Elevator systems - optimization

Abstract

Imagine we want to construct a building and we want to design an elevator system for it. How can we do it, so the elevator system is the most efficient one for this specific building? We run simulations of different elevator systems and different algorithms, compare them and pick the best one. This is what this program is about.

Problem

Imagine a building with an elevator system. How should the elevator system work to be the most efficient one? What is the best elevator system strategy for this concrete building?

Before we dive into these questions, we need to start with what an elevator system actually is. Elevator system consists of elevators, each having some parameters (speed, capacity, ...), actions (move up, move down, stay, open doors, ...) and strategy (SCAN, first comes first served, ...), that controls elevators. This strategy is what we would like to optimize.

Thus we want to find the best way how should an elevator system behave. Elevator system makes it's decisions based on these information:

- on what floors are requests
- what floors must each elevator visit
- where each elevator is

These are also information that every elevator system needs to have at it's disposal, otherwise it could only operate randomly. Just having these information as input is sufficient for developing some more sophisticated strategy, but there are other information, that an elevator system can have access to:

Current situation information:

- how many people each elevator has
- how many people in elevator want to go in each floor
- how many people requesting for elevator is on each floor
- how many people is in the building

Population predictions:

- how likely a request appears at some floor (can change over time)
- how likely a person from floor A would like to go to floor B (can change over time)

If elevator system has some of these (or possibly some different) information at his disposal, his strategy can be much more sophisticated and has potential to operate much better. Note, that information, that each elevator needs to have at it's disposal are of course **Current situation information**.

How likely a request appears at some floor changes over time. For example in up peak period, it is very likely that requests occur in ground floor. In down peak period, most of the requests would appear in higher floors. Similiarly, for how likely a person would like to go from floor A to a different floor B. In up peak, persons from ground floor would probably like to go to their offices, let's say floors 5 to 10 and during down peak period, they would like to go from their offices to ground floor.

What information elevator system has depends on how sophisticated it is. Some elevator systems for example have the abilitiy to know how many people is in the building, each floor, or in each elevator, because elevator users have some sort of identification card with which they request the elevator.

Some other systems have the ability to know how many people in elevator want to go in each floor, because when user calls for the elevator, he doesn't just press a button, but also configures on a display where would he like to go.

There is many different elevator systems with different information at their disposal. It is safe to say, that elevator systems with more available information have the biggest potential to be the most optimal, but we might not always have the financial needs for the best elevator systems. Sometimes, we would like to find the most optimal strategy for some not so sophisticated elevator systems.

Consequently, there is also a handful of metrics against we can measure how some strategy is succesful. Some very reasonable metrics are average waiting time for an elevator, average waiting time in an elevator, worst-case waiting time or even some average of all of these metrics combined …

So now we know what problem we want to solve. We want to find the best strategy for some elevator system or more elevator systems with different available information about the enviroment. How good a strategy is could be predefined by some metrics.

Formalization of the problem

Input

- Buildig $B = (F, E_s)$
  - $F \subseteq \mathbb{Z}$, floors
  - $E_s = (E, C)$, elevator system
  - E is set of elevators
    - $e \in E : e = (e_A, e_P)$
      - e_A are elevator's possible actions
      - e_P are elevator's parameters
  - C, current situation information that elevator system can obtain from the enviroment
- P_B, population predictions for B
- Metrics M that define how succesful strategy is

Output

- find the most optimal strategy s for given building B against given metrics M

My approach

Let there be some efficiency function $q_M : q(s, B) \rightarrow [0, 1]$, obeying given metrics M. This efficiency function takes strategy s and building B and rates it by number from 0 to 1. The bigger the number, the more s is optimal for elevator system E in B.

If we have q_M and some set of strategies $S_{strategies}$, we can very easily find

$$s_{optimal} \in S_{strategies} : q_M(S_{optimal}, B) = max(\{q_M(s, B)|\forall s \in S_{strategies}\}$$

that is the most optimal.

Obtaining some $S_{strategies}$ isn't very difficult. It can be for example a set of some well-known scheduling algorithms, such as SCAN, First comes first served, priority scheduling, round robin scheduling, . . . Set $S_{strategies}$ could also potentially contain some user defined algorithms or some algorithms developed specificaly for B (for example by some genetic algorithm,. . .).

It is much harder to obtain q_M, which is right now the only missing piece needed for solving the problem.

I want q_M to run a discrete simulation.

Discrete Simulation

Efficiency function q_M will run discrete event simulation using next-event progression paradigm.

The simulation schedules events. Events are:

1. Move elevator to the current floor
2. Move elevator to the floor above
3. Move elevator to the floor beyond
4. Board people to elevator
5. Spawn people to floors

Each event takes some time. For example, time of moving an elevator to the floor above or beyond can be calculated from elevator's speed and height of rooms, it can also be an elevator's parameter ($\in e_P$) or perhaps most conviniently, it could be predifined before the start of the simulation.

Same goes for boarding time, interval between spawning people etc . . . Each event has some time for how long it will take. This time, after picking up the event from scheduler's queue is of course added to the total time t. Time t represent total running time of the simulation.

Each of these events is then scheduled by scheduler at the appropriately calculated time. What the next event is going to be is determined by some logic, depending on what event exactly is being scheduled. If the event regards elevators, next event is determined by current situation context and by the strategy. If the event regards enviroment, such as spawning people, the decision is based on P and t.

So what is being simulated is the enviroment (where people spawn, how many, where people want to go, . . .) and elevators (where to move in each step of the simulation, boarding, . . .).

Now we have to measure how well are elevators operating under control of given strategy, in order to measure how good the strategy is. What is being measured depends heavily on what the metrics are. Neverthless, during the simulation, we need to keep track of statistical data. This could be average waiting time so far, worst waiting time so far, etc . . .

After the simulation is terminated (either if t is too big or by some other condition), quality of the strategy will be based on collected statistics.

Why this approach

Running a simulation is much easier than doing complex mathematical analysis. Also, user can see how exactly will the designed system behave in real time. This can give some important insights how to either improve strategy even further or whether to either invest in a better elevator system or be satisfied with a cheaper one (with less information about current situation context).

Remarks

I would mainly like to focus on developing flexible, maintainable, easy to read and overall clean piece of software. I think it's more important to be able to easily change some of the approaches described above in order to play around with this problem and try to understand it through this software better than try to come up with the most optimal solution right off the bat.

I would also like to add support for possibility of developing some self learning algortihms that could learn from given data (building) and try to come up with the most optimal algorithm for a given building.