*Elevator Scheduling*
*chunmei xia*
*2019/1/13*

# REPORT

---

Firstly, I should know the elevator real steps in the building, there is some basics about elevator:

- Non-homogeneous stochastic arrival of customers
- Two types of calls: internal and external
- Has a speed and direction at any point in time
- Doors open and close
- Stationary on a floor until doors close
- Customers can abandon call

---

Secondly, we should find our exceptions in this test

- calculate the average waiting and traveling time for all humans
- calculate the average waiting and traveling time for one humans

---

Finally, I need to turn this practical problem into mathematical model. The entire process of mathematical modeling is enclosed below, Please check out the attachment for more details.

$$T_s = \begin{cases} t\,|j-i| + \beta \sum_i^j c_{ij} + \dfrac{p_s \lambda \sum_i^j c_{ij}}{\sum_1^j c_{ij}} & (1) \\ -t\,|j-i| + |T| + \beta \sum_i^j c_{ij} + \dfrac{p_s \lambda \sum_i^j c_{ij}}{\sum_j^n c_{ij}} & (2) \\ t\,|2*topInUpList - j - i| + \beta(\sum_1^i c_{ij} + \sum_1^j c_{ij}) + \dfrac{p_s \lambda (\sum_1^i c_{ij} + \sum_1^j c_{ij})}{\sum_1^j c_{ij} + \sum_1^n c_{ij}} & (3) \\ t\,|j+i-2*bottomInDownList| + \beta(\sum_i^n c_{ij} + \sum_j^n c_{ij}) + \dfrac{p_s \lambda (\sum_i^n c_{ij} + \sum_j^n c_{ij})}{\sum_1^n c_{ij} + \sum_j^n c_{ij}} & (4) \end{cases}$$

But in the program, your timer code not considered the costs for opening and closing the door, the costs when humans get in or get off. So I add a Assumption and Simplify the mathematical model.

**Assumption:**

Ignoring the cost when the humans get in and gets off, and the cost to open and close the door.

**Mathematical model:**

$$T_s = \begin{cases} |j-i| \quad (1) \\ -|j-i| + |T| \quad (2) \\ |2 * topInUpList - j - i| \quad (3) \\ |j + i - 2 * bottomInDownList| \quad (4) \end{cases}$$

$$T = topInUpList + topInDownList - bottomInDownList - bottomInDownList$$

$$(1) = \begin{cases} \text{(same down direction, i <= j)} \\ \text{(same up direction, i >= j)} \\ \text{(opposite Down direction ,topInUpList == null)} \\ \text{(opposite Up direction ,bottomInDownList == null) )} \end{cases}$$

$$(2) = \begin{cases} \text{(same down direction, i > j)} \\ \text{(same up direction, i < j)} \end{cases}$$

$(3) =$ (opposite Down direction ,topInUpList != null)
$(4) =$ (opposite Up direction ,bottomInDownList != null)

---

Depends on the mathematical model, I choose the FD-SCAN algorithm to achieve the goal. It's all my thought process and how I tackled the problem.

The initial setup of the system is:
- One human on floor 1 who wants to go to floor 4
- One human on floor 8 who wants to go to floor 5
- One human on floor 7 who wants to go to floor 9

- One shaft where the cabin is on floor 6 going down
- One shaft where the cabin is on floor 1 going up
- -One shaft where the cabin is on floor 8 going down

You can choose if launch the to keep generating random humans in the terminal, "yes" or "no"?

* How would you enhance the Threads class to be able to execute work on a specific thread after a specific delay?

```
void func(){

        Threads::GetInstance().Start();
        MessageBus::GetInstance();

        Threads::GetInstance().AddElevatorWork(std::bind(&RunElevators));
        Threads::GetInstance().AddHumanWork(std::bind(&RunHumans));

        Threads::GetInstance().Wait();

}
void SpecificThread(std::function<void()> func, unsigned int interval)
{
    std::thread([func, interval]() {
        std::this_thread::sleep_for(std::chrono::milliseconds(interval));
        func();
    }).join();
}

int main()
{

    SpecificThread(func, 6000);
    return 0;
}
```

---

* In the supplied code, the system runs forever until the process is terminated. How would you perform a clean exit?

We execute the command Ctrl+C to exit the program, will send the signal to the program, if there is resources need to be released. Could use the function below:

```
void End(int sig)
{
    if (sig == SIGINT)
    {
        //Release resources
   }
}

int main()
{

    signal(SIGINT, &End);

    return 0;
}
```

* How would you extend the system to also consider buildings containing n elevators and n humans, where buildings can be of different sizes, that is to extend the system to act as a centralised controller?

My codes could be used in the situation with the n elevators and n humans, where buildings can be of different sizes. It use the FD-SCAN, a real-time scheduling algorithm. But considering that I Ignored the cost when the humans get in and gets off, and the cost to open and close the door. If we want to extend the system to act as a centralised controller, we should change the time counter method to count the more precise costs. To improve my mathematical model, adopt the more advanced algorithms like the genetic algorithms. we can consider a multiple bits binary to represent the direction, destination floor number, source floor number, etc. And treat this multiple bits binary as a gene, by the way the mutation, crossover, inheritance function will find a optimal value to reduce the costs.

