For office use only	Team Control Number	For office use only
T1 _____	**1923210**	F1 _____
T2 _____		F2 _____
T3 _____	Problem Chosen	F3 _____
T4 _____	**D**	F4 _____

2019
MCM/ICM
Summary Sheet

Time to Leave the Louvre

Summary

With the development of the times and the progress of science and technology, evacuation in emergencies of large buildings like the Louvre is becoming more and more feasible. In our paper, after the establishment of the model, the optimization of the model, the calculation of specific data, and the additional consideration of different factors, we give some suggestions on the emergency management of the Louvre. Finally, we give the best evacuation plan for the Louvre.

Firstly, our group abstracted the connected exhibition areas into weighted undirected graphs. Using Yen's algorithm, we established a model to find the shortest path from every point in the museum to each exit. Then, by establishing the grid model on undirected graph and applying Little's Law, a time calculation model for each path is developed. Through the above two models, we can get the exact evacuation time for the people in each grid on a specific route.

Then, we optimize the original model: in order to solve the problem of inaccurate calculation of evacuation time caused by inadequate mesh fineness, we further refine the grid; in the absence of considering the delayed response time of visitors in emergencies, we adopt GSPN model to estimate the additional response time of visitors, which makes the final result closer to the actual situation.

Afterwards, we select the best evacuation schemes by setting a threshold at the intersection point (when the number of intersection point is large, the evacuation time will be long) among various feasible solutions. Taking their shortest fully evacuation time as a basis for comparison, we get a solution with the shortest evacuation time, which is 349.8 s.

After the establishment of our model and data calculation, we conducted sensitivity analyses of the model. We concluded that the more people in a unit grid, the longer the fully evacuation time will be; the finer the meshing, the more accurate the fully evacuation time will be.

Finally, we put forward some suggestions for the emergency management of the Louvre to help implement the evacuation plan.

Contents

1 Introduction

1.1 Background

Evacuation is a common strategy in emergency management. In many hazardous events, such as terror attacks, fire disasters and riots, the best option is to relocate threatened populations to safer areas. There is existing work related to stimulating occupants evacuation from a public building. The paper of Zheng Xiaoping[8] discussed the advantages and disadvantages of seven models—cellular automata model, lattice gas model, social force model, fluid-dynamic model, agent-based model, game theoretic model and approaches based on experiments with animals—to study crowd evacuation. And the investigation of Guo[4] presented a mobile lattice gas model for simulating the pedestrian evacuation process in a public building. Also the research of Fang Zheng[7] demonstrated a spacial grid model that evaluates the movement of each individual during an emergency evacuation from building. Moreover, the study of Chang[2] developed an algorithm that sends evacuees to several exits through paths of shortest length.

In this particular case, we need to empty the Louvre as fast as possible. This is a complex problem with many behavioral and management facets (Perry, 1985; Vogt and Sorensen, 1992; Dow and Cutter, 1998; Drabek, 1999). A zone to evacuate must be agreed upon (Sorensen et al., 1992), shelters and exits must be designated (Sherali et al., 1991), and evacuees must be routed to safety under dynamic hazard and traffic conditions (MacGregor-Smith, 1991; Southworth, 1991). Any number of transportation problems can arise during an evacuation. For example, notifying evacuees may be difficult, traffic delays are common, and transportation lifelines are often compromised by the hazard.

1.2 Our work

In this particular case, we need to empty the Louvre as fast as possible. First of all, we establish the following two models for the proposed problem:

- The model, which finds k shortest paths between two nodes, using the Yen's algorithm[5].
- Time-evaluation model, which empties the Louvre with optimal solutions by evaluating evacuation time. We assessed the full evacuation time of the alternative schemes, and eventually, we concluded the most effective one.

Afterwards, we studied the feasibility of our solutions. Since there are a few cross parts of different routes in our solution, the problem of population flow congestion may

arise. To solve this problem, we considered the results from our first model (all possible shortest paths) and developed a function that filters out the optimal evacuation routes with intersection-part-number less than a certain threshold. With this optimization, the workload of our subsequent time-calculation is reduced to a large extent. Also we addressed a broad set of considerations, such as the inconvenience for disabled visitors when trying to exit the building, the response time of visitors after being informed of the emergency evacuation, and some tourists may not actually behave as what our solution suggested. We developed practical solutions for all the issues mentioned above.

2 Restatement of the Problem

What we should do to solve the problem:

- Develop an emergency evacuation model that has all occupants leave the building as quickly and safely as possible.
- Meanwhile, emergency personnel should be able to enter the building as quickly as possible.
- We should take the variation of the number of guests in the museum, the diversity of visitors (speaking different languages, groups traveling together, and disabled visitors), the use of additional exits, and various types of potential threats (terror attack, fire disaster, etc.) into consideration.

3 Assumptions and Justifications

- **Elevators are not available during the evacuation.** Elevators are electricity powered. If a fire hazard appears, or the terrorists shut off the power supply, people can be trapped in an elevator, while escalators can serve as stairs. For the safety of visitors, the stairs and escalators are the only approach to going to a lower or upper floor.
- **Each exit and each segment of a route has a maximum capacity limit.** That is, the number of visitors passing through a section of a path or an exit within certain time period is limited. Obviously, this is consistent with facts and common sense.
- **The pace of visitors in the straight way is constant.** To simplify our model, we assume the velocity of tourists in the straightway is constant, on condition that there is no congestion. Also, people slow down when making a turn.
- **Everyone follows the instruction.** To simplify our model, we assume all visitors will listen to the instructor. No one will deliberately go to the opposite direction of the population flow. After all, when a terror attack or fire disaster takes place, everyone would try to get out of the building by instinct.

4 Notation

Here are the symbols used in our model and their descriptions.

Symbols	Description
a	the length of the path
b	the width of the path
w	the weight of an edge
k	the number of shortest paths for each vertex
m_t	total number of vertices
m_{INO}	total number of exit vertices
T_c	threshold of the number of crossovers
v_i	a certain vertex on a certain path
c_{v_i}	number of crossovers at vertex v_i
C_t	total number of crossovers on an optimized path
d	size parameter of the unit grid
P	a finite set of places (in GSPN model)
T	a finite set of transitions (in GSPN model)
u_i	traveling speed in grid i
t_i	time needed for passing through grid i

5 The Model

We start with the requirements given in the topic: develop an emergency evacuation model to evacuate tourists from the Louvre as quickly as possible, and also allow emergency personnel to enter the building as quickly as possible.

The final output of the time calculation model is the evacuation time corresponding to each grid on each path selected by model 1.

For each path, the longest evacuation time of all grids is taken as the fully evacuation time of the path.

The route with the shortest fully-evacuation-time is selected as the optimal route under the situation of the current path condition and the number of people in the museum.

5.1 Model 1: Optimal Path Finding Model

By establishing the grid model and finding the shortest path between two points, we find out a variety of alternative evacuation routes.

5.1.1 Abstraction of the weighted undirected graph

Graph is a mathematical object that represents the relationship among objects. An undirected graph is a graph whose edges are not directed. Weighted means assigning a value to each edge, which represents the share and proportion of the edge.

The paths in the museum can be abstractly represented as weighted graphs. Since the paths in the museum are objectively bidirectional, so they are also undirected graphs. The corresponding meanings of edges, vertices and weights are as follows:

(1) **Edges and vertices**
Each edge represents a path in the Louvre. The length of a section of a path is measured along the direction of the path, while the width of the path is measured along the direction that is vertical to the direction of the population flow. A vertex means there is a change of path structure, including the change of the width of a path (which means that the width of the path represented by the arcs connected to the vertex are different), and the change of direction of a path. Also, stairs, escalators and exits are represented with vertices (we call them exit vertices for short afterwards).

