Genetic Algorithms for Load Profile Parametrization

2021-07-26 NOAH PFLUGRADT | LEANDER KOTZUR | DETLEF STOLTEN

IEK-3: Techno-Economic Systems Analysis

Context

Context

- For a paper I needed to generate a representative neighborhood.
- That meant, among other things, generating load profiles for every household.
- To specify a household, I need to know how many people are in the household, how they are working, if they commute etc.
- For privacy reasons, no real building level information was available.
- The population was described with a mixture of local, county and country statistics.
- For example:
  - Age pyramid
  - Number of families
  - Average family size
  - Number of families with 1 child
  - …
- The statistics are frequently overlapping.

Context II

- For realistic households there are a lot of implicit and common-sense constraints
- Implicit:
  - Children can not be parents in a family
  - Children can not earn money
  - …
- Common-sense:
  - Couples are mostly close in age
  - Most families have a father and a mother
  - Couples above 60 usually don’t have babies
  - …
- Implicit constraints need to be taken into account as much as possible
- Common-sense constraints need to be mostly considered to make things “better”.

Method

Genetic Algorithm Basics

- You transform the problem into a genome.
- A genome is a bit-array
- Genomes can be combined (take some part of one genome and some part of another)
- Genomes can be mutated (flip one or more bits)
- Each generation consists of a number of genomes
- The next generation is based on the previous generation
- The best examples are preserved
- The worst ones discarded
- All preserved ones get to procreate and generate new ones by combining randomly
- Then random mutation happens
- Mutation should mostly be really low.

Approach

- Due to non-linear, overlapping constraints, implicit constraints and more a strictly mathematical model didn’t seem like fun.
- Instead the following approach was used:
  - Build an object oriented model for the town.
  - Fill it with randomized data
  - For each statistic, each implicit and common sense have an evaluation functions that each output a weighted value for how well the neighborhood fits the criteria
- Build functions to turn the OO-model into a genome (short bit-string) and back
- Use a genetic algorithm to optimize the town configuration.

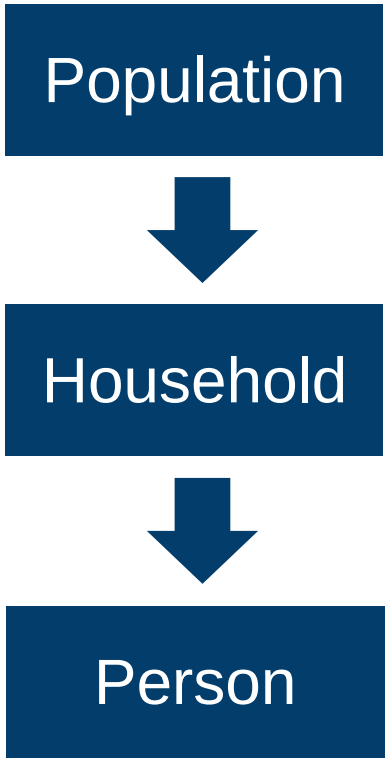
Genome

- For a genetic algorithms the genome mapping seems to be one of the most important criteria.
- In this project I coded the genome as 10-bit integer ID of the target household.
- The people were pre-created. From the age pyramid gender and age were available.
- Every person has a position on the genome.
- Every person has 10 bits on the genome

Person 1 (Age 12, male)	Person 2 (Age 30, female)	Person 3 (Age 65, female)
01010 10100	01011 11011	01001 11111

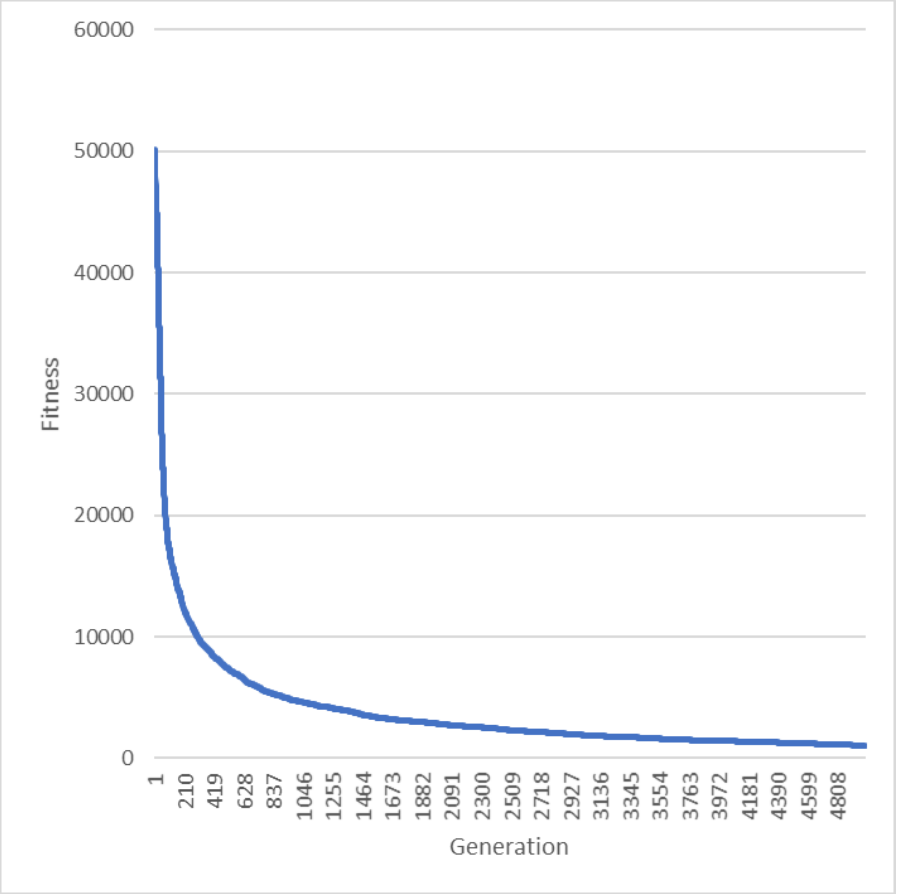
Class Structure

- Population class
  - Is able to calculate a fitness for a given genome
  - Transforms a genome into a population
  - Calls all the evaluators for the population
  - Calculates a combined fitness score
  - Contains all the households
- Household Class
  - Contains persons
- Person Class
  - Describes a person
  - Can read the household ID from the genome
- Evaluators
  - Gets a list of households
  - Generate a fitness value



Parameters

- 2000 people
- Max. 1024 households
- 20000 bit genome = 2500 bytes
- Genome Population size 500
- Top 25% preserved
- Children generated from the top 75%
- Mutate: 90% of children barely, 9% medium, 1% a lot (think radioactive accident)
- Time per generation about 0.3s
- Rapid improvements for about 500 generations, then progress slows down



Conclusion

- Easy and quick way to do complex parametrization
- Genome transformation function is critical and could probably be further improved
- Adding unlimited evaluators for different criteria is really useful, since statistics from different sources use different criteria
- Scaling seems to be limited. 10000 people didn’t yield good results anymore.

Thank you for your attention!