairway in rail station

each stairway, the number of passengers it provides service with is $(90 * n * N)/s$. If we assume that the walking speed and walking distance of each passenger on the platform are the same, we can simply calculate the time interval between the first and the last passenger arriving at the queue as the maximum alighting time t_0. Then the input time of each stairway $t_{input} = t_0$, we can represent λ as:

$$\lambda = \frac{90 * n * N}{s * t_0}$$

2. **Queuing**: When a passenger arrives at the queue before the staircase, if the stair is vacant,the passenger can use the stair immediately. If the stair is occupied, the passenger will wait until the stair becomes vacant. The service rule of the stair is "first come first service".

3. **Outputting**: The queue before the staircase has same output rate as the staircase. We use μ (unit: persons/second)to denote the output rate of a stairway. According to the existing research work[5] and design specifications for underground in China[6], the maximum capacity of a stairway in the stations is around 3600 $p/m/hour$, that is 1 $person/m/s$, the width of the stairway is 1 meter. From our observation in Shanghai subway stations, a stair of 1- meter width can only accommodate 1 person. Since the given information does not describe the width of stairway, we can calculate that one stair of the stairway could contain $c = k/q$ people. k is the number of people a stairway could accommodate, q is the number of stairs in

each stairway. Hence:

$$\mu = c = \frac{k}{q}(person/s).$$

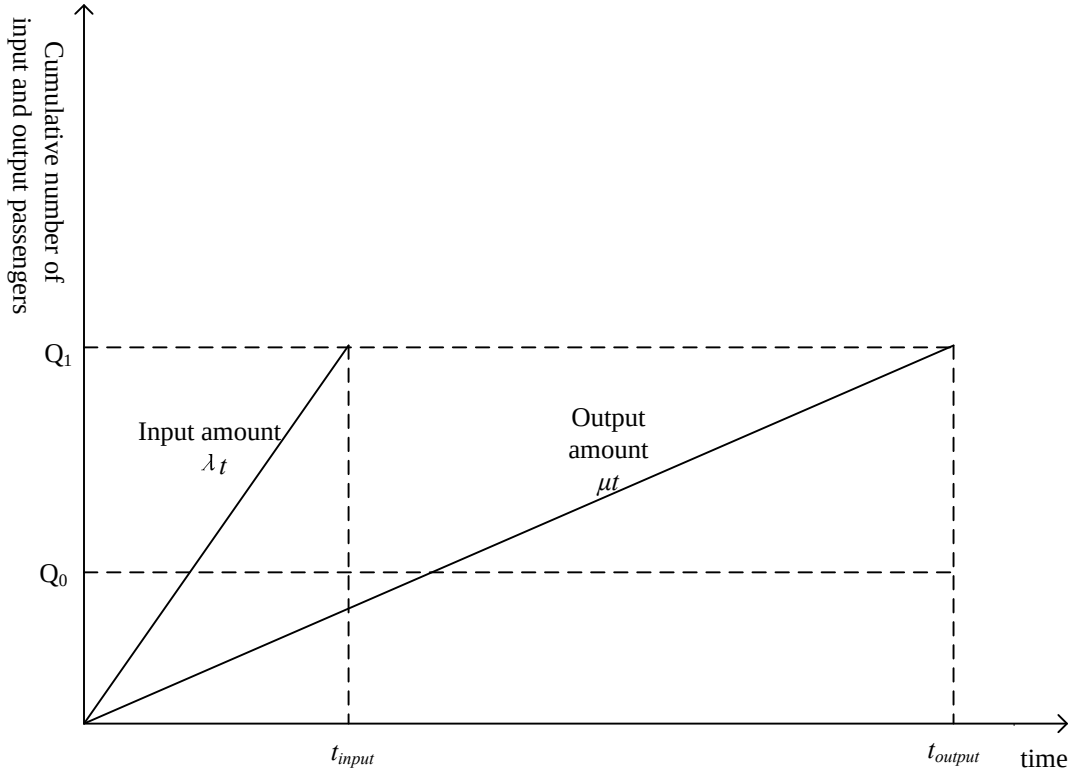


Figure 7: The input and output rate of the d/d/s queuing system. Cited from [7]

From the assumptions and analysis we know that the service system of each stairway is a fixed-length input, fixed-length output and single channel queuing system. The s stairways on the platform are a fixed-length input, fixed-length output and multiple channel queuing system, that is $d/d/s$ queuing system. Fig. 7 is an approximate representation of the $d/d/s$ queuing system. The horizontal axis represents time, and the vertical axis shows the cumulative number of input or output passengers. The input number and output number are shown with two curves: λt and μt.

The parameters in Fig.7 are:

t_{input} : the end time of the queue's input

t_{output} : the end time of the queue's output

Q_0 : the total output amount at the end time of the queue's input

Q_1 : the total input and output amount of the queue. For each stairway $Q_1 = 90nN/s$.

λ : the input rate of the queue

μ : the output rate of the queue

Then we can calculate some important result.

the maximum number of waiting passengers:

$$Q = Q_1 - Q_0 = \lambda \times t_{input} - \mu \times t_{input}$$

the end time of the queue's output:

$$t_{output} = Q_1/\mu$$

the maximum delay time in the queue:

$$t_2 = t_{output} - t_{input} = \frac{Q_1}{\mu} - t_0 = \frac{90 * n * N}{s * c} - t_0 \tag{4}$$

3.4 Walking on the stairway

When a passenger walk on the foot of the stairway, he passes through the stairway and then exit the station. Now we calculate t_3, the time used to pass through the stairway.

The research work[3] of Shouhua Cao collected many samples in Beijing rail transit. The work observed the average pace of the passengers on the up-going stairways is 1.85 steps/second. Hence,

$$t_3 = \frac{q}{1.85} \tag{5}$$

where q is the number of stairs(steps) on the stairway. If there are $q = 15$ stairs in each stairway, $t_3 \approx 8s$.

