# Assignment 2 Report

*Dezhao Chen (z5302273) and Ziqiao Ringgold Lin (z5324329)*

## Algorithm Brief

We use Hidden Markov model to solve the problem. The model can be divided into three parts. Part one is the calculation of transition matrix based on given data. Part two is the calculation of observation probability. Part three is the calculation of prediction states based on previous states together with transition matrix and observation probability.

## Step 1: Transition Matrix Generation

### Assumptions:

1. If the number of people in a room does not change, then it is likely that no one get in or get out of this room. The people who go through this room to another room are not considered since they are not reflected in the number of this room during the 15 seconds.
2. A person can move among three rooms in 15 seconds. In other words, one can go from room A to room C if B is between B and C. But he cannot get to D. (A-B-C-D). In other words, rooms within depth of 2 are connected while far apart rooms are not. Some part of the building such as room r10-r21 and c3 are tightly connected, but they tend to stick together so even these parts have many nodes connected, the graph is not very complicated.
3. When deciding the destination of a person, we use the maximum of the increased number of the neighbour rooms. For example, if the number of room A decreased 4, and its neighbours B, C and D increased 2,3 and 1 respectively, we assume people go to C first since its increase is the largest, then the remained 1 person go to B since it is the second largest.
4. According to the data file provided, we assume that during the time from 17:30 to 18:00, the overall flow of the people is different from the other time as people are more likely to leave the building and go to outside. Based on this observation, the data for transition matrix is generated with two periods, so that the first period is responsible for the time from 8:00 to 17:30, and the second period from 17:30 to 18:00.

### Calculation:

The probability of going to other rooms is calculated by:

$$P_{i,j} = \frac{Dest_{i,j}}{Sum_i}, i \neq j$$

$Dest_{i,j}$ is the number of people going from room i to room j during the time section.
$Sum_i$ is the sum of number of people throughout the whole time section.

The probability of going to other rooms is calculated by:

$$P_{i,i} = 1 - \frac{L_i}{Sum_i}$$

$L_i$ is the sum of numbers of people leaving room i during the time section.

$Sum_i$ is the sum of numbers of people throughout the whole time section.

The number change itself are calculated by comparing every 15 seconds.

## Step 2: Observation Probability

### Assumptions:

The room sensors (reliable and unreliable) only care about the room where they are placed in. The door sensors only show people passing it no matter the directions, which means there may be people in the rooms that it connects. And even if there are no people passing a door, it does not mean that there is no people in the rooms.

The robots report the number of people in the current room, which is not connected to other time.

### Calculation:

Since we only care about the probability that there are people in the room, we calculate the true positive (sensor implies there are people and there are) and false negative (sensor implies no people but there are) of the room sensors. For door sensors we calculate the true positive (sensor implies people in the two rooms and there are) and don't care about the false negative. For robots, we calculate the accuracy of the robots and find out the result of robots are always accurate which means we can directly use the result of robots as people in the certain rooms.

## Step 3: States Calculation

### Assumption:

The start state is 20 people in the Outside (coming to work).

We use a number (baseline) to decide whether there are people in a room, which is calculated by person*productivity = electricity price*(1-person). The default price and productivity are 1 and 4, so it is 4*p=1-p, and p=0.2.

### Calculation:

The state is an array of 41 length that keeps the number of people in the rooms, and it is multiplied with the transition matrix using numpy @, so that we can get the number of people.

Then we use the sensors and robots to further adjust the states. If the number of people is lower than the baseline, and the sensor shows that there are people in the room then we replace the state of the room with the tp rate of the sensor (There should be people in the room according to the sensor). If the number of people is higher than the baseline, and the sensor shows that there are no people in the room then we replace the state of the room with the fn rate of the sensor (There should be no people in the room according to the sensor). For the door sensor, we just adjust the two room when they are lower than baseline and the sensor is active. For the robots, the states are directly replaced.

When deciding the actions after the states are calculated, if the state of a room is higher than baseline, then the light is on, otherwise it is off.

## Time Consumption Analysis

### Transition Matrix Generation: *O((V + E) * N)*

The value *(V+E)* represents the memory size of the data outcome graph, where *V* is the vertexes representing all of the 41 rooms, and *E* is the size of the leaves under each vertex representing the possible outcomes of each room after people left. Since for each room, the program must iterate through the possibility of people leaving that room, the total time consumption is multiply of *N* (total data rows) with *(V+E)*. And the final result is *O((V+E) * N)*.

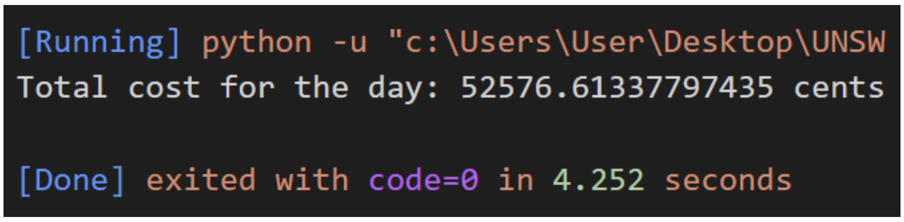
### Observation Probability: *O(M * N)*

The value *M* means the total amount of sensors used in this floor. Since the program must compute the effectiveness of each sensor with *N* rows of data provided, the total time consumption is simply *O(MN) * O(1)* where *O(1)* is the computing time consumption for each individual sensor. And the final result is *O(MN).*

### States Calculation: *O(N)*

The computing time of each state for *get_action()* call is *O(1)*. This is because the state calculation process simply looks up the value provided by the transition matrix and the reliabilities records. The value *N* represents the total rows of the testing file. And the final result for solution file execution time consumption is *O(N).*

## Final Cost Demonstration



```
[Running] python -u "c:\Users\User\Desktop\UNSW
Total cost for the day: 52576.61337797435 cents

[Done] exited with code=0 in 4.252 seconds
```

*Figure 1. Cost demo with with data.csv as the test file*

The total cost calculated with the provided original *data.csv* is about 52577 cents. Although we did not submit for early assessment of the performance of our code, and result could be a little overfitting considering the nature of this assignment, we are still satisfied with this result.