TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Imre Gretsi 164003IAPB

ENERGY LOAD DISAGGREGATION APPROACH BASED ON HEURISTIC OPTIMIZATION ALGORITHMS

Bachelor's thesis

Supervisor: Margarita Spitšakova
PhD

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Imre Gretsi 164003IAPB

ENERGIATARBIMISE AGREGEERITUD ANDMETE KOMPONENTIDEKS LAHUTAMINE HEURISTILISTE OPTIMEERIMISALGORITMIDEGA

Bakalaureusetöö

Juhendaja: Margarita Spitšakova
PhD

Tallinn 2019

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Imre Gretsi

20.05.2019

Abstract

Smart meters allow to collect comprehensive energy consumption data, which can be used to help energy providers in energy demand management area and to help consumers to understand and manage energy usage. Non-Intrusive Load Monitoring is an affordable alternative to smart meters and its essential part is energy load disaggregation. Purpose of energy load disaggregation consists in extraction of appliances' separate loads from total energy load. One of popular approaches for implementation of mentioned process is Genetic Algorithm usage.

The main goal of this thesis was to produce a program for reasonably precise energy load disaggregation based on Genetic Algorithm. For realization of the program were used Python, Tkinter library and Matplotlib library. Genetic algorithm part is represented by Simple Genetic Algorithm and Non-dominated Sorting Genetic Algorithm II.

The result of this work is functioning energy load disaggregation program, which allows user to choose one of four energy load disaggregation techniques and set up parameters of Genetic Algorithm. Depending on selected program's working mode it is possible to evaluate accuracy of every separate appliance's estimated load or to identify appliances' loads using total energy load's disaggregation.

This thesis is written in English and is 38 pages long, including 12 chapters, 7 figures and 0 tables.

Annotatsioon

Energiatarbimise agregeeritud andmete komponentideks lahutamine heuristiliste optimeerimisalgoritmidega

Nutikad elektriarvestid annavad võimalust saada detailseid energiatarbimise andmeid, mis võivad kasutada energia varustajad energianõudluse juhtimiseks ning võivad aidata tarbijaid aru saada ja juhtida energia kasutamist. Mitte-Pealetükkiv Energia Tarbimise Jälgimine on taskukohane asendus nutikatele elektriarvestitele ja energia tarbimise agregeeritud andmete komponentideks lahutamine on oluline osa sellest tehnoloogiast. Energia tarbimise agregeeritud andmete komponentideks lahutamise eesmärgiks on üksik seadmete energiatarbimise identifitseerimine agregeeritud andmetest. Üks populaarsetest lähenemistest on Geneetilise Algoritmi kasutamine.

Käesoleva töö põhieesmärgiks on mõistlikult täpse programmi loomine energia tarbimise agregeeritud andmete komponentideks lahutamiseks ning see lahutamine peab olema baseerutud Geneetilisel Algoritmil. Selle programmi realiseerimiseks olid kasutatud Python, Tkinter raamistik ja Matplotlib raamistik. Geneetilise Algoritmi osa on esindatud Hariliku Geneetilise Algoritmi ja Mitte-domineeriva Sorteerimise Geneetilise Algoritmiga II.

Töö tulemuseks on töötav energia tarbimise agregeeritud andmete komponentideks lahutamise programm, mis võimaldab valida üks neljast lahutamise meetoditest ning Geneetilise Algoritmi parameetreid. Sõltuvalt valitud programmi töötamise režiimist, on võimalik hinnata lahutamise resultaadis iga üksikseadme täpsust või leida iga seade energiatarbimist kasutades agregeeritud andmete lahutamist.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 38 leheküljel, 12 peatükki, 7 joonist, 0 tabelit.

List of abbreviations and terms

EA	Evolutionary Algorithms
GA	Genetic Algorithm
SGA	Simple Genetic Algorithm
NSGA-II	Non-dominated Sorting Genetic Algorithm II
MOO	Multiple Objective Optimization
TRS	Truncation Selection
RWS	Roulette Wheel Selection
LRS	Linear Rank Selection
PAS	Proposed Annealed Selection
TOS	Tournament Selection
SPC	Single-point Crossover
MPC	Multi-point Crossover
UNC	Uniform Crossover
NILM	Non-Intrusive Load Monitoring
GUI	Graphical User Interface

Table of contents

List of figures

1 Introduction

Energy consumption of humankind is increasing with every passing year. New power stations are built, and bigger amount of energy resources is used as a solution for growing energy demand. Such tendency caused problems like global warming and energy resources depletion.

For situation improvement, many different actions are taken. One of the perspective areas to work on, in the scope of this problem, is energy management. Planning of energy production and energy consumption helps energy providers to minimize energy losses and manage resources more efficiently. Precise planning requires to consider a lot of factors, like time of day, time of year, weather, etc. Among these factors, disaggregated energy consumption has an important position.