3.5 Calculating the total unloading time

Since t_3 is a constant in our assumption, and it is a short time if compared with other periods, in the later part of the paper, we mainly focus on optimizing unloading time, which is the process before a passenger walks on a staircase. The total unloading time of a train includes: alighting time, walking time on platform, and queuing time. According to equation(1) to (4), we can get:

$$\begin{aligned} t^{'} &= t_0 + t_1 + t_2 = t_0 + t_1 + t_{output} - t_0 \\ &= \frac{5 \times d}{s \times v_p} + \frac{90 \times n \times N}{s \times c} \end{aligned} \tag{6}$$

$$v_p = \begin{cases} 1.4\text{m/s}, & \text{if N=1, one train arrives} \\ 1.04\text{m/s}, & \text{if N=2, two trains arrive} \end{cases}$$

Table 1: The unloading time(s), $\text{t}^{'} = t_0 + t_1 + t_2$, $\text{t}=\text{t}^{'} + 8$(d= 20,n= 10)

number of trains	stairways	t_0	t_1	t_2	$t^{'}$	t
$N = 1$	$s = 2, c = 2$	26	36	199	261	269
$N = 2$	$s = 2, c = 2$	26	48	424	498	506
$N = 1$	$s = 2, c = 3$	26	36	124	186	194
$N = 2$	$s = 2, c = 3$	26	48	274	348	356
$N = 1$	$s = 3, c = 2$	26	24	124	174	182
$N = 2$	$s = 3, c = 2$	26	32	274	332	340

Table 1 shows the total unloading time in different situations. The worst case is: if there are only 2 stairways which contain 2 people on each stair, when 2 trains arrives at the same time, the total unloading time is $498s$.

The result also shows that this model does not distinguish some situations such as 2 stairways which contains 3 person on each stair and 3 stairways which contains 2 person on each stair. In the mathematical model, the output rate of the stairways service provided by these two circumstances is the same, so the estimated unloading time of these two situations are slightly different.

3.6 Strengths and weaknesses

Strengths:

- Our model gives a simple theoretical approach to calculate the exiting time. We separate the exiting process into different periods and calculate the average time or maximum time of each part. In this way, we get relatively accurate exiting time.

Weakness:

- We assume that all people walk at a fixed speed on the platform which does not meet the real expectation because people who come out first walk at a faster speed, whereas people who get off later walk at a slower speed since the platform is more and more crowded.
- In our theoretical model, the output rate of the stairways' service is (s×c), which could not distinguish the difference of two situations: more staircase with narrow stairs and less staircase with wide stairs.
- Our theoretical model assume the stairways are distributed uniformly on the platform and could not calculate different unloading time, particularly when the location of the stairways is changed.
- We assume passenger's walking speed on staircase is a constant, once a passenger stands on the staircase, he will exit the station after a fixed time. This assumption could be optimized further.

To solve these problems, we add a simulation approach based on the cellular automaton model.

4 Simulation Approach

To understand the effect of congestion level and number and location of the stairways, we simulate the whole unloading process with a cellular automaton model. We represent the plan of the platform in a grid of cells. Passengers can move from one cell to another cell according the

(a)Passengers begin to get off from the train

(b)Passengers gather before the staircases

(c) More and more passengers wait before the staircases

(d) Most of the passengers have walked on the staircases

Figure 8: The unloading process(N=2, n=5,d=20)

specified moving rules. Fig. 8 is the simulation of unloading process of 2 trains arriving at the same time.

In this simulation, the platform is partitioned into a grid structure, each with a size of $0.5m*0.5m$. The basis for this partition is that the minimum space requirement for a person is $0.28m^2$, which is the result[8] pointed out by National research council.

Cell is the basic unit of the grid structure. Each cell can contain only one passenger at one time step(1 second). The location of a passenger can be represented as the location of cell on the grid map. As shown in Fig.9, the adjacent 8 cells are the possible moving directions of a passenger. The model not only can simulate the movement of passengers, but also simulate passengers changing speed under different crowd densities. From Fig.9 we can see that in each time step, a passenger can move from 0 to 2 cells in different directions based on the crowd degrees.

In addition, the initial setup of the simulation includes: the length of each car is $20m$; each train has 10 cars; the length of platform is $200m$; the width of platform is 10m. Besides, the program can adjust the length and width of the staircase according to its capacity. The detailed algorithm used in the simulation is introduced in the next subsection.

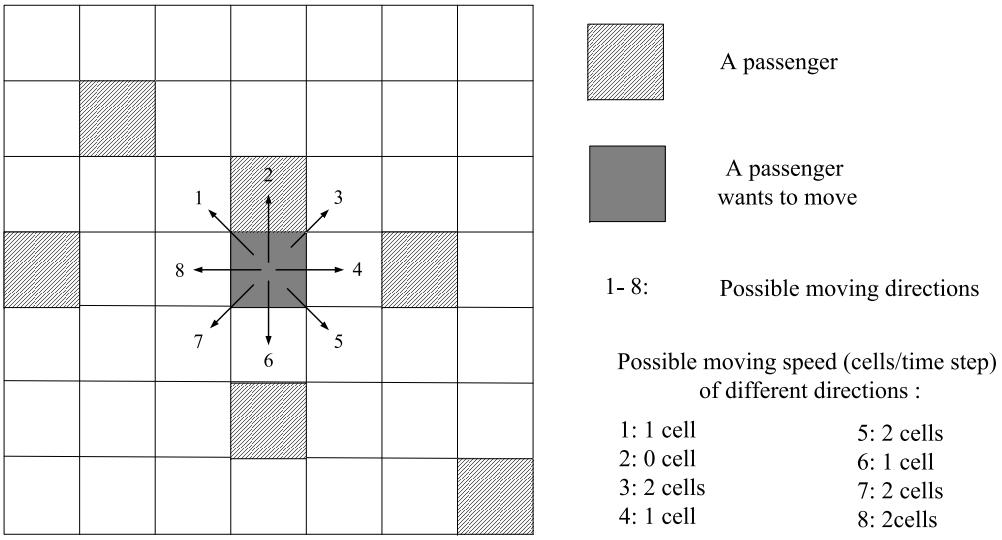


Figure 9: The moving direction of a passenger

4.1 Algorithm

The program was run with Javascript, an interpreted scripting language that can run only in an interpreter such as Internet Explorer. The process of the simulation program is shown in Fig.11. The complete code is given in Appendix B.