(2) **Weight of an edge**
The cost of time passing through paths varies due to different path structures. We assign a value to each edge, that is, the weight of an edge, to stand for the cost. The following equation demonstrates the method to calculate weights:

$$w = a \ln(\frac{100}{b})$$

Where the weight is possessively correlated with the length a, and negatively correlated with the width b, which also matches the fact.

(3) **Virtual exit vertex**[2]
Virtual vertex is a family of special vertices. A virtual vertex does not represent a change of path structures. We assign one virtual vertex for each weighted undirected graph. For every weighted undirected graph, we connect each exit vertex to the virtual exit vertex with an edge. Those edges do not represent an actual path, and they all have the same weight. The value of the weight has no impact on the result of our model. During experiment process, the weight is 50.

Advantages of the virtual exit vertex: In the process of finding k shortest paths for a non-exit vertex, a virtual vertex can solve the problem that the shortest path may have different endpoints, which will be further discussed in the next subsection. Moreover, it can simplify the process of calculating evacuation time. To be specific, when computing evacuation time, the exit vertices can be treated the same as non-exit vertices, thus no additional algorithm is included. This will also be further discussed in the latter subsection. In this case, the virtual vertex has the similar function to that of the head node in linked list.

The specific weighted undirected graph depends on the specific architectural structure of the building. A graph is constructed on the basis of a series of connected exhibition areas. Disconnected paths in the museum are included in different weighted undirected graphs. For the structure of the Louvre discussed in this question, we draw undirected graphs according to the topographic map of the Louvre as follows: (due to the size of the figures, the weight of each edge is not indicated in the graph)

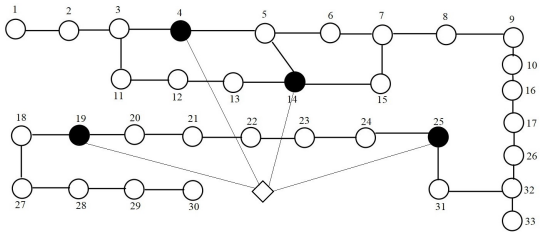

Figure 1: 2nd floor

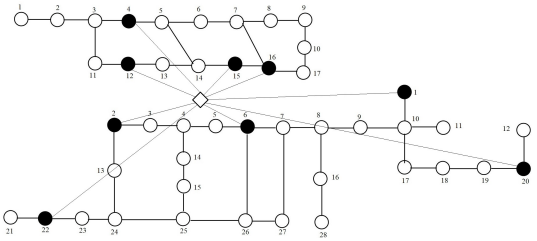

Figure 2: 1st floor

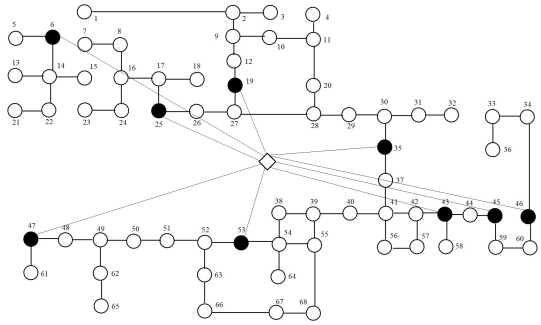

Figure 3: Ground floor

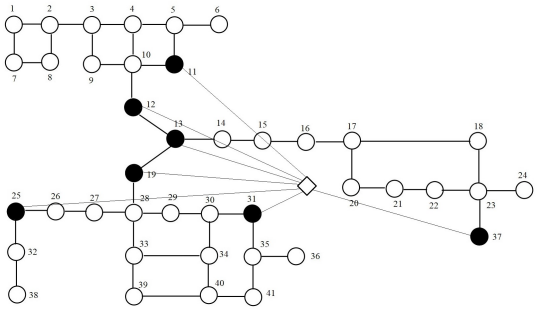

Figure 4: 1st underground floor

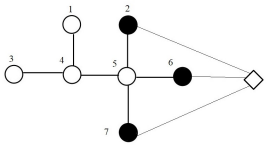

Figure 5: 2nd underground floor

The circles represent normal vertices, solid circles represent exit vertices, and the diamond represents the virtual exit vertex.

To meet the need for evacuating the museum as quickly as possible, we have to find an evacuation route that has the least fully evacuation time. There are numerable factors that affect the fully evacuation time. As we can see in the following discussion of our model, we considered the increase of crowd density caused by factors such as different selections of alternative evacuation routes, intersection of different routes, the direction-change of population flow, and the merge of population flow. We also considered possible reasons for time delay, such as the response time needed for visitors to understand what has happened, the change of path-structure during evacuation, congestions at exit gates, and the entering of emergency personnel to help keep things under control.

5.1.2 Find out k shortest paths for each non-exit vertex in every connected graph

(1) **Description of Yen's algorithm**

Yen's algorithm is one of derivation algorithms for ranking the k shortest loopless paths between a pair of nodes[5]. It always searches the shortest paths in a pseudo-tree containing k shortest loopless paths. In this problem, for a given integer $k \geq 1$, is intended to determine successively the shortest path, the second shortest path, ..., until the k-th shortest path between the given pair of nodes. The very shortest one

is obtained in the first place, and the second shortest path is always explored on the basis of the shortest paths that are shorter.

(2) **Implementation of Yen's algorithm in our model**
We apply Yen's algorithm for each non-exit vertex in the weighted undirected graphs. The output is the corresponding k shortest loopless paths. Total path number is $m_{INO} \times k$. The order of the results is: first the shortest path, then the rest $k-1$ paths in non-strict descending order.

After adjusting parameters several times, considering the complexity of time and space of the algorithm and the results of our model, in the case of Louvre, the performance of our model is the best when $k = 3$.

The starting node is a non-exit vertex, and the terminal node is the virtual exit vertex. We recursively call this method to traverse all non-exit vertices. In the process of obtaining k shortest paths of each vertex, the advantage of adding virtual exits is remarkable. We need to select k shortest loopless paths between a non-exit vertex and an exit vertex. In the case of not adopting the virtual exit vertex, we have to recursively modify the terminal node for m_{INO} times to find out k shortest loopless paths to a certain exit vertex for each non-exit vertex. As we can see, the process is very complex. However, if we add a virtual exit vertex that connects all exit vertices in a graph, we only need to set the virtual exit vertex as the terminal node to get the expected results. The reason is: the virtual exit vertex is connected to all exit vertices, such that all paths to the virtual exit vertex must correspond to one and only one actual exit vertex. Since the weight of each edge that is connected to the virtual exit vertex is constant, the edge has no actual contribution to the ranking of path length. Then the k shortest paths with virtual exit vertex as the terminal node correspond fully to the k shortest paths to all exit vertices. And the exit vertex that is contained in the route is also the exit that visitors will go to. In conclusion, in the process of calculating the k shortest paths for each non-exit vertex, the introduction of virtual exit vertex has the advantages of simplifying the algorithm and unifying the operation of vertices.

The output of the algorithm will be fully used in subsequent models. These paths will be arranged to synthesize a complete final evacuation route, and then serve as the basic data for subsequent optimal path filtering.

5.1.3 Select a path whose crossover number is less than the crossover threshold Tc

For all calculated k shortest paths of each non-exit vertex, there are inevitable crossovers. The following discusses the number of crossover for different combinations of those paths.