*Elevator Scheduling*
*chunmei xia*
*2019/1/13*

# ENCLOSED

The details of the elevator scheduling algorithm that I studied before coding.

---

## Basic model

### Assumption:

- The elevator velocity is uniform when it's upping or downing, ignoring the acceleration and the gravity
- Customers can not abandon call
- Ignoring the elevator capacity

### Exceptions:

- calculate the average waiting and traveling time for all humans
- calculate the average waiting and traveling time for one humans

Mathematical model

**Variable:**

- The cost that the human gets into or gets out the elevator: $T_{io} = \frac{n}{k}\sum_{i=0}^{n}(\sum_{j=0}^{k}\lambda p_j)$
- The totals of the requires and calls in one shaft: $r = \sum_{j=0}^{c_i} 1$

**Constant:**

- The floors of the building: k
- The direction of the elevator : d = up or down
- The numbers of the elevator: n(int)
- The time that one human gets into or gets out the elevator: $\lambda = 0.5s$
- open and close the elevator door at every time in one floor: $\beta = 3s$
- $C_{ij}$ ( i equals which elevator, j equal which floor)
- All the calls and requests for one shaft:

$$C_{ef} = \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1f} \\ c_{21} & c_{21} & \cdots & c_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ c_{e1} & c_{e2} & \cdots & c_{ef} \end{pmatrix}$$

$$c_{ef} = \begin{cases} 000 & \text{(no requst and call)} \\ 001 & \text{(a internal request to the top direction)} \\ 010 & \text{(a internal request to the bottom direction)} \\ 011 & \text{(one external call to the top direction same as the way the elevator runing)} \\ 100 & \text{(one external call to the top direction contrary to the elevator way runing)} \\ 101 & \text{(one external call to the bottom direction same as the way the elevator runing)} \\ 110 & \text{(one external call to the bottom direction contrary to the elevator way runing)} \end{cases}$$

**Excepted Model:**

- All the time to open and close the elevator door:$T_{oc} = \sum_{i=0}^{n} \beta r_i \quad (\beta = 3,$ Assumption A$)$
- All the Time to gets into or gets out the elevator :$T_{io} = \sum_{i=0}^{n} (\sum_{j=0}^{r_i} \lambda p_j) \quad (\lambda = 0.5,$ Constant D$)$
- The time to schedule one shaft for one call:

$$T_s = \begin{cases} t\,|j-i| + \beta \sum_i^j c_{ij} + \dfrac{p_s \lambda \sum_i^j c_{ij}}{\sum_1^j c_{ij}} & (1) \\ -t\,|j-i| + |T| + \beta \sum_i^j c_{ij} + \dfrac{p_s \lambda \sum_i^j c_{ij}}{\sum_j^n c_{ij}} & (2) \\ t\,|2*topInUpList - j - i| + \beta(\sum_1^i c_{ij} + \sum_1^j c_{ij}) + \dfrac{p_s \lambda (\sum_1^i c_{ij} + \sum_1^j c_{ij})}{\sum_1^j c_{ij} + \sum_1^n c_{ij}} & (3) \\ t\,|j + i - 2*bottomInDownList| + \beta(\sum_i^n c_{ij} + \sum_j^n c_{ij}) + \dfrac{p_s \lambda (\sum_i^n c_{ij} + \sum_j^n c_{ij})}{\sum_1^n c_{ij} + \sum_j^n c_{ij}} & (4) \end{cases}$$

$$T = topInUpList + topInDownList - bottomInDownList - bottomInDownList$$

$$(1) = \begin{cases} \text{(same down direction, i <= j)} \\ \text{(same up direction, i >= j)} \\ \text{(opposite Down direction ,topInUpList == null)} \\ \text{(opposite Up direction ,bottomInDownList == null) )} \end{cases}$$

$$(2) = \begin{cases} \text{(same down direction, i > j)} \\ \text{(same up direction, i < j)} \end{cases}$$

(3) = (opposite Down direction ,topInUpList != null)
(4) = (opposite Up direction ,bottomInDownList != null)

- $i$ equals which floor number was called
- $j$ equals the current floor number
- $p_s$ initial by the random,  it is the sum of the humans exist
- $c_{ij}$ is a binary value to show an request or an call means on the $j^{th}$ floor of the $i^{th}$ shaft

- $topInUpList$ is the highest floor number in the list of up call and request
- $bottomInUpList$ is the lowest floor number in the list of up call and request
- $topInDownList$ is the highest floor number in the list of down call and request
- $bottomInDownList$ is the lowest floor number in the list of down call and request

- $t = 1$ is the average run time between any two floors
- $\beta = 3$ is the time to open and close the shaft door
- $\lambda = 0.5$ is the cost time that every human get into or get out the elevator
- $n = 20$ initial by the random, it's the numbers of floors in the building

**Reference**

1. *http://www.columbia.edu/~cs2035/courses/ieor4405.S13/p14.pdf*
2. *http://www.gaudisite.nl/ElevatorPhysicalModelSlides.pdf*