In the simulation program, each passenger first selects the nearest stairway as his target, then moves to the staircase according to the given moving rule. Fig.10 is an example of the moving rule. In this example, the staircase is the nearest one to the source node. Since each stair could contain 3 passengers, the source has three moving targets. In our model, we first calculate the shortest routes from the source to the three targets with A^* algorithm[9], then calculate the weights of each route according

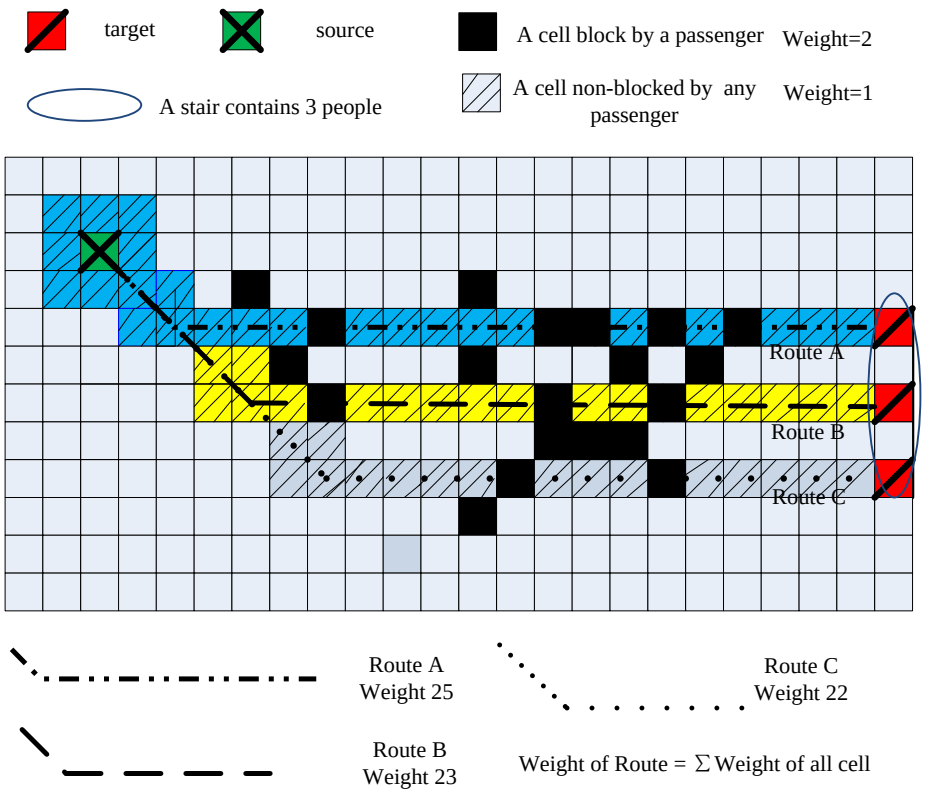


Figure 10: Use A^* algorithm and weight value to find the best route

to the equation:

$$\text{weight of a route} = \sum \text{weight of cells on the route}$$

$$\text{weight of a cell} = \begin{cases} 1, & \text{if no passenger holds the cell} \\ 2, & \text{if one passenger holds the cell} \end{cases}$$

From Fig 10. we can see the current best route of the source is route C, because it has smallest weight. This moving rule can assure that a source node selects his target by considering both distance and crowd level. This rule conforms to reality. Algorithm 1 is the description of the moving rule.

The simulation process is shown in Figure 11. In each timestep, the program moves the passengers on the cells of the grid map simultaneously and iterate the loop until all passengers are moved out of the staircases.

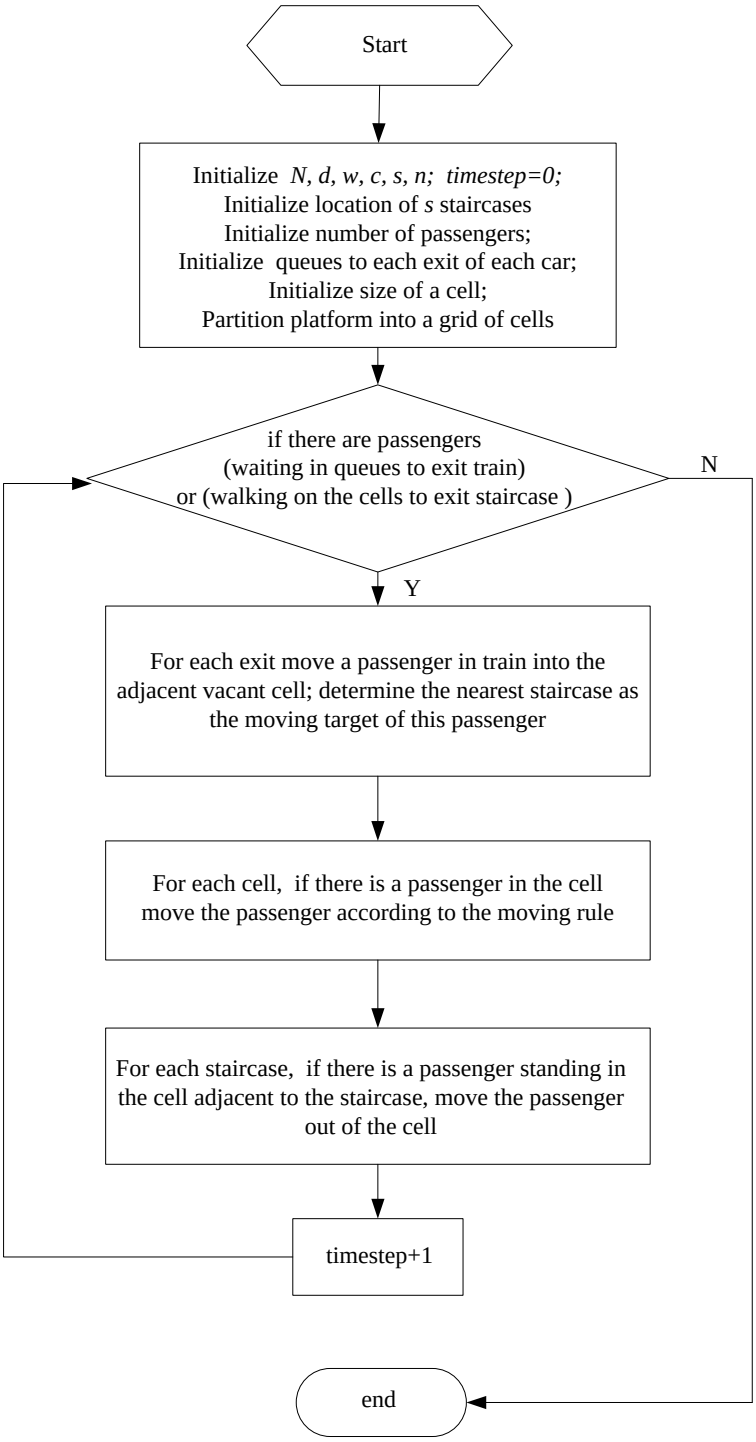


Figure 11: The flow chart of the simulation program

Algorithm 1 Moving rule.

Input:

source: location of the passenger.

c: the capacity of one stair.

$target[1..c]$: c cells on the grid map, represent the parallel c exits of one staircase.

$gridmap[\ ]$: All cells on the grid map, record the locations of all passengers on the platform.