(1) **The case that has the least crossover number**
The analysis shows that in the $k^{m_{INO}}$ paths derived from the combination of $m_{INO} \times k$ paths, the route map composed of all the shortest paths must have the least number of crossings, and the least number of crossings is 0. The following is the explanation of why it is 0.

In the combination of all the shortest paths, there is no crossover, and the crossover number is 0. Each vertex corresponds to a shortest path starting from itself. Each

vertex of the shortest path corresponds to a shortest path to the nearest exit. That is to say, every time a new vertex is added to the path, the shortest path after the new vertex is the same as the shortest path corresponding to the new vertex itself. In this way, there will be no crossover except path overlap. In addition, according to the way we construct the path, there will only be overlap in the same direction, while there will not be overlap in the opposite direction. We classify the congestion problem caused by overlapping routes into the population flow congestion category in the time calculation model.

(2) **Normal case**

Except for the case where each path is the shortest path corresponding to the vertex, that is, the case discussed in (1), in most other cases, there will be intersections of routes. If a path starting from a vertex (assumed to be v_1) is not the shortest path, then the path starting from a subsequent vertex (assumed to be v_2) of the first vertex is not the shortest path. This will result in a situation where there are many different paths starting from v_2, and then considering the cases where there are many different segments of paths may end with v_2, we can conclude that there are two or more paths intersecting at this point.

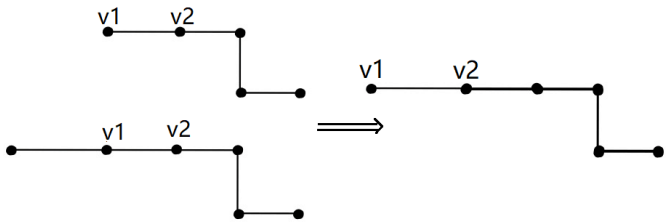

Figure 6: The case with no crossover

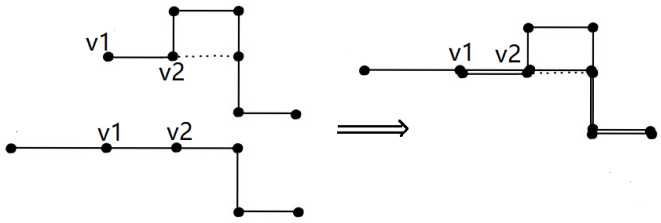

Figure 7: The case with one crossover

(3) **Taking the merge of population flow into consideration**

In the case where the number of crossover is the least, as stated in (1), there are chances that many paths will merge into one path. However, when we consider various factors affecting the evacuation time, we should not only take the number of crossover as the sole criterion. Meanwhile, we should consider the effect of the increase of the population density caused by the confluence. Therefore, we must balance the advantages of reducing time consumption resulted from the reduction of crossover-number, and the shortcomings of increasing time consumption caused by the increase of population flow density (caused by a large number of confluences). Therefore, we do not directly choose the path with the least number of crossover as the final path, instead, we include all the paths with crossover-number less than Tc

into the scope of investigation. We compare the severity of the impact caused by the above two factors and select the optimal path.

We will focus on the influence of the increase of the density of people on evacuation time due to changing of direction and confluence in the time calculation model.

(4) **The algorithm that filters the path with crossover-number less than** Tc
First: determine whether there is a crossover at a certain vertex.

Judgment condition: $t_i \neq 1 \parallel t_i' \neq 1$

The crossover-number c_{v_i} at vertex v_i follows two-point distribution:

$$c_{v_i} = \begin{cases} 1, & t_i \neq 1 \text{ or } t_i' \neq 1 \\ 0, & \text{otherwise} \end{cases}$$

The total number of crossover on the final route obtained by permutation and combination:

$$C_t = \sum_{i=0}^{m_t} c_{v_i}$$

Filter the path based on C_t:

$$C_t \leq T_c \quad (t \leq k^{m_{INO}})$$

5.2 Model 2: Time Calculation Model

After the establishment of the first model, several evacuation plans with the least intersection points are selected by using a function that takes all feasible paths into consideration and then filters out the optimal ones. In order to evaluate the advantages and disadvantages of several plans with the same amount of intersection points, a time calculation model is established to find the fastest and most effective evacuation route.

5.2.1 Grid partition of passageway

(1) **Description of the original grid model**
The grid model established a numerical model that can by applied to assessing each individual's movement. The model splits the building using plane grids that can describe the location of every occupant at any time. The moving velocity and direction of an occupant are dominated by the characteristics in the grids. The grid model demonstrated a method for calculating the motion mode and exact evacuation time of each visitor[7].

Crowd evacuation is a group behavior, also there are constraints like the initial location of a person and the relationship between individuals. In the process of establishing the evacuation model, considering the position and speed of each person at any time, the Lagrangian method is adopted to describe the individual's trajectory:

$$x(m,t) = x(m,t-1) + U_m \triangle t \tag{1}$$

$$y(m,t) = y(m,t-1) + V_m \triangle t \tag{2}$$

where $m = 1, 2, \ldots, N, x(m,t), y(m,t)$ represent the location of individual m at time t, U_m, V_m represents the individual's velocity of x, y components at time t respectively.

Assume that the speed of a person's movement is only related to his geometric location and the population density within a certain range of the location.

(2) **Implementation of the grid model in our model**

In our model, we use the grid model to estimate the characteristics of the density of population flow in a specific region. With these key factors fully considered, a grid will function as the smallest unit in the following time cost calculation.

Considering the influence of many factors, such as the actual size of the museum and the actual movement of the visitors, our team has come to a conclusion that the optimal size of the selected grid is 2×2 when the Louvre is used as the test object. That is, $d = 2$. The factors taken into account include the huge time cost of algorithm caused by excessively small grid-selection, and large error with actual travel situation of population flow caused by excessively large grid-selection, which will lead to model application failure.

5.2.2 The time calculation model

In the mathematical model of calculating the specific evacuation time, we consider the following factors:

a. People's rational behaviors, such as alert to emergencies and delay of response, will have an objective and unavoidable impact on evacuation time. The influence of these factors on the fully evacuation time is reflected and quantified by General Stochastic Petri nets (GSPN).

b. The influence of the population flow in a certain grid on the number of people in the next grid while traveling. And then it includes the influence of the change of number of people in the grid on the change of the travel speed of visitors in the grid. We introduce Little's Law and the core mathematical model of time model to deal with this factor.

c. At the same time, we consider the time loss caused by confluences, change of direction, crossovers and the impact of emergency personnel entering the museum on the evacuation of tourists. We still evaluate these losses with the above theorem and core model.

(1) **GSPN model**

GSPN is the abbreviation of Generalized stochastic petri nets, providing a simple way to mathematically model the evacuation process. The proposed evacuation model takes into account human behavior parameters such as panic among occupants, and realistic situations namely stampede in the stairwell, and the time cost of finding the guided evacuation path. The adequacy of Petri net types to the study of various problems related to dependability such as risk analysis and probabilistic assessment has been proved in literature.[6]

A stochastic petri net consists two types of nodes called places and transitions. Places are represented as circles, transitions are represented as rectangles, arcs are represented as arrows. The classical Petri net is a directed bipartite graph as in the following figure:

Transitions model activities (events). Events may take place atomically (the transition fires) and change the system state.

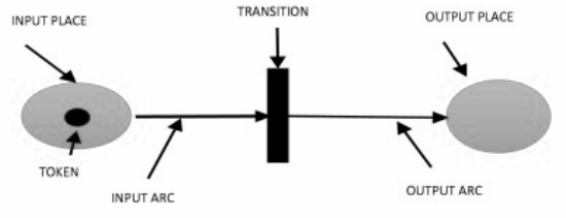

Figure 8: Classical petri net

Input arcs are directed arcs drawn from places to transitions, representing the conditions that need to be satisfied for the event to be activated. Output arcs are directed arcs drawn from transitions to places, representing the conditions resulting from the occurrence of an event.