Using information about energy consumption, parted by devices, which used this energy, it is possible to estimate more accurately future energy demand, and this information also can be used for energy demand management, the main purpose of which is a shift of energy usage from peak hours to off-peak times.

For the purpose of achieving such detailed energy monitoring, energy meters should be installed separately for each appliance to observe its energy consumption. However, this energy monitoring approach demands extra hardware cost and installation complexity. Instead of the foregoing method energy load disaggregation could be used. It uses aggregated data received from a single point of measurement and disaggregates it to appliances' loads [1].

This thesis deals with creation of an energy load disaggregation application based on the genetic algorithm.

1.1 Problem overview

Energy load disaggregation is tightly connected with non-intrusive load monitoring (NILM). NILM is dealing with the analysis of changes in the voltage and current, which are going into a house, and is determining a set of appliances, which are used in this house, as well as appliances power consumption.

Energy load disaggregation can be considered as a part of the NILM. Its task consists of division of one aggregated load into set of appliances' loads. Load curve is a sum of active appliances' loads and measurement errors [2]. Hence, if appliances and their loads are known, it is possible to define the task as a search of the appliance's state at each point in time as a combinatorial optimization problem [2]. The problem is following – it is needed to choose the set of appliances' loads at each moment in time and minimize the error [2]. This problem's complexity is NP-complete, and it is similar to a knapsack problem [2], [3].

1.2 Objectives

The main goal of the thesis is to produce a program for reasonably precise energy load disaggregation based on genetic algorithm. This aim involves analysis of existing programs [4], [5], determining appliances power consumption using dataset analysis [6], searching for the viable fitness function [2], [7], implementing a genetic algorithm for energy load disaggregation as a standalone Python program and the program's work results analysis.

1.3 Outline

The thesis is organized in the following way:

1. Chapter 2 contains problem statement.
2. Chapter 3 contains introduction to Genetic Algorithm concepts.
3. Chapter 4 contains a brief overview of energy load disaggregation techniques and existing programs.
4. Chapter 5 describes the choice of tools, the choice of energy load disaggregation techniques and the way of chosen techniques validation.
5. Chapter 6 contains functional requirements for the program.
6. Chapter 7 describes architecture of the program.
7. Chapters 8-10 describe implementation of architecture layers.

8. Chapter 11 contains evaluation of used energy load disaggregation methods.

2 Problem statement

Energy load disaggregation task can be treated as knapsack problem [2]. Initial input data for this problem is power measurements from single power meter, which monitors power consumption of whole house, and average values of appliances' power consumption. Measurements' data is structured as power value for each point of time (time between measurements may vary). In [Figure 1] graphical representation of total energy load and its parts can be seen. Total load (data) is indicated by a blue line and 4 separate appliances are indicated by yellow (basement plugs and lights), green (dishwasher), red (clothes dryer), purple (heat pump) lines.

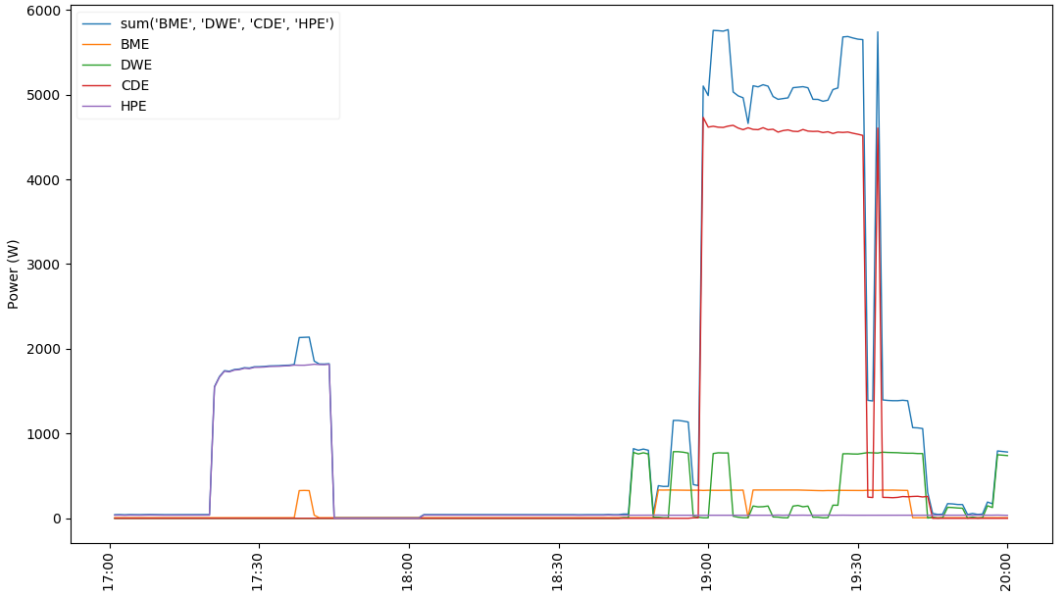

Figure 1. Energy load example.