Output:

best_target: select one from $target[1..c]$ as the moving target;

best_route: a route to *best_target*;

1: calculate shortest routes from source to $target[1..c]$ with A^* algorithm, get $route[1..c]$;

2: calculate the weight value of $route[1..c]$, for each cell on the route, if no passenger hold the cell, then weight+=1, else weight+=2;

3: select the route with smallest weight as *best_route*;

4: select the target of *best_route* as *best_target*;

5: if the first two cells in *best_route* are vacant, move *source* to the second cell in *best_route* .

6: if the first cell in *best_route* is vacant but second cell is occupied, move *source* to the first cell in *best_route* .

7: if the first two cells in *best_route* are all occupied, no movement for *source*.

8: **return** *best_route* and *best_target*;

4.2 Simulation result

Table 2: Compare the theoretical and simulated unloading time(s)

Number of train	Stairways	Simulation model	Mathematic model
		t′	t′
$N = 1$	$s = 2, c = 2$	251	261
$N = 2$	$s = 2, c = 2$	498	498
$N = 1$	$s = 2, c = 3$	203	186
$N = 2$	$s = 2, c = 3$	373	348
$N = 1$	$s = 3, c = 2$	170	174
$N = 2$	$s = 3, c = 2$	322	332

All simulation results are achieved through running the program 10 times

for each group of parameters to calculate the average values. We selected 6 different situations to prove the reliability of our model. Table 2 shows the comparison of the simulated result and the result of mathematical analysis. The two results are similar, so they prove each other very well. After careful observation we find that the simulation result is more close to reality. It successfully distinguished the difference between two situations: more staircases with narrow stairs and less staircases with wide stairs. In the simulation result, the unloading time at the case (s=2,c=3) is larger than the one at (s=3,c=2). This result is reasonable because more staircases can alleviate the crowd level to help passengers move faster.

4.3 Sensitive analysis

To determine how the simulation model responds to the fluctuations in the original setup, we run our model using different grid size (equivalent to change people's walking speed.) Our model proves to be not very sensitive to small changes. After increasing the width of a cell by 10%, maximum unloading time decreases less than 3%; after a similar decrease, maximum unloading time increases less than 3%. As we can see, the unloading time does not change much as the walking speed fluctuates. Our model is proven to be stable.

Table 3: The simulated unloading time(s)
with different grid sizes(m*m)

trains	staircases	0.5*0.5	0.45*0.45	0.55*0.55
$N = 1$	$s = 2, c = 2$	251	252	249
$N = 2$	$s = 2, c = 2$	498	499	494
$N = 1$	$s = 2, c = 3$	203	209	199
$N = 2$	$s = 2, c = 3$	373	381	364
$N = 1$	$s = 3, c = 2$	170	177	164
$N = 2$	$s = 3, c = 2$	322	330	318

4.4 Strengths and weaknesses

Strengths:

- Our model visualizes the whole process.

- The moving rule considers the effect of congestion level on moving speed and moving direction, which is more realistic.
- The layout of the grid structure not only considers the space requirements of passengers, but also considers the space requirement of staircases, which is more realistic.
- Our model is flexible that it can be easily applied to more complex transfer stations' design only after adapting some parameters.

Weaknesses:

- Our model uses A* algorithm to calculate the shortest route to the target at each time step, so the time complexity of the algorithm is high. Optimization of the algorithm will be our future work.
- In this model we still assume passenger's walking time on staircase is a constant, we only simulate the process before passengers stand on the staircases.

5 Redesign the Stairways

5.1 More stairways

Table 4: The unloading time(s), c=2, s= 2 to 5

Number of train	Stairways	theoretical result	simulation result	
		Max=$t_0 + t_1 + t_2$	Max	Avg.
$N = 1$	$s = 2, c = 2$	261	251	154
$N = 2$	$s = 2, c = 2$	498	498	320
$N = 1$	$s = 3, c = 2$	174	170	99
$N = 2$	$s = 3, c = 2$	332	322	202
$N = 1$	$s = 4, c = 2$	131	136	72
$N = 2$	$s = 4, c = 2$	249	274	154
$N = 1$	$s = 5, c = 2$	104	101	55
$N = 2$	$s = 5, c = 2$	199	241	112

As we assumed, the staircases are placed uniformly along the longitudinal direction of the platform. When staircases are added, the platform is divided into more parts, which results in the change of passenger flow.

With the cellular automaton model, we assumed the situation where N trains arrive at the same time. Each train has 10 cars with 900 people. In this occasion, we varied the number of staircases (s = 2,3,4,5) to find out how the unloading time varies with s. We compare the theoretical result with the simulated one, which is shown in Table 4. Fig.12 shows how the unloading time varies with the number of staircases when two trains arrive.

Overall, more stairways is an effective solution to reduce the total unloading time of trains. From the result we can see the unloading time reduces greatly when we add a new stairway (s=3). The total unloading time for one train decreases by 81 seconds. Nevertheless, the unloading time does not decrease linearly with the increase of the number of staircases. The subsequent additional staircase does not seem as effective as the former one, as the decreasing time is 34 sec and 35 sec respectively. The pattern is more obvious in such case as two fully loaded trains reach the station at the same time.

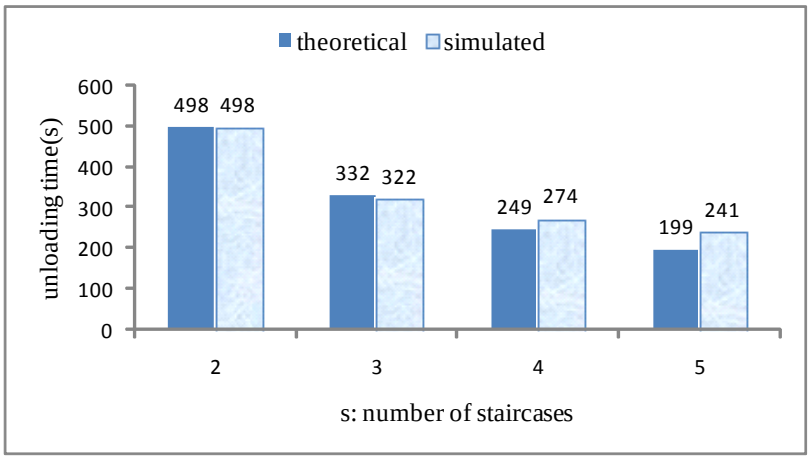


Figure 12: The unloading time varies with the number of staircases. (N=2, c=2)

5.2 Widening staircase

we analyzed how the unloading time varies with the alteration of c(the number of passengers a single stair can accommodate.) We set a similar occasion: N trains arrive at the same time and each train has 10 cars with 900 people. We compare the theoretical result with simulated one in Table 5. The results show that widening staircase is still an effective way to reduce the unloading time. But when we compare the results in Table 4 and Table 5, we find that adding new staircases is more effective than widening staircases.