The collection of places that are connected to the transitions through input arcs are called the input places. The collection of places to which output arcs exist from the transitions are called the output places.

Tokens are dots associated with places; A place containing tokens indicates that the corresponding condition is active. When input places of a transition has the required number of tokens, the transition is enabled.

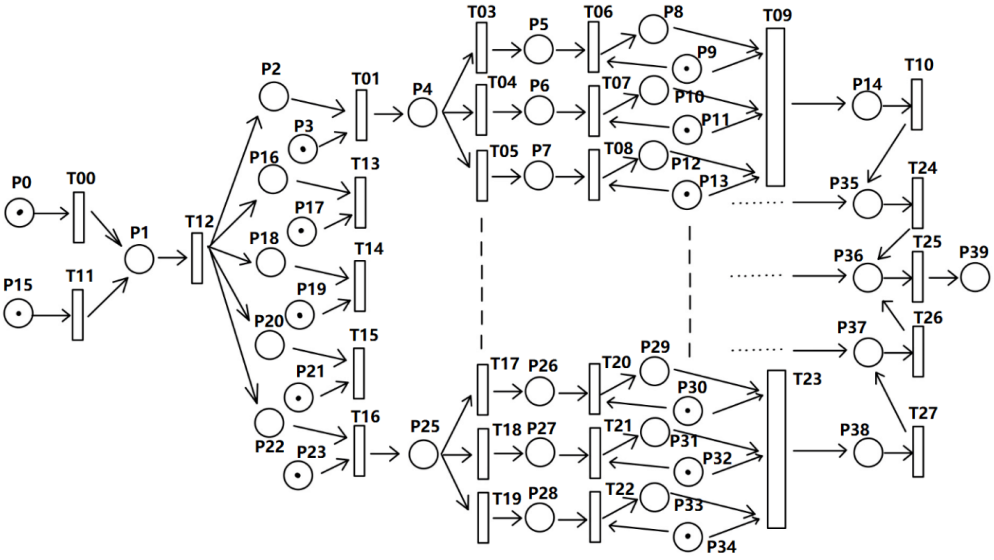

Figure 9: The petri net of our model

Some of the reasons such as terrorist on P_0, on P_15 are considered in the model. A token gets deposited in P1 which indicates the activation of a transition. Attack alert (on P1) immediately activates the transition T_{12} and deposits tokens on $P_2, P_{16}, P_{18}, P_{20}, P_{22}$. An initial number of evacuees indicated by tokens in P_3, P_{17}, P_{19}, P_{21}, P_{23} move at a certain exponential rate to the guided path. Once the evacuees step on the guided path as indicated by tokens in P14 they proceed accessing the next floor and are assumed to be safely evacuated. The detailed meaning of these representations are as Table 1 and Table 2.

Table 1: Significance of places

Place	Meaning
P_0, P_{15}	Terrorist attack
P_1	Detect an alarm signal
$P_2, P_{16}, P_{18}, P_{20}, P_{22}$	Attack alert
$P_3, P_{17}, P_{19}, P_{21}, P_{23}$	Evacuees
P_4, P_{25}	Evacuees ready to move
P_5, P_{26}	Panic monitoring
P_6, P_{27}	Evacuation path monitoring
P_7, P_{28}	Stampede monitoring
P_8, P_{29}	Panic recognition
P_9, P_{30}	Panic assistance
P_{10}, P_{31}	Match with path guided
P_{11}, P_{32}	Finding guided path
P_{12}, P_{33}	Stampede occurrence
P_{13}, P_{34}	Stampede clearance
$P_{14}, P_{35}, P_{37}, P_{38}$	Evacuation of the floor
P_{36}	Evacuation of the ground floor
P_{39}	Safe area

Table 2: Significance of transitions

Transition	Meaning
P_0, P_{15}	Terrorist attack
P_1	Detect an alarm signal
$P_2, P_{16}, P_{18}, P_{20}, P_{22}$	Attack alert
$P_3, P_{17}, P_{19}, P_{21}, P_{23}$	Evacuees
P_4, P_{25}	Evacuees ready to move
P_5, P_{26}	Panic monitoring
P_6, P_{27}	Evacuation path monitoring
P_7, P_{28}	Stampede monitoring
P_8, P_{29}	Panic recognition
P_9, P_{30}	Panic assistance
P_{10}, P_{31}	Match with path guided
P_{11}, P_{32}	Finding guided path
P_{12}, P_{33}	Stampede occurrence
P_{13}, P_{34}	Stampede clearance
$P_{14}, P_{35}, P_{37}, P_{38}$	Evacuation of the floor
P_{36}	Evacuation of the ground floor
P_{39}	Safe area

The application of GSPN in this model can better reflect the impact of objective human factors on evacuation time.

(2) **Little's Law**

In queuing theory, Little's law is a theorem which states that the long-term average number L of individuals in a stationary system is equal to the long-term average effective arrival rate λ multiplied by the average time W that an individual spends

in the system. Expressed algebraically the law is

$$L = \lambda W$$

The implementation of Little's Law will be discussed in the following section.

(3) **Core of the time calculating model**

We will get the corresponding evacuation time for each grid, and repeat the following steps through the evacuation route starting from each grid. Starting with the initial grid discussed in the current iteration, the indexes are $0, 1, 2, \ldots$, corresponding to the subscripts of the number of people and their traveling speed.

a. Initial data:

The initial number of people in each grid that the evacuation path passes through is obtained by monitoring the data at the scene of the terror attack. The initial number of visitors passing through the grid is $n_1, n_2, n_3 \ldots$. From the functional relationship between the number of people in the grid and the flow velocity of people in the grid (Equation 3), the initial flow velocity of people in each grid is $u_1, u_2, u_3 \ldots$.

$$u = \begin{cases} 2, & \text{if } n < 12 \\ 2 - \frac{(n-12)^2}{100}, & \text{if } n \geq 12 \end{cases} \tag{3}$$

For ease of illustration, the traveling speed of the people at the front and the end of a grid are shown in the figure. By default, people in different locations in the same grid travel at the same speed every minute. The initial situation is shown in the following figure.

A grid is added at the end of each path, and the initial number of people in the grid is the same as the number of people in the penultimate grid. The role of this additional grid is to unify the processing of the last grid with that of other grids, which is similar to the role of the virtual exit vertex.

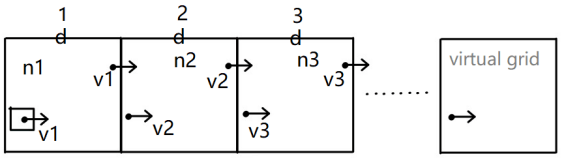

Figure 10: Initial state

b. Total evacuation time of grid 1:

$$T = \sum_{i=1} t_i$$

We calculate t_1 first:

$$t_1 = \frac{d}{u_1}$$

c. Update the number of people n and traveling speed v for all subsequent grids.