It is possible to see that there is some measurements' error in energy load data. Appliances does not have one or more exact power values. Instead powers of appliances are standing in some ranges. Because of this noise, it is more difficult to identify, which appliances states (current power usage) are hidden in aggregated energy load.

Another problem is a low differentiation between some appliances' power consumption. For example, if there are presented two appliances with power usage of 100 W and 110 W accordingly, then it is the complicated task to distinguish one from another, due to the noise.

3 Theoretical background

3.1 Evolutionary Algorithms

Evolutionary Algorithms (EA) are using heuristic technique to achieve a solution for a problem, which cannot be solved with polynomial time [8]. For example, NP-hard problems and other time-consuming exhaustive search tasks like combinatorial problem solving [8].

EA are based on Darwin's theory of evolution. To be more precise, these algorithms adopted idea of natural selection. On the one hand, fit individuals will survive and proliferate, and, on the other hand, unfit ones will die out and their set of genes will be not inherited by further generations.

First, a population of individuals is generated. Every individual represents a possible solution for the problem and, basically, individual is a data structure. As the next step, fitness of every individual is evaluated using fitness function. Further, best individuals are selected for reproduction and less suitable individuals are killed off. Together, parents and offsprings create new generation and the process is repeated until sufficient solution is found, or number of maximum generations is achieved.

3.2 Genetic Algorithm

Genetic Algorithm (GA) is the most popular type of EA. It is the random based search algorithm, which is using mechanics of natural selection.

As in EA, GA operates with population of individuals. Every individual is a possible solution of a problem and it is represented by chromosome, which consists of genes. Hence, gene is a smallest part of the solution. For example, if task is formulated as a search of characters combination to compose a specific word, the one gene is one character and chromosome is a word.

In GA reproduction part of natural selection consists of selection, crossover and mutation operators. Due to crossover and mutation chromosomes are changing throughout generations and thanks to selection only fittest individuals will pass into next generation.

3.3 Simple Genetic Algorithm

Simple Genetic Algorithm (SGA) is the most common and is widely used. This type of genetic algorithm follows basic steps of EA: population initialization, fitness evaluation, selection of parents and creation of offsprings using recombination of parents' chromosomes [9].

SGA is represented in [9] by the next workflow:

1. Generate random population of n chromosomes.
2. Calculate the fitness of each chromosome.
3. Repeat until n offspring have been created:
    a. Select two parent chromosomes from the population.
    b. According to crossover probability:
        i. If crossover happens, then cross over the pair of parents at random point to form two offsprings;
        ii. Else form two offsprings, which are exact copies of parents
    c. Mutate each offspring with mutation probability at each gene. (If population size is odd, then one offspring can be discarded).
4. Replace the current population with offsprings.
5. Go to step 2.

3.4 Non-dominated Sorting Genetic Algorithm II

Non-dominated Sorting Genetic Algorithm II (NSGA-II) extends the genetic algorithm to be able to solve Multiple Objective Optimization (MOO) problems and is an instance of EA [10].

Processing of population in NSGA-II differs from SGA. During work of NSGA-II, population is organized into groups of non-dominated and dominated individuals. These groups are distributed using Pareto Front, which is constrained by objective functions, and Pareto optimal ideas [10], [11]. Pareto optimal is a solution, which objectives' values cannot be improved without degrading the other objectives [11]. Pareto Front consist of Pareto optimal solutions, which can be also named as non-dominated [11]. Process of population distribution into groups is named non-dominated sorting [7]. During this sorting following steps are performed [7]:

1. Individuals in the population are compared and non-dominated ones are detected.

2. Non-dominated individuals are extracted into separate set.

3. Repeat until all population is divided into sets of individuals or in other words divided into fronts.

Another difference from SGA is usage of crowding distance sorting. This sorting is used within the process of parents' collection for the reproduction. Population can be bigger than needed parents pool and because of that parents' pool is formed from higher rank fronts (if front consists of non-dominated solution its rank is 0, which is highest one, and each next front's rank is incremented by 1). If sum of chosen fronts cannot fit into parents' pool, then lowest rank front from chosen ones is sorted using crowding distance sorting and required number of individuals are extracted from this front into parents' pool [7]. This process helps to preserve diversity of individuals in the next generation [7].