Table 5: The simulated unloading time(s), s=2, c= 2 to 5

Number of train	Stairways	theoretical result	simulation result	
		Max=$t_0 + t_1 + t_2$	Max	Avg.
$N = 1$	$s = 2, c = 2$	261	251	154
$N = 2$	$s = 2, c = 2$	498	498	320
$N = 1$	$s = 2, c = 3$	186	203	118
$N = 2$	$s = 2, c = 3$	348	373	241
$N = 1$	$s = 2, c = 4$	149	179	99
$N = 2$	$s = 2, c = 4$	273	324	192
$N = 1$	$s = 2, c = 5$	126	168	86
$N = 2$	$s = 2, c = 5$	228	296	160

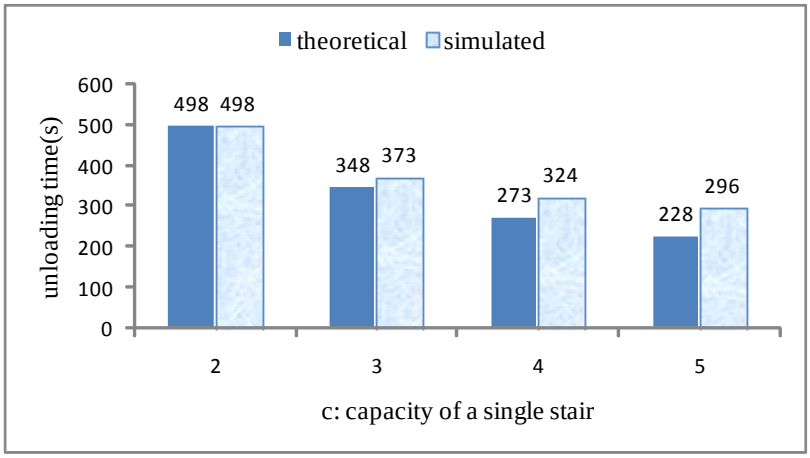
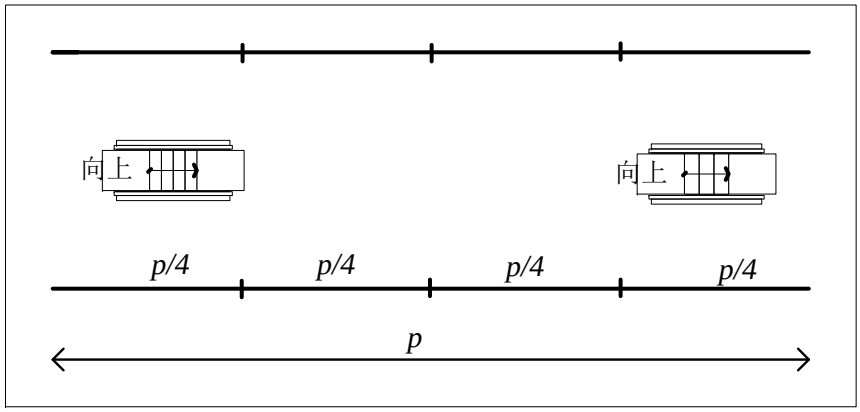


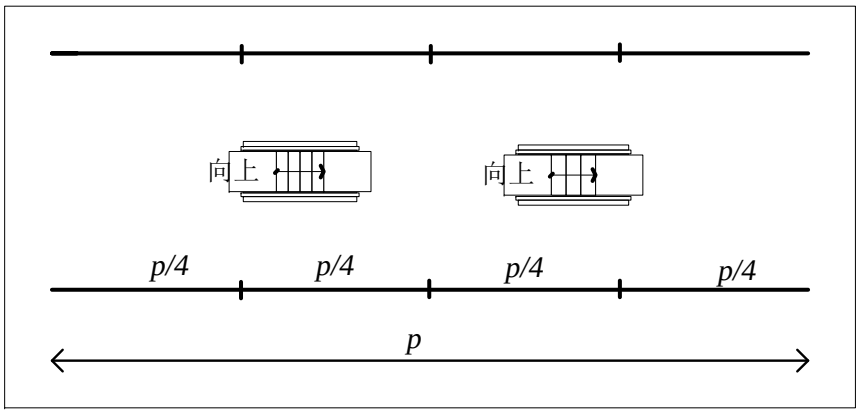
Figure 13: The unloading time varies with the capacity of staircases. (N=2, s=2)

5.3 Changing the location of stairways

To minimize the evacuation time, we search the best location of the stairways in the simulation program. The search process is shown in Fig.14. Initially, we placed the stairways averagely distributed on the platform as shown in Fig.2, then we moved the stairways outward and inward in a changing distance to search for the best location of the stairways which can yield the shortest unloading time.



(a)Moving outwards the two staircases



(b)Moving inwards the two staircases

Figure 14: Search the best location of the stairways

In the example we simulate the situation of $N(=1 \text{ or } 2)$ trains arriving at the same time. There are 10 cars in each trains and 90 people will get off from each car. The length of the platform is 200 meters and there are 2

stairways on the platform. Initially, the platform is separated by the two stairway into an average of 4 partitions. Then we move the locations of the stairway inward and outward x (= 5, 10, 15, 20) meters respectively. The processes are run 10 times to calculate the average time. From the result in Table 6 we find that the effect of changing locations of staircases is not significant. Compared with the original symmetrical distribution of the stairways, the unloading time of new lactations only changes a little which can be ignored.

We proved this result through investigating two platforms in People's Square underground station in Shanghai. The platform of line 2 is divided by four stairways into five parts averagely, while the platform of line 8 is separated unequally, with two stairways in center and two stairways at start and end of the platform. Both distributions do not show obvious imbalanced passenger flows.

Table 6: The unloading time(s)
with different stairways' locations

Locations	N=1	N=2
uniform	251	498
$\rightarrow\leftarrow 5m$	253	506
$\rightarrow\leftarrow 10m$	251	488
$\rightarrow\leftarrow 15m$	240	494
$\rightarrow\leftarrow 20m$	243	500
$\leftarrow\rightarrow 5m$	254	501
$\leftarrow\rightarrow 10m$	248	502
$\leftarrow\rightarrow 15m$	250	496
$\leftarrow\rightarrow 20m$	243	487

6 Conclusion

We model the unloading process in a central station with two approaches: theoretical mathematics and computational simulation. By dividing the process into four periods, we can compute exact alighting time, time on the platform, queuing time and time on the staircase. However, this method can not represent some factors, such as staircases' different locations. Therefore, we also use a computational model to simulate the exiting process. By comparing the results of theoretical and simulation

approaches, we are able to see that the two results are approximately the same, while the result of the simulation approach may be better because it takes many real factors into consideration. Also, we are able to vindicate the accuracy and reliability of our simulation model, so we can use it in further analysis when taking more complex factors into consideration.