Because the number of people in each grid is different and the speed of population flow in the grid is different, the number of people in each grid will change after a period of time, and then the speed of population flow will also change. Therefore,

when the observation object is passing through the second grid, those two factors cannot be calculated directly based on the initial data. Moreover, it has effects on the number of people and traveling speed not only in the second grid, but also in the grids that are in front of the observation object, and in each iteration of time calculation, which depends on the results of the former one. Therefore, in each iteration, we need to update the number of people N and population flow velocity V of all subsequent grids on the path at the same time.

d. Calculate t_2.
By Little's Law, we know that n_2' is related with u_1, u_2 and n_2, the relation is demonstrated as follows:

$$n_2' = (u_2 - u_1) \times \frac{d}{u_2} + n_2$$

According to the formulas of the number of people and the speed of people in the grid (Equation 3), the traveling velocity u_1' of grid 2 is changed due to the population flow in grid 1.
The traveling speed of the observation object in grid 2 should be the same as u_2'.
We can obtain

$$t_2 = \frac{d}{u_2'}$$

Similarly, repeat step c.

e. Perform similar operations on all subsequent grids on the path (except the last grid of each path) as on grid 2 in d. We can obtain $t_3 \ldots$

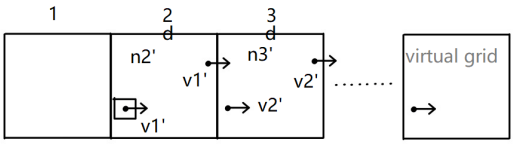

Figure 11: After an update

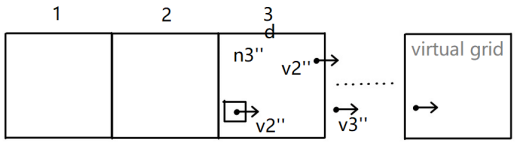

Figure 12: After two updates

f. Get the summation of t_i, we can obtain the evacuation time corresponding to grid 1: $T = \sum_{i=1} t_i$.
Perform the above operations on all grids that are not connected to the exit, we can get the evacuation time for all those grids.

(4) **The time loss caused by confluences**
Confluences are mainly resulted from change of direction and intersection of different routes, which can be abstracted as integration and separation of population flow. The influence of integration/separation on time calculation is reflected in the following two situations: many population flows from different directions coming into the confluence grid, or separating the population flow at the separation grid into several sub-flows that go to different directions. We take the confluence of population flow as an example to illustrate our corresponding measures to this kind of time loss.

The grids at the nearest confluence of the two paths are numbered 1 and 2 respectively, the grid at the confluence is numbered 3, and the first grid after the confluence is numbered 4. The situation at the confluence is shown in the figure below. In this model, we assume that the traveling speed of people coming from two different

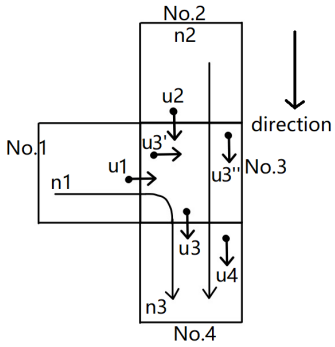

Figure 13: Confluence state

directions after entering the confluent grid is u_3', u_3'', respectively. According to the principle of the model, we have

$$u_3 = u_3' = u_3''$$

Therefore, for grids 1, 2, and 4, the calculation method of traveling speed and number of people is the same as that in (3), since our original model can provide the data needed for calculation.

For grid 3, because the import speed is not unique, it is worth considering separately. In the original time calculation model mentioned above, for each grid, there is a unique precursor grid and a unique successor grid, so the speed of entering and leaving any grid is also unique. Thus we merge the inflow velocities from grids 1and 2 into a velocity along the direction indicated by the arrow in the graph. The principle of velocity synthesis is described below.

In this model, the speed is a function of the number of people, and the number of people is an independent variable. Therefore we consider:
Real-time increase in the number of people in grid 3: $\triangle n_1 = (u_1 + u_2) \times t$
Real-time decrease in the number of people in grid 3: $\triangle n_2 = u_3 \times t$
Number of people in grid 3 after a time period t: $n_3' = n_3 + \triangle n_1 - \triangle n_2$
We can get u_3' by the equation 3
The cost of time passing through grid 3 is: $t_3 = \frac{d}{u_3}$

(5) **The impact of emergency personnel entering the museum on evacuation time**

The entry of emergency personnel will have a certain impact on the evacuation time. In our model, the path through which emergency personnel travel from the outside of the Louvre to the place where the emergency occurs is the same as the reverse path of the individual evacuation route at the site of the incident. Thus it will inevitably lead to the increase of the crowd density along the path. The increase is caused by the admission of emergency personnel.

In order to take the increase into account and combine with the actual rescue situation, we add a certain number of people in each grid on the path of emergency personnel to simulate the increase of population flow density. In this way, we do not have to consider the special impact of the emergency personnel on the whole time calculation model, instead, we replace this change by increasing the density of population flow in the grid.

6 Sensitivity Analysis

6.1 The influence of the number of people in the museum on the quickest evacuation time

According to the relationship between the simulated traveling speed u and the number of people n in a grid

$$u = \begin{cases} 2, & \text{if } n < 12 \\ 2 - \frac{(n-12)^2}{100}, & \text{if } n \geq 12 \end{cases}$$

We can draw an image of the speed u with respect to the number of people n as follows:

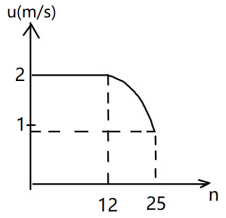

Figure 14: The relation diagram of u and v

As can be seen from the figure above, when the number of people in a grid is greater than 12, the speed of people's travel is slowing down with the increase of the number of people, and the more the number of people, the greater the degree of slowdown for each additional person.

We simulate the following two situations: the first is that the number of people in each grid is less than 10, and the other is that the number of people in each grid is more than 20.

Therefore, we change the number of people for many times (in each change, each grid increases the same number of people), and calculate the corresponding evacuation time in each case. The results and corresponding parameters are as follows:

Vector $\vec{N}$ is used to identify the vector consisting of initial number of people in each grid. At this point, we get the evacuation time 349.8s.

And the evacuation time is 380.1s with the parameter $\vec{N} + 2$ (here, each element in the representative vector is added 2).

After that, the parameters were operated on the same approach as above, and the evacuation time is 417.9 s, 459.4 s, 511.8 s and 590.2 s respectively.

In conclusion, we know that the number of people in the museum has a great influence on the final evacuation time.

6.2 The influence of mesh fineness on the quickest evacuation time

In order to reduce the computational complexity, we set the parameters of a unit grid to be 2×2. If the museum is applied to the 2×2 model, the quickest evacuation time we can get is 349.8 s.

Moreover, we ignored the differences among people, instead, we consider all people in a certain region as a whole.

But if we refine the partition furthermore, though the amount of calculation will be very large, the data obtained in this way will be closer to the actual situation. For example, if we set the size of unit grid to be 1×1, and after calculation, we get another evacuation time, which is 354.6 s.

From the above analysis, it can be concluded that the setting of unit grid parameters (i.e. partition fineness) will also affect the final calculation of the quickest evacuation time when using the grid model.

7 Conclusions

In this section, we present our solution to the proposed problem. First, we build an undirected graph for each floor. The weight of each arc is related with the length and breadth of each segment of paths. Then we use the k-shortest-path algorithm to find out k shortest paths between a node and an exit. We introduced a virtual exit node that is the terminal node of all exit nodes in each floor to simplify and optimize our algorithm. To avoid the intersection of different routes, we add a method that filters all the shortest paths without or with minimum crossing parts. By now, we have successfully generated a solution that evacuates all visitors in the Louvre through the shortest paths. Afterwards, we design another algorithm to calculate the fully evacuation time, and after evaluating the time needed for previous routes, we can select the optimal evacuation route for visitors in the Louvre.

