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, ...) and 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 *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 building's population. How good a strategy is could be predefined by some metrics.

Formalization of the problem

Input

- Buildig $B = (E, I)$ with elevator system E that has some available information about building's population I
  - $I = (C, P)$
  - where C are current situation information and P are population predictions
- Metrics that define how succesful strategy is M

Output

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

My approach

Let there be some efficiency function $q_M : q(S, B) \to [0, 1]$, 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 forr elevator system E in B.

If we have q_M and some set of strategies S_{set}, we can very easily find $S_{optimal} \in S_{set} : q_M(S_{optimal}, B) = max(\{q_M(S, B)|\forall S \in S_{set}\}$ that is the most optimal.

Obtaining some S_{set} 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_{set} 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 to define q_M this way. Efficiency function q_M will run discrete simulation. This simulation simulates elevators and population. Elevators obey S. Population obey P. If no specific P is defined, simulation could simulate people according to some distribution (e.g. Poisson distribution, Uniform distribution,...).

TODO: Discrete events describe The discrete events The simulation runs for some reasonably long time and after it ends it calculates how well S did according to M.

Simulation

Simulation referes to discrete event simulation obeying next-event time progression paradigm (TODO: reference wiki). Simulation will start at some initial situation T_0, for example situation, where all elevators are in first floor ($f_t \in T_0, f_t(l) = 0 \ \forall l \in L \in T_0$), there are no people yet ($g_t \in T_0, g_t(f) = 0 \ \forall f \in F \in T_0$) and hence no elevator must visit some floor ($h_t(e) = \emptyset \ \forall e \in E$). One step of a simulation coresponds to transition from one situation to some other situation according to elevator system strategy function. Formally defined by induction: $T_1 = s(T_0)$ and $T_i + 1 = s(T_i)$, $i \in \mathbb{N}$.

In each step, from T_i to T_{i+1}:

- update global time t by time of step t_{s_i}, $t = t + t_{s_i}$
  - global time keeps track of for how long the simulation is going and dictates population distribution
- time of step is determined by speed of currently moved elevator(s)
  - elevators can have different speeds, so some elevators move from one floor to another in one simulation step, but others are not that fast, so they are between some two floors
- update population distribution, $p_d(t) \in (s(T_i) = T_{i+1})$
- update elevators locations, $f_t \in (s(T_i) = T_{i+1})$

- spawn requests/people, update $g_t \in (s(T_i) = T_{i+1})$.
- update what floors each elevator needs to visit, update $h_t \in (s(T_i) = T_{i+1})$.

Another step of discrete event simulation is triggered by some event. If strategy function is reasonably defined, each step should corespond to simulating an event when elevator or elevators arrive to a new floor.

In strategy function definition, there are no constraints on what situations can some situation be mapped. But in order to model a real world scenario, strategy function should be able to map situation only on situations, where elevators don't change their parameters, elevators positions are only one floor away from each other and so on ...

TODO: dodefinuj strategy function definition, aby byli dosazitelne jenom nejake sitauci - tak by potom situace odpovidali nejakemu stavovemu prostoru (graf kde hrany reprezentuji dosazitelnost)

Efficiency function

We will measure efficiency by some efficiency function $q_b \in Q_b : E_b \to \mathbb{R}$, where Q_b is set of all efficiency functions for b. If for some two elevator systems $e_{b1}, e_{b2} \in E_b \; q_b \in Q_b : q_b(e_{b1}) > q_b(e_{b2})$, we say that e_{b1} is more efficient than e_{b2} according to q_b. Depending by what metrics we consider elevator system efficient, we choose appropriate efficiency function.

Some reasonable metrics are:

- average/worst-case/median/... waiting time for elevator
- average/worst-case/median/... waiting time in elevator
- how it behaves under little bit different population distribution
- TODO others, might be a good idea to reference to current knowledge section

We can define waiting time of a person for elevator as number of situations between first button press (request) of a person on some floor and first elevator on the same floor with action board, such that the person can actually board the elevator (e.g. maximum capacity isn't surpassed).

Defining waiting time of a person in an elevator is very similiar. It is number of situations between person's boarding and getting off the elevator.

Efficiency function evaluates elevator systems by running simulations.

My approach

We take some elevator systems and evaluate them through efficiency function. Efficiency function runs several simulations on this elevator system. We choose reasonable set of efficiency functions beforehand. What efficiency functions we want to use depends on what metrics are important for us. After all elevator

systems have been evaluated, we pick the best elevator system based on requirements and collected data (e.g. pick elevator system that performs best on average for every efficiency function).

This approach has several advantages. Firstly, we can easily see how specific elevator system behaves and what decisions does it make in each situation. Secondly, we can also easily tweak input parameters and see by how much different elevator systems differ. Thridly, we are very flexible in what efficiency functions to choose and by what metrics evaluate elevator systems. And last but not least, we can very easily add on new strategies in the future and test them against already collected data.

The only disadvantage I see is that each simulation might take some nontrivial amount of time, but I don't think it should be an issue (definitely not on simple strategies, like some scheduling algorithms).

Program implementation of formal model

TODO: * make this more software oriented, math definitions above * delete some and update

Simulation

- every step of the simulation elevators can either move up, down, stay or board people (these are all the events).

Attributes

- scheduler
- global time
- current situation

Elevator

- Controlled by strategy
- Elevator in a building. There might be elevators with different parameters in the same building, hence each of a different type.
- Elevator doesn't need to have all attributes set. Some elevators aren't sophisticated enough to know how many people is on board and knows just the current weight. Some others might not even know the current weight.

Attributes

- speed
- capacity
- acceleration
- average waiting time of elevator for passengers getting on/off

- current number of people
- current weight

Actions

- up()
- down()
- stay()
- board()

Building

- Building where we want our efficient elevator system.
- number of floors
- population distribution

Population distribution

Attribues

- each floor has assigned probability - coresponds to $w_b \in p_b$
- each floor has list of probabilities to what floors person would likely want to go - corresponds to $w_f \in p_b$, each entry coresponds to $w_i \in w_f$
- population size - coresponds to $s \in p_b$

Actions

- Distribute(time)
  - assigns each floor requests/persons according to distribution

Situation

- Some attributes might be set or might not. It depends how sophisticated you want your elevator system to be. For example, if elevator system users have some sort of ID card, than each person can call an elevator by the id card and therefore the CES could be certain about the number of people in a given floor. In this scenario, situation should carry this information. But in a different scenario, where users don't have an identification, CES couldn't know how many people is actually waiting on each floor. It's only information is how many times a button is pressed (and one person can press the button how many times he likes), so in this scenario it might not make sense to remember people count.

Attributes

- list of elevators with their positions
- list of floors with people count
- list of floors with indication whether there is a request for elevator or not
- list of floors to visit for each elevator

Elevator system

Attribues

- elevators
- strategy

Evaluation function

- evaluates given strategy

Action

- Evaluate(simulation)

TODO: * too early to specify user guide * general and user simulations arent very clear * disclaimer. this section is outdated ## How to use? You can either run your own simulations based on different parameters, compare different algorithms and try to optimize it for yourself or you can use more sophisticated approach and let this program run several simulations with different algorithms and tweaked parameters to find the most optimal solution.

Parameters

These are parameters you are able to set before running the simulation: 1. number of floors in the building 1. number of elevators 1. population distribution 1. each elevator's parameters 1. CES strategy