NSGA-II is represented in [10] by the next workflow:

1. Generate random population.
2. Evaluate every individual against objective functions.
3. Perform non-dominated sorting.
4. Select parents.
5. Generate offsprings.
6. Do until end condition is achieved:
    a. Evaluate every individual against objective functions.
    b. Form union by merging parents and offsprings.
    c. Perform non-dominated sorting on union and get fronts set as a result.
    d. For each front do:
        i. Calculate crowding distance for individuals in the front.
        ii. If sum of parents and front is less than population size, then add front members into parents' pool.
        iii. Else sort current front by rank and crowding distance and add front members into parents' pool one by one until population size threshold for parents' pool is achieved.
    e. Select parents by rank and distance.
    f. Assign offsprings value to population.
    g. Generate offsprings.

3.5 Representation of chromosome

The classical representation of chromosome is a string of ones and zeros or, with other words, bits [9]. Each chromosome is composed of genes, for example bits, and each gene represents one allele, for example 0 or 1 [9].

Depending on the problem, chromosome representation can differ from classical one. According to a solution encoding approach, genes of chromosome can be other data types than bits and it is possible to use more than two alleles.

3.6 Genetic operators

Work of GA is based on genetics−inspired operators like selection, crossover and mutation [9]. Genetic operators are used for selection between solutions (selection), combination of existing solutions (crossover) and maintenance of diversity (mutation) [12].

3.6.1 Selection

Selection operator's purpose is to select fit individuals and allow them to pass their genes to the next generation.

Truncation Selection (TRS) is the simplest selection technique. Its usage is rare, except very large population cases [13]. TRS is represented in [13] by the next workflow:

1. Order individuals in population according to their fitness.
2. Set the portion p of individuals to select.
3. Calculate selection pressure sp by multiplying population size by p.
4. Select the first sp fittest individuals.

Roulette Wheel Selection (RWS) is a selection technique, according to which every individual has a chance to be allowed for the reproduction [13]. Chance to be selected depends on the fitness of individual [13]. The greater the fitness, the higher the chance to be selected [13]. Thanks to this, it is possible to save diversity among the population. However, there is a big risk, that dominant individual with high fitness will always win and convergence of individuals in the population will occur too early [13].

Linear Rank Selection (LRS) is a selection technique, which extend idea of RWS and tries to problem of premature convergence [13]. This technique is based on ranks. Every individual has probability to be selected based on its rank [13]. Unlike RWS, LRS is explorative technique and it prevents early convergence, because of uniform selection pressure [14].

Proposed Annealed Selection (PAS) is a selection technique, which unites properties of RWS and LRS. During GAs work selection criterion moves from exploration to exploitation [14]. To be more precise, this technique changes its behaviour according to current generation's number [14]. Depending on generation's number fitness contribution of every individual is calculated and selection pressure changes [14]. At the start fitness contribution influence on chance to be selected is bigger than at the ending of GA's work [14].

Tournament Selection (TOS) is a selection technique, which is like LRS. This technic workflow starts with random selection of k individuals from population [13]. Further these individuals are ranked by their fitness and the fittest individual is selected for reproduction [13]. This process is repeated until mating pool is filled [13].

3.6.2 Crossover

Crossover is simulating biological recombination between two organisms [9]. Hence, crossover operation is an exchange of chromosome parts between two individuals. As a result of this process two new chromosomes are obtained. If parents of these chromosomes have good fitness values, then it is likely that offsprings (new chromosomes) are even better than parents [12].

Single-point Crossover (SPC) is a crossover technique, according to which chromosome is sliced into two sets of genes at randomly chosen point and second slices of chromosomes are swapped between individuals [15]. For example, if there are two chromosomes 10001111 and 01110000, and as crossover point is chosen index 4, then two resulting chromosomes will look like 10000000 and 01111111 accordingly.

Multi-point Crossover (MPC) is a crossover technique, which has the same concept as the SPC, but following the name multiple crossover points are used. For example, if there are two chromosomes 111000101 and 000010010, and as crossover points are chosen indexes 3 and 6, then two resulting chromosomes will look like 111010101 and 000000010 accordingly.

Uniform Crossover (UNC) is a crossover technique, which is used for uniform distribution of genes between two individuals. In this technique crossover probability is used [15]. If randomly generated value is within crossover probability value, then first offspring gets first parent's gene and second offspring gets second parent's gene, otherwise first offspring gets second parent's gene and second offspring gets first parent's gene [15].

3.6.3 Mutation

Role of mutation operator in GA consists in maintenance of diversity within population and prevention of early convergence [12]. This purpose is achieved by random change in a chromosome. The classical approach of mutation is usage of mutation probability. For each gene in chromosome is generated random value and if this value is within range from 0 to mutation probability value, then gene value is changed. Gene change depends on chromosome representation and number of alleles.

4 State of the art

4.1 Energy load disaggregation methods

Since Nonintrusive Appliance Load Monitoring was introduced in Hart's original article [3], there was developed many different methods for the energy load disaggregation problem resolving. This section is aimed to provide quick overview of existing methods.