7.1 Results

In our paper, we have completed the task given by the subject.

First of all, we abstract the actual path structure of the museum and obtain the weighted undirected graphs. Input the graphs and then process them with the model we established, we obtain several evacuation schemes with total route intersections less than a certain threshold. We put these evacuation strategies into the time calculation model, and then process the random number of people in each unit grid generated by matlab. After filtering, we get the solution with shortest evacuation time. In the process of data processing, we generate the random number table, and then obtain the final evacuation time of 349.8 s.

Then we carry out sensitivity analysis and get the following two conclusions: the traveling speed slows down with the increase of the number of people in one grid, and the larger the number is, the more the speed of each person is slowed down; the better the fineness of the grid is, the closer the evacuation time is to the real situation.

Here is the best evacuation map of 2nd floor. The rest can be obtained using similar approaches.

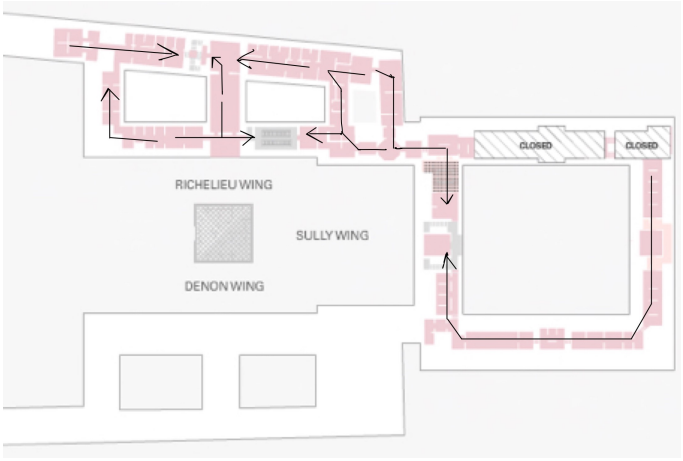

Figure 15: Evacuation map of 2nd floor

7.2 Further discussion

There are several improvements of our model that can be made in the future:

- There are inevitable intersection points of different paths in our evacuation model. In order to minimize the time delay caused by this problem, we can adopt the network flow model for lane-based evacuation routing[3], which includes minimum-cost flow model, network simplex method, and manual capacity analysis. They are helpful for reducing crossing-conflicts at intersections, and merging the population flow at intersection points.

- The grid-size in our grid model is 2×2, which contains more than one person. Consequently, our evacuation route is not very exact. Thus, we can reduce the size to 0.4×0.4, which contains exactly one person. Then we can get each visitor's movement route, such that we can come up with a more detailed, personal evacuation plan.

- We can test our model using more accurate data, and find out a more realistic approach and equations to obtain a more precise evacuation time. For instance, the velocity of visitors when trying to get out of the building can be expressed as a function $U = f(x, y)$, where x, y are the geometric location of a person. In crowded circumstances, a visitor that is closest to an exit goes first, and the person whose distance from the visitor is within a certain value r stays put or goes to the other sides. Also we can develop an accurate equation that represents the relation between the walking velocity of visitors and the crowd density.

- We try to make appropriate changes in the method of finding k shortest paths, such as using different algorithms: changing Dijkstra algorithm to Floyd algorithm for calculation, or using improved methods such as skipping nodes without branches to improve the existing model[1]. Finally, we evaluate the evacuation time of different schemes, and find out the optimal algorithm and establish the optimal solution model.

- Apply our model to other complex architectures to acquire the adaptability of our model for all kinds of large architectures, and make appropriate improvements. Therefore, our model will be suitable for a variety of buildings and emergency events.

8 Strengths and Weaknesses

8.1 Strengths

- **Our model fully considers the coordination of all sides in the actual evacuation situation.** In the case that the fully evacuation time of tourists is the shortest, the optimal route for emergency personnel to enter the museum can also be organized at the same time, so as to ensure that the overall interests in the evacuation process are the largest in the actual situation.
- **Our model quantifies various potential bottlenecks** and factors that may limit the movement towards the exit in the real-time situation, and takes into account the impact of these factors on the actual evacuation time. All kinds of time uncertainties caused by tourists' response time delay and congestion are also included in the scope of our investigation.
- **Our model has good adaptability to various emergency situations.** It can cope with potential threats such as the change of evacuation route caused by the structural change of passageway in the museum (such as the damage of passageway caused by a fire hazard or terror attack).
- **Our model has strong universality.** It does not rely on the architectural structure of specific buildings, but interactively abstracts the architectural structure of various buildings as the input of this model, which means that it can also function well in the evacuation of other large crowded buildings.

8.2 Weaknesses

However, our model is not without weakness.

- We ignored the fact that visitors may be blocked by people in front of them while walking along the aisle, which leads to a time delay. Consequently, the calculated evacuation time may not be strictly accurate.
- We assumed that the moving direction of people in one grid is consistent. That is, we did not considered the situation where visitors may go to random directions in a grid. It will also lead to evacuation time inaccuracy.
- We used some formulas and data that are defined by ourselves, which are reasonable, but cannot fully match the fact.

9 Suggestions to the Louvre

In order to ensure a safe and effective evacuation in case of emergency, we put forward the following suggestions for the emergency management of the Louvre:

- Hand out a smart bracelet that is equipped with GPS utility for each visitor before entering the Louvre. During a normal tour route, it can inform the staff in case the visitor is in need of help. And during emergency evacuation, it can tell visitors their evacuation route, such that the evacuation process can proceed effectively.
- In the event of an emergency, inform all security personnel in time, and implement security work on each main road where tourists evacuate, so as to prevent tourists from rioting and trampling uncontrollably during the evacuation process, resulting in unnecessary casualties or damage to cultural relics.
- During emergencies, the existing online app "affluences" should be used to inform the visitors who have not yet entered the Louvre that there is an emergency, so that the visitors outside the museum can stop entering, therefore avoid the inconvenience for the evacuation caused by the increase of the number of visitors inside the museum.
- Inform tourists of all secret passages except those occupied by emergency personnel through the bracelet, so as to improve the efficiency of evacuation.

References

[1] `https://en.wikipedia.org/wiki/Yen%27s_algorithm#Improvements`.

[2] Liu Chang, F. U. Zhi-Min, and Mao Zhan-Li. Emergency evacuation model and algorithm in the building with several exits. *Fire Science and Technology*, 2015.

[3] Thomas J Cova and Justin P Johnson. A network flow model for lane-based evacuation routing. *Transportation Research Part A*, 37(7):579–604, 2015.

[4] R. Y. Guo and H. J. Huang. A mobile lattice gas model for simulating pedestrian evacuation. *Physica A-statistical Mechanics & Its Applications*, 391(3):582–592, 2012.

[5] Ernesto Q. V. Martins and Marta M. B. Pascoal. A new implementation of yens ranking loopless paths algorithm. *Quarterly Journal of the Belgian French & Italian Operations Research Societies*, 1(2):121–133, 2003.

[6] Angel A. Sheeba and R. Jayaparvathy. Performance modeling of an intelligent emergency evacuation system in buildings on accidental fire occurrence. *Safety Science*.

[7] Fang Zheng and Prof Lu Siuming. A spatial grid model for emergency evacuation from building. *China Safety Science Journal*, 2001.

[8] Xiaoping Zheng, Tingkuan Zhong, and Mengting Liu. Modeling crowd evacuation of a building based on seven methodological approaches. *Building & Environment*, 44(3):437–445, 2009.

Appendices

Appendix A

Here are simulation programs we used in our model.