Our computational simulation suggests that the number of staircases contributes most to the decreasing of excavating time. While the width of each staircase is an important relevant variable, it does not contribute as much as the the number of staircases. On the contrary, the location of staircases does not have a great impact on the unloading time.

Therefore, we give following suggestions to help relieve the traffic stress on public transit stations. First, if two trains arrive at the same time, 1800 passengers will exit the station through 2 staircases, the operator should arrange for the next train to arrive in no less than 500 seconds since unloading time for 2 trains is 498 seconds. Secondly, when the layouts of platform at present can not bear the flow of people, the transportation station can simply add a staircase anywhere on the platform because the location is not important and because widening the existing staircases is neither efficient nor cost-effective.

In the future, we can apply our simulation model to more complex transfer stations with criss-cross platforms and different train lines. Also, it would be interesting to consider the distribution of walking speed rather than set it to a fix number. Our model can be applied to different situations and help rail transit designers to find the best plan for platforms and staircases.

7 A letter to Director of Transportation

Dear Director of Transportation:

We built a mathematical model to examine the exiting process of a large number of passengers. In the model we adapted some factors to reflect the practical design requirements. These factors include the location and number of the staircases and the capacity of each staircase. In addition we used a computational approach to simulate the exiting process. Both approaches could calculate the exiting time under different kinds of situations. We were glad to find out that the results of the mathematics model and the simulation model are consistent with each other. According to the model and simulation we created for the stations' original condition (2 staircases at a platform, 1 or 2 trains unloading at a time, every stair holds 2 commuters at a time), the total time of unloading the passengers is around 251 seconds for one train while approximately 498 seconds for two trains.

The result revealed a problem: it took nearly 10 minutes, which is a relative long time to evacuate passengers from two trains. Besides, if the next train arrives at the station in 10 minutes before all the passengers of the last train have exited the station, more and more passengers would gather on the platform. This may lead to increased risk of accident. To solve the problem, our model provided the solutions: to increase the number of staircases or to expand the capacity of staircases. In addition we concluded that the locations of staircases are not the key factors to affect the exiting time.

We strongly recommend you to use our model to help design your transportation system. The benefits of our model are that it can visualize the whole unloading process and show the congestion level in a direct way and that it can be applied to more complex transfer stations with criss-cross platforms and different train lines after slightly adjusting some variables. Our model is also strong because all the assumptions we make are consistent with the real situation and are not arbitrary. The passengers' moving speeds and moving rule were based on observation data given in some former research. We feel confident that our models are fairly accurate. Therefore, we respectfully recommend that the model be used to help design the platforms for complex transform stations.

Sincerely yours, Himcm 2014 team 4671

References

[1] www.virgintrains.co.uk/assets/pdf/global/seating-plan.pdf

[2] William H.K. Lam, Chung-Yu Cheung, C.F. Lam. A study of crowding effects at the Hong Kong light rail transit stations. Transportation Research A, 33: 401-415, 1999.

[3] Shouhua Cao. Analysis and Modeling on Passengers Traffic Characteristics for Urban Rail Transit. Ph.D Thesis. Beijin Jiaotong Univ. 2009.

[4] Qingmei Hu, Weining Fang, Guangyan Li, Yuquan Jia. Review on Pedestrian Behavior Characteristics and Crowding Mechanism in Public Buildings. China Safety Science Journal. Vol.18, No.8, Aug. 2008.

[5] J.J.Fruin. Pedestrian Planning and Design. Metropolitan Association of Urban Designers and Environmental Planners Inc, 1971.

[6] GB50157-2003, Design Specifications for Underground. China Planning Press.Beijin, 2003.

[7] Xueping Rao, An analysis of passenger delays in stairs and escalators of urban rail transit station. Traffic and Transportation. 2005,7.

[8] Transportation research board. Highway capacity manal 2000. Washington DC, National research council, 2000.

[9] Thomas H. Cormen, Charles E. Leiserson,Ronald L. Rivest, Clifford Stein, Introduction to Algorithms. MIT press. 2005.

8 Appendix

8.1 Appendix A. The unloading time with varied parameters

Table 7: The simulated unloading time with varied parameters(s), N=1

	$s=2$	$s=3$	$s=4$	$s=5$
$c=2$	251	170	136	101
$c=3$	203	142	112	94
$c=4$	179	133	91	83
$c=5$	168	126	93	74

Table 8: The simulated unloading time with varied parameters(s), N=2

	$s=2$	$s=3$	$s=4$	$s=5$
$c=2$	498	322	274	241
$c=3$	373	317	236	204
$c=4$	324	225	168	150
$c=5$	296	154	204	151