Hart's initial approach is an optimization-based task, which consists in matching of received power measurements P(t) to a possible combination of appliances' power values [1], [3]. Appliance has two states (on and off) and according to this state is decided which appliance's power value should be presented in a combination [3]. This model is presented in [3] by the following equation: $P(t) = \sum_{i=1}^{n} a_i(t)P_i + e(t)$, where n is appliances number, t is a time point, $a_i(t)$ is the appliance state (in or off) at time t and $e(t)$ is a noise (error) at time t.

Hart's method of energy load disaggregation is extended in [2] research. Base of the method remains the same, but concept of states' changes frequency is introduced [2]. According to this idea, it is more probable that appliance keeps its state the same in the next point of time [2]. Hence, number of state changes between two neighbour time points tends to be minimal [2]. Also, this research promotes idea that there is no need to analyse measurements at every point of time, only such points where occurs appliances' state changes should be used for energy load disaggregation [2].

Another extension of Hart's method is presented in [7] research. Originally Hart's method uses only active power measurements data. Idea from [7] consists in usage of active and reactive power measurements both. According to mentioned paper [7], work process includes parallel search for best combination of appliances for active and reactive power measurements at every point in time with usage of Multi-Objective Optimization.

Furthermore, there is a lot of other approaches of energy load disaggregation. These includes usage of Hidden Markov Model [16], Bayes Model [17], Neural Network [18], Pattern based Genetic Algorithm [19] and others.

4.2 Existing programs for energy load disaggregation

Themes of Non-Intrusive Load Monitoring (NILM) and energy load disaggregation are considered in many researches. There is a reasonably strong base for realisation of these technologies.

One of NILM practical implementations is described in [4]. Mentioned computer program has two work modes: sampling and evaluation. In the sampling mode program operates with data, which includes full measurements of every appliance power consumption. This data is used for appliance load recognition algorithm and, also, some of pre-processors require appliances' operating characteristics [4]. In the evaluation mode appliance load recognition algorithm analyses current, which is measured with single meter (for a whole house power consumption measurement) and uses previously identified statistics of each appliance [4]. Workflow of this program is divided into 4 separate blocks:

1. First block works only in sampling mode and calculates appliances characteristics.

2. Second block filters data from house general power meter to achieve rectangular shapes in signal, which represents on/off events.

3. Third block consists of appliance load recognition algorithm.

4. Fourth block deals with calculation of power consumption of each appliance.

Another implementation of energy load disaggregation algorithm is a NILMTK [5], which is an open source toolkit for NILM. This toolkit includes parsers for publicly available data sets (REDD, BLUED, Smart, Tracebase, Sample, HES, AMPds, iAWE, UK-DALE), some pre-processing algorithms (Downsample, Voltage normalisation, Top-k appliances) to remove noise from a signal and two reference benchmark disaggregation algorithms (based on Combinatorial Optimisation and Factorial Hidden Markov Model). This toolkit's purpose is to give ability to compare energy disaggregation approaches on multiple publicly available data sets [5].

5 Methodology

There are many different approaches for the energy load disaggregation in NILM [1]. The ones used in this thesis are chosen considering efficiency and author's skills.

5.1 Tools

As a programming language for this project Python was chosen. Python has a lot of scientific packages for data analysis and genetic algorithm libraries. Moreover, Python is a platform independent language and its interpreters are available for wide variety of operating system. In addition, there are presented some tool, which allow to package python code into standalone programs.

For data visualization Matplotlib library was chosen. Matplotlib is an easy and friendly Python 2D plotting library. Also, it is the most widely used library for plotting in Python and has is decently documented. There are another good plotting libraries for Python like Pandas and Seaborn. However, due to author's previous experience and familiarity with Matplotlib, it was preferred.

For program's Graphical User Interface (GUI) Tkinter library was used. This library is included in the standard Python distribution. Moreover, it is open source and is available under Python License. One of Tkinter advantages is that it is built-in library and because of that there is a lot of tutorials, books and its community is quite large, so it is possible to get a support for solving doubts. Another popular GUI libraries for Python are Kivy, PyQT and WxPython. Comparing to Tkinter these libraries have some disadvantages:

1. Kivy is more focused of mobile GUIs and does not support native styles.
2. PyQT has some usage limitations connected with its license.
3. WxPython does not support native styles.
4. All these libraries are not bundled with Python distribution and they are needed to be installed separately.

Many GA libraries for python can be found, but significant part of this libraries turned out to be not maintained. Some of most popular libraries are pyeasyga, Pyevolve and DEAP.

Unfortunately, pyeasyga is supported only for Python 3.4 and older, and Pyevolve is supported only for second version of Python, which will be not supported since 2020. Single remaining option is DEAP. This is powerful library for rapid prototyping and testing of ideas [20]. It allows to use such evolutionary computation techniques as genetic algorithm, genetic programming, evolution strategies, particle swarm optimization and others [20]. After consideration of DEAP it was decided that it is too complex and time consuming for learning and has a lot of unnecessary components for this thesis' topic. As a result, it was decided to create own implementation of SGA and NSGA-II.