Core Java source code:

```
/*
 *
 * Copyright (c) 2004–2008 Arizona State University.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY ARIZONA STATE UNIVERSITY ``AS IS'' AND
 * ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL ARIZONA STATE UNIVERSITY
 * NOR ITS EMPLOYEES BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */

package icm;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Vector;

import icm.Graph;
import icm.Path;
import icm.VariableGraph;
import icm.BaseGraph;
import icm.BaseVertex;
import icm.Pair;
import icm.QYPriorityQueue;

/**
 * @author <a href='mailto:Yan.Qi@asu.edu'>Yan Qi</a>
 * @version $Revision: 783 $
 * @latest $Id: YenTopKShortestPathsAlg.java 783 2009–06–19 19:19:27Z qyan $
 */
```

```
public class YenTopKShortestPathsAlg
{
    private VariableGraph graph = null;

    // intermediate variables
    private List<Path> resultList = new Vector<Path>();
    private Map<Path, BaseVertex> pathDerivationVertexIndex = new HashMap<Path, BaseVertex>();
    private QYPriorityQueue<Path> pathCandidates = new QYPriorityQueue<Path>();

    // the ending vertices of the paths
    private BaseVertex sourceVertex = null;
    private BaseVertex targetVertex = null;

    // variables for debugging and testing
    private int generatedPathNum = 0;

    /**
     * Default constructor.
     *
     * @param graph
     * @param k
     */
    public YenTopKShortestPathsAlg(BaseGraph graph) {
        this(graph, null, null);
    }

    /**
     * Constructor 2
     *
     * @param graph
     * @param sourceVertex
     * @param targetVertex
     */
    public YenTopKShortestPathsAlg(BaseGraph graph,
            BaseVertex sourceVertex, BaseVertex targetVertex)   {
        if (graph == null) {
            throw new IllegalArgumentException("A NULL graph object occurs!");
        }
        this.graph = new VariableGraph((Graph)graph);
        this.sourceVertex = sourceVertex;
        this.targetVertex = targetVertex;
        init();
    }

    /**
     * Initiate members in the class.
     */
    private void init() {
        clear();
        // get the shortest path by default if both source and target exist
        if (sourceVertex != null && targetVertex != null) {
            Path shortestPath = getShortestPath(sourceVertex, targetVertex);
            if (!shortestPath.getVertexList().isEmpty()) {
                pathCandidates.add(shortestPath);
                pathDerivationVertexIndex.put(shortestPath, sourceVertex);
            }
        }
    }

    /**
     * Clear the variables of the class.
```

```
     */
    public void clear() {
        pathCandidates = new QYPriorityQueue<Path>();
        pathDerivationVertexIndex.clear();
        resultList.clear();
        generatedPathNum = 0;
    }

    /**
     * Obtain the shortest path connecting the source and the target, by using the
     * classical Dijkstra shortest path algorithm.
     *
     * @param sourceVertex
     * @param targetVertex
     * @return
     */
    public Path getShortestPath(BaseVertex sourceVertex, BaseVertex targetVertex)   {
        DijkstraShortestPathAlg dijkstraAlg = new DijkstraShortestPathAlg(graph);
        return dijkstraAlg.getShortestPath(sourceVertex, targetVertex);
    }

    /**
     * Check if there exists a path, which is the shortest among all candidates.
     *
     * @return
     */
    public boolean hasNext() {
        return !pathCandidates.isEmpty();
    }

    /**
     * Get the shortest path among all that connecting source with targe.
     *
     * @return
     */
    public Path next() {
        //3.1 prepare for removing vertices and arcs
        Path curPath = pathCandidates.poll();
        resultList.add(curPath);

        BaseVertex curDerivation = pathDerivationVertexIndex.get(curPath);
        int curPathHash =
            curPath.getVertexList().subList(0, curPath.getVertexList().indexOf(curDerivation)).

        int count = resultList.size();

        //3.2 remove the vertices and arcs in the graph
        for (int i = 0; i < count-1; ++i) {
            Path curResultPath = resultList.get(i);

            int curDevVertexId =
                curResultPath.getVertexList().indexOf(curDerivation);

            if (curDevVertexId < 0) {
                continue;
            }

            // Note that the following condition makes sure all candidates should be considered
            /// The algorithm in the paper is not correct for removing some candidates by mista
            int pathHash = curResultPath.getVertexList().subList(0, curDevVertexId).hashCode();
            if (pathHash != curPathHash) {
```

```
            continue;
        }

        BaseVertex curSuccVertex =
            curResultPath.getVertexList().get(curDevVertexId + 1);

        graph.deleteEdge(new Pair<Integer, Integer>(
                curDerivation.getId(), curSuccVertex.getId()));
    }

    int pathLength = curPath.getVertexList().size();
    List<BaseVertex> curPathVertexList = curPath.getVertexList();
    for (int i = 0; i < pathLength-1; ++i) {
        graph.deleteVertex(curPathVertexList.get(i).getId());
        graph.deleteEdge(new Pair<Integer, Integer>(
                curPathVertexList.get(i).getId(),
                curPathVertexList.get(i + 1).getId()));
    }

    //3.3 calculate the shortest tree rooted at target vertex in the graph
    DijkstraShortestPathAlg reverseTree = new DijkstraShortestPathAlg(graph);
    reverseTree.getShortestPathFlower(targetVertex);

    //3.4 recover the deleted vertices and update the cost and identify the new candidate r
    boolean isDone = false;
    for (int i=pathLength-2; i>=0 && !isDone; --i)  {
        //3.4.1 get the vertex to be recovered
        BaseVertex curRecoverVertex = curPathVertexList.get(i);
        graph.recoverDeletedVertex(curRecoverVertex.getId());

        //3.4.2 check if we should stop continuing in the next iteration
        if (curRecoverVertex.getId() == curDerivation.getId()) {
            isDone = true;
        }

        //3.4.3 calculate cost using forward star form
        Path subPath = reverseTree.updateCostForward(curRecoverVertex);

        //3.4.4 get one candidate result if possible
        if (subPath != null) {
            ++generatedPathNum;

            //3.4.4.1 get the prefix from the concerned path
            double cost = 0;
            List<BaseVertex> prePathList = new Vector<BaseVertex>();
            reverseTree.correctCostBackward(curRecoverVertex);

            for (int j=0; j<pathLength; ++j) {
                BaseVertex curVertex = curPathVertexList.get(j);
                if (curVertex.getId() == curRecoverVertex.getId()) {
                    j=pathLength;
                } else {
                    cost += graph.getEdgeWeightOfGraph(curPathVertexList.get(j),
                            curPathVertexList.get(j+1));
                    prePathList.add(curVertex);
                }
            }
            prePathList.addAll(subPath.getVertexList());

            //3.4.4.2 compose a candidate
            subPath.setWeight(cost + subPath.getWeight());
```

```
                subPath.getVertexList().clear();
                subPath.getVertexList().addAll(prePathList);

                //3.4.4.3 put it in the candidate pool if new
                if (!pathDerivationVertexIndex.containsKey(subPath)) {
                    pathCandidates.add(subPath);
                    pathDerivationVertexIndex.put(subPath, curRecoverVertex);
                }
            }

            //3.4.5 restore the edge
            BaseVertex succVertex = curPathVertexList.get(i + 1);
            graph.recoverDeletedEdge(new Pair<Integer, Integer>(
                    curRecoverVertex.getId(), succVertex.getId()));

            //3.4.6 update cost if necessary
            double cost1 = graph.getEdgeWeight(curRecoverVertex, succVertex)
                + reverseTree.getStartVertexDistanceIndex().get(succVertex);

            if (reverseTree.getStartVertexDistanceIndex().get(curRecoverVertex) >
cost1) {
                reverseTree.getStartVertexDistanceIndex().put(curRecoverVertex, cost1);
                reverseTree.getPredecessorIndex().put(curRecoverVertex, succVertex);
                reverseTree.correctCostBackward(curRecoverVertex);
            }
        }

        //3.5 restore everything
        graph.recoverDeletedEdges();
        graph.recoverDeletedVertices();

        return curPath;
    }

    /**
     * Get the top-K shortest paths connecting the source and the target.
     * This is a batch execution of top-K results.
     *
     * @param source
     * @param sink
     * @param k
     * @return
     */
    public List<Path> getShortestPaths(BaseVertex source,
                                       BaseVertex target, int k) {
        sourceVertex = source;
        targetVertex = target;

        init();
        int count = 0;
        while (hasNext() && count < k) {
            next();
            ++count;
        }

        return resultList;
    }

    /**
     * Return the list of results generated on the whole.
     * (Note that some of them are duplicates)
```

```
     * @return
     */
    public List<Path> getResultList() {
        return resultList;
    }

    /**
     * The number of distinct candidates generated on the whole.
     * @return
     */
    public int getCadidateSize() {
        return pathDerivationVertexIndex.size();
    }

    public int getGeneratedPathSize() {
        return generatedPathNum;
    }
}
```

```
/*
 *
 * Copyright (c) 2004–2008 Arizona State University.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * THIS SOFTWARE IS PROVIDED BY ARIZONA STATE UNIVERSITY ``AS IS'' AND
 * ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
 * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL ARIZONA STATE UNIVERSITY
 * NOR ITS EMPLOYEES BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
 * DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
 * THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 * (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 * OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 *
 */

package icm;

import java.util.List;

import icm.Graph;
import icm.Path;
import icm.VariableGraph;
import icm.DijkstraShortestPathAlg;
import icm.YenTopKShortestPathsAlg;
```

```
/**
 * TODO Need to redo!
 * @author <a href='mailto:Yan.Qi@asu.edu'>Yan Qi</a>
 * @version $Revision: 784 $
 * @latest $Id: YenTopKShortestPathsAlgTest.java 46 2010-06-05 07:54:27Z yan.qi.asu $
 */
public class YenTopKShortestPathsAlgTest {
    // The graph should be initiated only once to guarantee the correspondence
    // between vertex id and node id in input text file.
    private static Graph graph = new VariableGraph("C:\\Users\\malechi\\Desktop\\k-shortest-pat

//  @Test
    public void testDijkstraShortestPathAlg()
    {
        System.out.println("Testing Dijkstra Shortest Path Algorithm!");
        DijkstraShortestPathAlg alg = new DijkstraShortestPathAlg(graph);
        System.out.println(alg.getShortestPath(graph.getVertex(4), graph.getVertex(5)));
    }

//  @Test
    public void testYenShortestPathsAlg()
    {
        System.out.println("Testing batch processing of top-k shortest paths!");
        YenTopKShortestPathsAlg yenAlg = new YenTopKShortestPathsAlg(graph);
        List<Path> shortest_paths_list = yenAlg.getShortestPaths(
                graph.getVertex(4), graph.getVertex(5), 100);
        System.out.println(":"+shortest_paths_list);
        System.out.println(yenAlg.getResultList().size());
    }

//  @Test
    public void testYenShortestPathsAlg2()
    {
        System.out.println("Obtain all paths in increasing order! - updated!");
        YenTopKShortestPathsAlg yenAlg = new YenTopKShortestPathsAlg(
                graph, graph.getVertex(4), graph.getVertex(5));
        int i=0;
        while(yenAlg.hasNext())
        {
            System.out.println("Path "+i+++" : "+yenAlg.next());
        }

        System.out.println("Result # :"+i);
        System.out.println("Candidate # :"+yenAlg.getCadidateSize());
        System.out.println("All generated : "+yenAlg.getGeneratedPathSize());
    }

    public void testYenShortestPathsAlg4MultipleGraphs()
    {
        System.out.println("Graph 1 - ");
        YenTopKShortestPathsAlg yenAlg = new YenTopKShortestPathsAlg(
                graph, graph.getVertex(4), graph.getVertex(5));
        int i=0;
        while(yenAlg.hasNext())
        {
            System.out.println("Path "+i+++" : "+yenAlg.next());
        }

        System.out.println("Result # :"+i);
        System.out.println("Candidate # :"+yenAlg.getCadidateSize());
        System.out.println("All generated : "+yenAlg.getGeneratedPathSize());
```

```
        ///
        System.out.println("Graph 2 – ");
        ///the following is the absolute path of the weighted undirected graph
        graph = new VariableGraph("C:\\Users\\malechi\\Desktop\\k–shortest–paths–java–version–m
        YenTopKShortestPathsAlg yenAlg1 = new YenTopKShortestPathsAlg(graph);
        List<Path> shortest_paths_list = yenAlg1.getShortestPaths(
                graph.getVertex(4), graph.getVertex(5), 10);
        System.out.println(":"+shortest_paths_list);
        System.out.println(yenAlg1.getResultList().size());
    }
}
```

