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

Given a building b from set of all buildings B, we would like to construct an elevator system for this building $e_b \in E_b$, where E_b is set of all elevator systems for b, such as e_b is the most efficient one.

Formal model

Now let's definice what exactly is a building and an elevator system. Building is defined by number of floors and population distribution. Population distribution at a given time is:

- how likely there is a request for elevator on some floor
- what's the probability of person going from floor A to floor B
- total population size

Population distribution changes over time.

Formally, building is a tuple (n_b, p_b), where $n_b \in \mathbb{N}$ and $p_b \in P_b$, where P_b is a set of all population distributions for building b. Population distribution $p_b \in P_b$ for a building b is a function $p_b : \mathbb{N} \to (w_b, w_f, s)$, where $w_b : F \to [0, 1]$, $F = \{1, 2, ..., n_b\}$, be a probability function representing how likely a request for elevator will occur at $j - th$ floor, w_f is an n_b-tuple $(w_1, w_2, ..., w_{n_b})$, where $w_i : F \to [0, 1]$ are probability functions and $w_i(i) = 0$, for $i \in \{0, 1, ..., n_b\}$, representing probability of person wanting to go from $i - th$ floor to $j - th$ floor, where $j \in F$, and $s \in \mathbb{N}$, representing total population size. Defining population distribution this way, $p_b(t)$ represents some population distribution at time t and we can simulate change of population distribution over time (t can represent parts of day, where in the morning there is an up peak and in the afternoon there is a down peak, so the population distribution has to change).

Elevator system is defined by set of elevators and strategy. It only makes sense to define elevator system only for some specific building. Before we formally define elevator system and strategy, we must first define the following.

Elevator $e \in E$ is a tuple (a, A, P), where A is a set of possible actions (such as move up, move down, idle, board,. . .), P is a set of elevator's parameters (such as capacity, acceleration, speed, current capacity,. . .), $a \in A$ is a current action and E is a set of all possible elevators.

Situation at time $t \in \mathbb{N}$ of building b is a tuple (L, p_b, f_t, g_t), $f_t : L \to F \cup (i \in F, j \in F)$, $g_t : F \to \mathbb{N}$, $L \subseteq E$. Situation describes elevators, population distribution, in what floors or between what floors are elevators and number of requests/people on a floor at a given time. Note that some elevator's parameters can change over time, such as people count or current capacity, so it makes sense to define situation this way.

Strategy is a function $s : S \to S$, where S is set of all situations.

And finally, elevator system $e_b \in E_b$ for building b is a tuple (L, s), where L is a set of elevators of a building b and s is a strategy function.

Now we know what an elevator system is, so we can try to optimize it.

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 ($f_t \in T_0, f_t(l) = 0\ \forall l \in L \in T_0$) and there are no people yet ($g_t \in T_0, g_t(f) = 0\ \forall f \in F \in T_0$). 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}, $t = t + t_{s_i}$
- 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 $g_t \in (s(T_i) = T_{i+1})$.

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.

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 probability of request of elevator occuring (e.g. floor 1 - 0.8, floor 2 - 0.05, ... floor 6 (last) - 0)
- each floor has list of probabilities, each corresponding to what floor a person might want to get (e.g. floor 1: 2 - 0.2, 3 - 0.3 4 - 0.2 5 - 0.2 6 - 0.1)
- population size

  - how many persons can spawn in a day
  - represents total number of people using building's elevators current day

Actions

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

Situation

- Represents where (in what floors) are all the elevators and where are all the people either waiting for elevator or already in an elevator.
- Central elevator scheduler makes decisions based on the current situation.
- Every time an event happens, the current situation changes to next situation. Situations are atomic.
- 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
- list of floors with people count
- list of floors with indication whether there is a request for elevator or not

Central Elevator Scheduler

- Gives instructions to elevators based on the current situation, meaning it assigns every elevator some action (event). Central Elevator Scheduler obeys some scheduling algorithm.
- Can use different scheduling algorithms at different times (e.g. would like to use different scheduling algorithm in the morning and in the afternoon).

Attribues

- population distribution
  - it's crucial that CES has this knowledge, because thanks to this, it can decide globally and not just locally by the current situation
- situation

- strategy - algorithms to obey at given times

Evaluation function

- evaluates given strategy

TODO: * too early to specify user guide * general and user simulations arent very clear ## 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

Optimization simulations

What to optimize?

- It is quite obvious, that the more elevators and the more efficient they are the more efficient is our elevator system going to be. However, we would like to minimize the number of elevators, because each lift schaft is economicaly just a wasted space, that could have been used for something more profitable.

- The same goes for elevator parameters (speed, capacity, ...). It's reasonable to keep them bounded below some maximum parameters, to make each elevator affordable.

- Hence, it makes sense to try to optimize our elevator system not just solely on performance but also on it's cost.

- Neverthless, how good elevator system is will ultimately determined by some Evaluation function, that can use completely different evaluating principle.

General simulation

- in general simulation, you don't specify number of elevators and their qualities.

- general simulation tries to not only optimize strategy, but also optimize number of elevators and their parameters to satisfy some Evaluation function.

- general simulation just runs multiple concrete strategies with different elevator counts and qualities and compares those using some quality function.
- You can choose the Evaluation function
  - best bet would be quality function based on some price-quality ratio

Concrete simulation

- you specify concrete number of elevators and their qualities
- concrete simulation just tries to find the best possible strategy for given building and population distribution

User Simulation

- After some optimization simulation has found the best approach how to tackle given building and population distribution, it might be convenient to try to run a simulation with found optimal elevator system and see how it behaves for yourself. This is exactly what user simulation is for.
- During user simulation, you can play around and change some parameters, to see how adaptive optimal solution is and have feel for how it behaves:
  1. spawn persons to exact floors
  2. change CES strategy
  3. tweak population distribution

Future

- it would be great to not just have very simple scheduling algorithms at our disposal but also some more sophisticated techniques, like genetic algorithms, machine learning etc ...