5.2 Method of energy load disaggregation

Energy load disaggregation solution search occurs to be time consuming task, which can be solved with exhaustive search, but amount of time spent on this process appears to be not reasonable, especially in the case of continuous measurements of power consumption. Program of energy load disaggregation is decided to be based on combinatorial optimization techniques. Two types of GA are chosen for this purpose: SGA and NSGA-II. The first one can deal with simple fitness functions, which depends on neighbouring criteria, and the second one can deal with more complex and not adjacent criteria. Another way is to use neural networks; however, this option is rejected due to author's lack of knowledge in this area.

5.3 Results validation

In the energy load disaggregation program is planned to use different fitness functions for solutions search. These solutions evaluation and search methods will be compared with each other basing on their accuracy of appliance's states estimation. As a baseline for comparison Hart's model [3] will be used.

6 Functional requirements

For the energy load disaggregation program following functional requirements are established:

1. User can set up list of appliances and their average power consumption. This list will be used for energy load disaggregation, so that obtained energy load will be divided into appliances' loads. This input must adapt according to a chosen fitness function and allow to associate with appliance two or more power consumption values.

2. User can set up GA parameters and chose fitness function. According to fitness function list of GA parameters can change.

    a. In case of SGA following parameters must be presented: generations limit, population size, mating pool size, gene mutation probability, elitism proportion, fitness function.

    b. In case of NSGA-II following parameters must be presented: generations limit, population size, gene mutation probability, fitness function.

3. User can set up data source for energy load disaggregation. Next input parameters for data source must be presented: files with power consumption measurements over time (according to chosen fitness function number of input files can be 1 or 2), measurements file line index to start reading from, number of lines to read.

4. During the process of energy load disaggregation user can see changes of best fitness value within population throughout all generations.

5. User can stop process of energy load disaggregation at any time and best solution from last generation must be returned.

6. At the start of the program user can choose one of two working modes:

    a. Fitness functions' efficiency evaluation mode. In this mode user should choose data source files with full power consumption measurements

information about every appliance. After GA work is ended, user can go to GA result analysis page and see overall accuracy of disaggregated load and accuracy of every appliance's estimated load. Also, user can see graphical representation of disaggregated load on the result analysis page.

b. Appliances' energy loads estimation mode. In this mode user should choose data source files aggregated power consumption measurements information. After GA work is ended, user can go to GA result analysis page and see graphical representation of disaggregated load (overall and each appliance separately).

7. After the end of energy load disaggregation and result review, user can save disaggregated energy load to a file.

7 Architecture

Program architecture is layer based and is divided into following layers: User Interface Layer, Business Logic Layer and Services Layer. Relationship between these layers can be seen in [Figure 2].

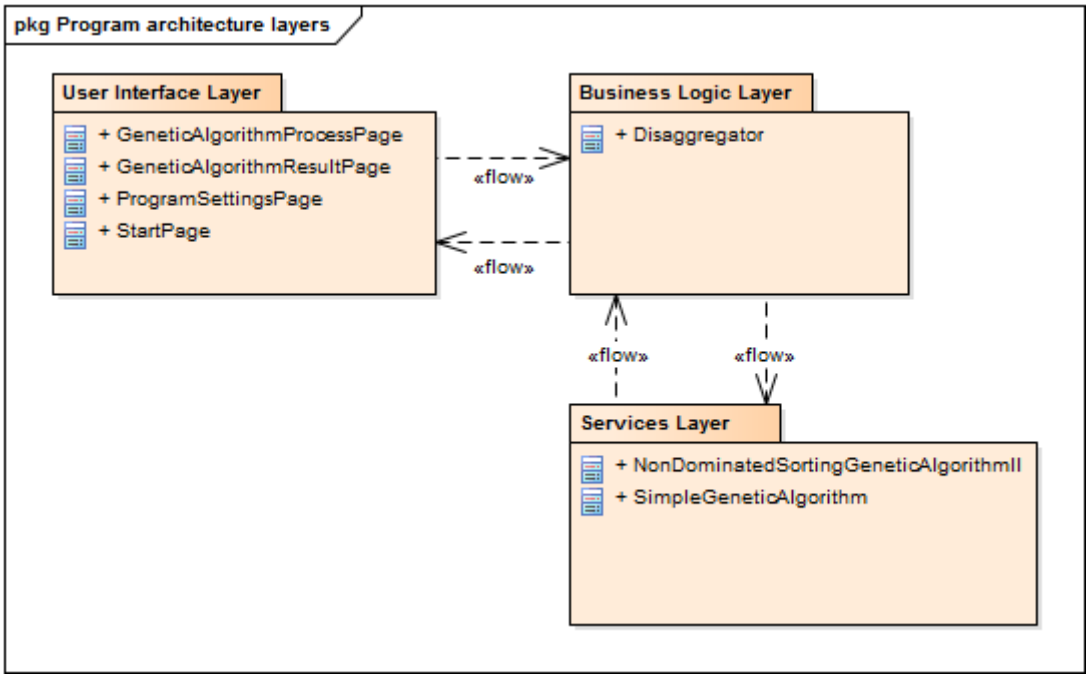

Figure 2. Program architecture layers.