Appendix B

Here are the weighted undirected graphs of each floor:

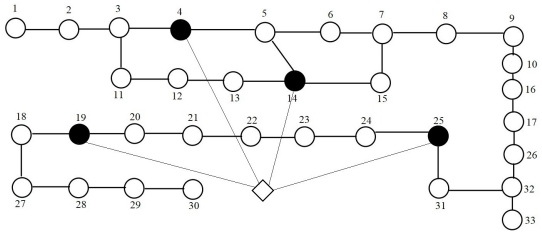

Figure 16: 2nd floor

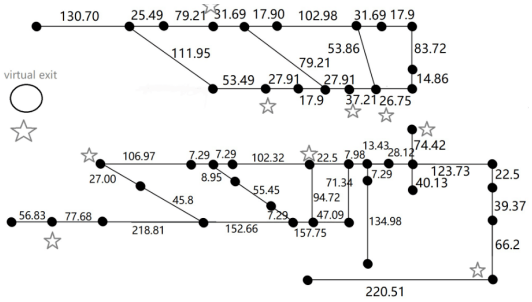

Figure 17: 1st floor

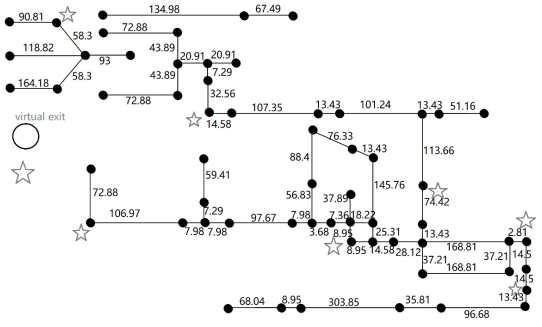

Figure 18: Ground floor

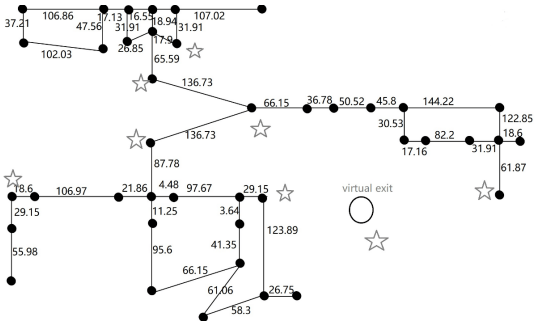

Figure 19: 1st underground floor

Figure 20: 2nd underground floor