8.2 Appendix B. The simulation program

```
var astar = require('./astar_1.js');
var scale_size = 6;
var length_car = 20;
var number_car = 10;
var number_passenger_per_car = 90;
var width_room = 10;
var length_staircase = 5;
var width_staircase = 3;
var length_passenger = 0.5;
var width_passenger = 0.5;
var passenger_per_staircase = 2;
var number_staircase = 2;
var left = 1; //the direction of staircase
var right = 2;
var staircase_direction = right;

var empty_entry = 0x000001;
var passenger = 0x000004;
var blocked = 0 xffffff ;
var car_body = 0 xffffff ;
var car_door = 0 xffffff ;
var staircase_entrance = 0x000008;
var staircase_exit = 0 xffffff ;
var stair = 0 xffffff ;

var number_train = 2;

var num_col = Math.floor(length_car * number_car / length_passenger);
var num_row = Math.floor(width_room / width_passenger + 2);

var total_number_passenger = number_car * number_passenger_per_car *
    number_train;
var passenger_in_car = total_number_passenger;
var passenger_out_station = 0;

var staircase_entrance_list = new Array();
var staircase_queue = new Array();
var staircase_queue_max_length = 10;
var staircase_offset = [8, -8];

var car_list = new Array(number_train);
// car_list [1] = new Array(number_car);
for (var i = 0; i < car_list.length; ++i) {
        car_list[i] = new Array(number_car);
        for (var j = 0; j < car_list[i].length; ++j) {
                car_list[i][j] = {'passenger_in_car' :
```

```
                    number_passenger_per_car};
        }
}

var station = new Array(num_row);
for (var i = 0; i < num_row; ++i) {
        station[i] = new Array(num_col);
        for (var j = 0; j < num_col; ++j) {
                if (i == 0 || i == num_row - 1) {
                        station[i][j] = car_body;
                } else {
                        station[i][j] = empty_entry;
                }
        }
}
var flag_matrix = new Array(num_row);
for (var i = 0; i < num_row; ++i) {
        flag_matrix[i] = new Array(num_col);
        for (var j = 0; j < num_col; ++j) {
                flag_matrix[i][j] = true;
        }
}

function buildCar() {
        for (var index_car = 0; index_car < number_car; ++index_car) {
                station [0][ Math.floor(index_car * (length_car /
                    length_passenger))] = empty_entry;
                station[num_row - 1][Math.floor(index_car * (length_car /
                    length_passenger))] = empty_entry;
                station [0][ Math.floor((index_car + 1) * (length_car /
                    length_passenger)) - 1] = empty_entry;
                station[num_row - 1][Math.floor((index_car + 1) * (length_car
                    / length_passenger)) - 1] = empty_entry;
                car_list [0][ index_car ][ 'row'] = 0;
                car_list [0][ index_car ][ 'left_door '] = Math.floor(index_car * (
                    length_car / length_passenger));
                car_list [0][ index_car ][ 'right_door '] = Math.floor((index_car
                    + 1) * (length_car / length_passenger)) - 1;
                if (number_train == 2) {
                        car_list [1][ index_car ][ 'row'] = num_row - 1;
                        car_list [1][ index_car ][ 'left_door '] = Math.floor(
                            index_car * (length_car / length_passenger));
                        car_list [1][ index_car ][ 'right_door '] = Math.floor((
                            index_car + 1) * (length_car / length_passenger))
                            - 1;
                }
        }
}
```

```
function buildStaircase() {
        var start_staircase_row = Math.floor((num_row - width_staircase /
            width_passenger) / 2);
        var end_staircase_row = Math.floor(num_row - (num_row -
            width_staircase / width_passenger) / 2);
        var interval = Math.floor((num_col / number_staircase -
            length_staircase / length_passenger) / 2);

        //setup staircase blocks
        for (var index_staircase = 0; index_staircase < number_staircase; ++
            index_staircase) {
            //var start_col = interval * (index_staircase + 1);
            //var end_col = start_col + Math.floor(length_staircase /
                length_passenger);
            var start_col =
            Math.floor(index_staircase / number_staircase * num_col +
                interval) + staircase_offset[index_staircase];
            var end_col =
            Math.floor((index_staircase + 1) / number_staircase * num_col
                - interval) + staircase_offset[index_staircase];
            for (var row = start_staircase_row; row <= end_staircase_row;
                ++row) {
                for (var col = start_col; col <= end_col; ++col) {
                    if (row == Math.floor((start_staircase_row +
                        end_staircase_row) / 2) ||
                            row == Math.floor((
                                start_staircase_row +
                                end_staircase_row) / 2) + 1) {
                        station[row][col] = stair;
                    } else {
                        station[row][col] = blocked;
                    }

                    if (staircase_direction == left) {
                        if (col == start_col && row <
                            start_staircase_row +
                            passenger_per_staircase) {
                            station[row][col] =
                                staircase_entrance;
                            staircase_entrance_list.push
                                ({
                                    'row': row,
                                    'col': col
                            });
                            var tmp_queue = new Array();
                            for (var i = 0; i <
                                staircase_queue_max_length
                                ; ++i) {
                                tmp_queue.push(
```

```
                                                empty_entry);
                                        }
                                        staircase_queue.push(
                                            tmp_queue);
                                    } else if (col == end_col && row <
                                        start_staircase_row +
                                        passenger_per_staircase) {
                                        station[row][col] =
                                            staircase_exit;
                                    }
                                } else {
                                    if (col == start_col && row <
                                        start_staircase_row +
                                        passenger_per_staircase) {
                                        station[row][col] =
                                            staircase_exit;
                                    } else if (col == end_col && row <
                                        start_staircase_row +
                                        passenger_per_staircase) {
                                        station[row][col] =
                                            staircase_entrance;
                                        staircase_entrance_list.push
                                            ({
                                                'row': row,
                                                'col': col
                                        });
                                        var tmp_queue = new Array();
                                        for (var i = 0; i <
                                            staircase_queue_max_length
                                            ; ++i) {
                                            tmp_queue.push(
                                                empty_entry);
                                        }
                                        staircase_queue.push(
                                            tmp_queue);
                                    }
                                }
                            }
                        }
                        if (staircase_direction == left) {
                            staircase_direction = right;
                        } else {
                            staircase_direction = left;
                        }
                    }
                }
```

```
function distance(row1, col1, row2, col2) {
        return Math.max(Math.abs(row1 - row2), Math.abs(col1 - col2));
}

function findPathToNearestStaircase(row, col) {
        var staircase_list = findNearestStaircaseList(row, col);
        var path_list = new Array();
        var graph = new astar.Graph(station, { diagonal: true });
        var start = graph.grid[row][col];

        // console.log( staircase_list );

        for (var index = 0; index < staircase_list.length; ++index) {
                var staircase_position = staircase_list[index];
                var end = graph.grid[staircase_list[index]['row']][
                    staircase_list[index]['col']];
                var result = astar.astar.search(graph, start, end);
                var distance_left = 0;
                for (var i = 0; i < result.length; ++i) {
                        distance_left += station[result[i]['x']][result[i]['y'
                            ]];
                }

                var local_path = {
                        'path': result,
                        'distance_left': distance_left
                        // - station[staircase_list[index]['row']][