User Interface Layer represents the front-end of the program and contains the actual GUI elements that users view and click. Basic elements of GUI are pages, which consists of Tkinter widgets and frames. ProgramSettingsPage is a GA settings and data source files entry point. GeneticAlgorithmProcessPage is an output point for disaggregated energy load data.

Business Logic Layer is represented by Disaggregator class, which processes settings and power measurements data. Disaggregator choses appropriate GA when data is processed and calls GA's run method to start energy load disaggregation.

Services Layer consists of two GA implementations: SGA and NSGA-II. Also, this layer includes all additional classes for GA's functioning. Basic structure of GAs' dependencies can be seen in [Figure 3]. Both SGA and NSGA-II use this structure.

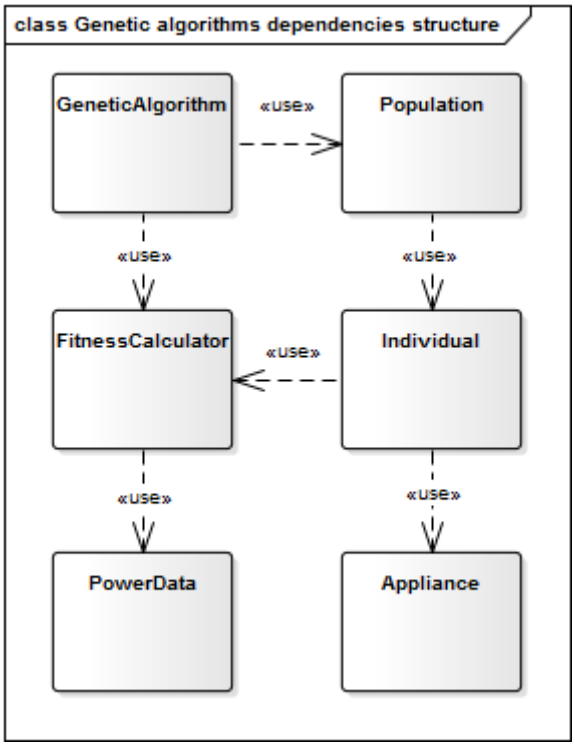

Figure 3. Genetic algorithms dependencies structure.

8 Services Layer implementation

Services Layer consists of SGA and NSGA-II. These GAs require some representation of chromosome (individual) and population. Also, some way of storing power measurements data and fitness functions is needed to be used. Realization of GAs and other essential components is presented in this section.

8.1 Representation of chromosome

Classical representation of chromosome is an array of bits. Such chromosome can be easily used for disaggregated energy load encoding. As described in [2], solution is a state of each appliance for every time point. Hence, if appliance has two states “on” and “off”, then “on” can be marked with 1 and “off” can be marked with 0. In addition, one gene is a set of appliances’ states for one time point. This gene has fixed length, because exact set of appliances must be specified for energy load disaggregation task. According to the mentioned details, chromosome suitable for “on / off” appliances’ load disaggregation can be implemented as a bit string, in which each gene stands for some time point and consists of appliances’ states. However, this chromosome representation method has limited possible states for an appliance (“on” and “off”). It is not difficult to add for each appliance more states and, in case of equal number of appliances’ states, to extend chromosome to use numbers instead of bits. For example, if appliances have three states, then state of an appliance can be encoded with numbers from zero to two. This approach works only if each appliance has the same number of states.

In the real world, almost always appliances with different number of working modes (states) are used. This fact means that it is reasonable to allow different number of states for each appliance. Therefore, it causes difficulties with gene mutation. Such approach requires to hold information about every appliance’s number of states and during mutation operation this information must be obtainable. Another problem consists in number of different power type measurements used for energy load disaggregation. Some methods use not only active power, but also reactive power measurements [7], and number of reactive power states between appliances can vary. As a result, it is a complex task to encode all mentioned information into chromosome and use it in the crossover and mutation operations. One possible solution for chromosome encoding is

usage of custom object for appliance depiction. This object can hold information about current active and reactive power values, possible states for every power type, name of appliance and timestamp of measurements. In addition to appliance's data storage this class has change_mode method, which is used for simplification of mutation operation. This implementation solves problem of additional appliance information storing and eliminates the need for appliance states information search in mutation operation.