                            staircase_list[index]['col']]
                }
                path_list.push(local_path);
        }

        return path_list;
}

function distanceLeft(row, col) {
        // var staircase_list = findNearestStaircaseList(row, col);
        var path_list = findPathToNearestStaircase(row, col);
        var min_left = num_row + num_col;

        for (var i = 0; i < path_list.length; ++i) {
                min_left = Math.min(min_left, path_list[i]['distance_left']);
        }

        return min_left;
}

function findNearestStaircaseList(row, col) {
        var nearest_staircase = null;
```

```
        var min_distance = -1;
        var distance_list = new Array();
        for (var index_staircase = 0; index_staircase <
            staircase_entrance_list.length; ++index_staircase) {
            distance_list.push({
                'row': staircase_entrance_list[index_staircase]['row'
                    ],
                'col': staircase_entrance_list[index_staircase]['col'
                    ],
                'distance': distance(row, col, staircase_entrance_list
                    [index_staircase]['row'], staircase_entrance_list
                    [index_staircase]['col'])
            })
        }
        // console.log(distance_list);
        distance_list.sort(function (m, n) {
            return m['distance'] > n['distance'] ? 1 : (m['distance'] < n[
                'distance'] ? -1 : 0);
        })

        return distance_list.slice(0, passenger_per_staircase);
}

function contains(path, position) {
        for (var i = 0; i < path.length; ++i) {
            if (path[i]['row'] == position['row'] &&
                path[i]['col'] == position['col']) {
                return true;
            }
        }
        return false;
}

var time_counter = 0;
var passenger_in_station_list = new Array();
var passenger_time_cost = new Array();

function get_passenger_time_cost(row, col) {
        for (var index = 0; index < passenger_in_station_list.length; ++
            index) {
            if (passenger_in_station_list[index]['row'] === row &&
                passenger_in_station_list[index]['col'] === col) {
                return passenger_in_station_list[index]['time'];
            }
        }
        return 0;
}

function move_passenger_into_station(row, col) {
```

```
        var distance_left = distanceLeft(row, col);
         passenger_in_station_list.push({
                'row': row,
                'col': col,
                'time': 0,
                'min_distance': distance_left
        });
}

function move_passenger_out_of_station(row, col) {
        for (var index = 0; index < passenger_in_station_list.length; ++
            index) {
                if (passenger_in_station_list[index]['row'] === row &&
                        passenger_in_station_list[index]['col'] === col) {
                        passenger_time_cost.push(passenger_in_station_list[
                            index]['time']);
                        passenger_in_station_list.splice(index, 1);
                        station[row][col] = staircase_entrance;
                        flag_matrix[row][col] = false;
                        break;
                }
        }
}

function move_passenger_to(row_from, col_from, row_to, col_to) {
        for (var index = 0; index < passenger_in_station_list.length; ++
            index) {
                if (passenger_in_station_list[index]['row'] === row_from
                    &&
                        passenger_in_station_list[index]['col'] === col_from
                            &&
                        (station[row_to][col_to] === empty_entry ||
                                station[row_to][col_to] ===
                                    staircase_entrance)) {
                        var distance_left = distanceLeft(row_to, col_to);
                        passenger_in_station_list[index]['min_distance'] =
                            distance_left;
                        passenger_in_station_list[index]['row'] = row_to;
                        passenger_in_station_list[index]['col'] = col_to;
                        flag_matrix[row_to][col_to] = false;
                        station[row_to][col_to] = passenger;
                        station[row_from][col_from] = empty_entry;
                        break;
                }
        }
}

buildCar();
buildStaircase();
```

```
var timer = 0;

function move_passenger() {
        for (var index = 0; index < passenger_in_station_list.length; ++
            index) {
            ++passenger_in_station_list[index]['time'];
        }

        for (var row = 0; row < num_row; ++row) {
            for (var col = 0; col < num_col; ++col) {
                flag_matrix[row][col] = true;
            }
        }

        for (var index_passenger = 0; index_passenger <
             passenger_in_station_list.length; ++index_passenger) {
            var current_passenger = passenger_in_station_list[
                index_passenger];
            // if (current_passenger['min_distance'] === 0) {
            //     // continue;
            //     console.log(current_passenger);
            // }
            // console.log(current_passenger);
            var path_list = findPathToNearestStaircase(current_passenger[
                'row'], current_passenger['col']);
            var next_position = undefined;
            var min_left = num_row + num_col;

            var choosen_path;
            for (var i = 0; i < path_list.length; ++i) {
                var path = path_list[i];
                if (path['distance_left'] < min_left) {
                    min_left = path['distance_left'];
                    choosen_path = path['path'];
                }
                // console.log(path['distance_left']);
            }

            var empty_entry_counter = 0;
            for (var j = 0; j < choosen_path.length &&
                empty_entry_counter < 2; ++j) {
                var position = choosen_path[j];
                if ((station[position['x']][position['y']] ===
                    empty_entry ||
                        station[position['x']][position['y']] ===
                            staircase_entrance) &&
                        flag_matrix[position['x']][position['y']]) {
                        ++empty_entry_counter;
```

```
                } else if (station[position['x']][position['y']] ===
                    passenger) {
                    break;
                }
                next_position = position;
            }

            if (next_position !== undefined && flag_matrix[next_position[
                'x']][next_position['y']]) {
                move_passenger_to(current_passenger['row'],
                    current_passenger['col'], next_position['x'],
                    next_position['y']);
            }
        }

        passenger_in_station_list.sort(function (m, n) {
            return m['min_distance'] > n['min_distance'] ? 1 : (m['
                min_distance'] < n['min_distance'] ? -1 : 0);
        })

        for (var index_staircase = 0; index_staircase <
            staircase_entrance_list.length; ++index_staircase) {
            var staircase_tmp = staircase_entrance_list[index_staircase];
            if (station[staircase_tmp['row']][staircase_tmp['col']] ===
                passenger) {
                if (time_counter % 2 == 0) {
                    passenger_time_cost.push(
                        get_passenger_time_cost(staircase_tmp['
                        row'], staircase_tmp['col']));
                }
                move_passenger_out_of_station(staircase_tmp['row'],
                    staircase_tmp['col']);
                ++passenger_out_station;
            }
        }

        //get passenger out of car
        for (var i = 0; i < car_list.length; ++i) {
            for (var j = 0; j < car_list[i].length; ++j) {
                var row = car_list[i][j]['row'];
                var col = car_list[i][j]['left_door'];
                if (car_list[i][j]['passenger_in_car'] > 0) {
                    if (station[row][col] == empty_entry) {
                        station[row][col] = passenger;
                        // setBitmap(row, col, passenger);
                        car_list[i][j]['passenger_in_car']
                            -= 1;
                        move_passenger_into_station(row, col);
                    }
```

```
                        }
                        col = car_list[i][j]['right_door'];
                        if (car_list[i][j]['passenger_in_car'] > 0) {
                                if (station[row][col] == empty_entry) {
                                        station[row][col] = passenger;
                                        // setBitmap(row, col, passenger);
                                        car_list[i][j]['passenger_in_car']
                                            -= 1;
                                        move_passenger_into_station(row, col);
                                }
                        }
                }
        }

        ++time_counter;
        ++timer;
}


while (passenger_out_station !== total_number_passenger) {
        move_passenger();
}
passenger_time_cost.sort(function (m, n) {
        return m > n ? 1 : (m < n ? -1 : 0);
})

var total_time_cost = 0;
for (var i = 0; i < passenger_time_cost.length; ++i) {
        total_time_cost += passenger_time_cost[i];
}

console.log(total_time_cost / total_number_passenger);
```