8.2 Power measurements storage and processing

Depending on a chosen fitness function for GA, different measurements data is needed for fitness function work. There are presented three measurements data types:

1. Basic active and reactive power measurements. This data is saved directly from data source files, which are specified in program settings page.

    a. In case of "fitness functions' efficiency evaluation mode", this data is used for total load calculation, appliances' probable state changes detection and for further comparison of estimated appliances' loads and actual appliances' loads.

    b. In case of "appliances' energy loads estimation mode", this data is used for appliances' probable state changes detection and is used as total load for further disaggregation.

2. Extracted appliances' state changes events from basic active and reactive power measurements. As described in the [2] research, it is reasonable to process only appliances' state changes events instead of whole measurements data. Thanks to this approach load disaggregation process can be accelerated. Events active and reactive power measurements structure is the same as in the basic measurements data, but only events points are included. Events points search is realized using two neighbour time points comparison in total load measurements. Such points detection criterion is following: between two neighbour time points power difference must be bigger than 50 W.

a. In case of “fitness functions’ efficiency evaluation mode”, this data is each appliance’s power measurements in a certain time points (events points) and is a base for chosen appliances’ total load calculation.

b. In case of “appliances’ energy loads estimation mode”, this data is total load power measurements in a certain time points (events points).

3. Active and reactive total loads. These total loads are calculated according to chosen fitness function. Some fitness function demand whole power measurements (all time points) and others demand only events power measurements (events time points).

    a. In case of “fitness functions’ efficiency evaluation mode”, this data is calculated from data source files. This data is a sum of chosen appliances loads.

    b. In case of “appliances’ energy loads estimation mode”, this data saved directly from data source files or in case of events time point data from data source files is processed and saved.

PowerData class deals with mentioned data storing and processing tasks. Structure of first two data sets is organised as dictionary, where key is a power measurements name (usually appliance name) and value is a list of power measurements. Structure of last data set for total loads is a list of power measurements. During creation of PowerData object, according to chosen fitness function, events or basic power measurements are used for total load calculation in “fitness functions’ efficiency evaluation mode”.

Data processing and storing are based on AMPds dataset described in [6].

8.3 Fitness value calculation

FitnessCalculator class is introduced for comfortable usage of fitness functions. According to this class data storing structure, FitnessCalculator object can hold power measurements data using PowerData as a container, set of appliances for energy load disaggregation, set of active and reactive power consumption values for each appliance, name of chosen fitness function.

Three main method of FitnessCalculator are calculate_fitness, calculate_objectives, calculate_decision_function, which chose appropriate functions to calculate according to chosen fitness function name (from program settings page).

This class contains implementations of four fitness functions:

1. Fitness function described by Hart in the research [3] and mentioned in the research [2]. This is a basic fitness function, which tries to find best combination of appliances' loads and minimize error between actual total load and estimated total load using only basic active power measurements.

2. Fitness function described in the research [2]. This fitness function extends idea of Hart from the research [3] and introduces idea of minimal appliances' state changes. This function also tries to find best combination of appliances' loads and minimize error between actual total load and estimated total load, but events active power measurements are used and error from each time point is multiplied by weight, which is a number of appliances' state changes plus one.

3. Fitness function describes in the research [7]. This fitness function is similar with second one, but instead of active power error and number of appliances' state changes composition into one function, it calculates these two objectives separately using NSGA-II and further fitness is calculated using decision making function. This decision-making function implements second fitness function.

4. Fitness function describes in the research [7]. This fitness function extends idea of Hart from the research [3] and, additionally to active power, it uses reactive power measurements. Two objectives of this fitness function are estimated active and reactive total loads' errors minimization. These objectives are calculated using NSGA-II. Decision function for fitness value calculation is represented as sum of active and reactive total loads' errors and division them by two as specified in [7].

8.4 Individual

Individual class is used for chromosome and its utility information storing. Moreover, Individual object can hold FitnessCalculator to get ability to calculate chromosome's fitness. Individuals stores such information as fitness value, chromosome length, chromosome, objectives' values, dominated individuals set, number of times this individual is dominated, rank and distance. First 3 variables are used in both SGA and NSGA-II, and last 5 variables are used only for NSGA-II operations. Dominated individuals set and number of times this individual is dominated are used for fast non-dominated sorting. Rank and distance are used for crowding distance sorting.

Furthermore, Individual class provides functionality for random chromosome generation, which is used in Individual object initialization, and fitness and objective values calculation using FitnessCalculator.

8.5 Population

With purpose for individuals storing and sorting Population class is introduced. Population class holds information about population size, best and worst fitness values among population. Also, it holds list of individuals for SGA and NSGA-II, and list of offsprings for NSGA-II. Moreover, Population class provides functionality for initialization of random population, search for the fittest and the least fit individuals, fitness update (calculation) for SGA and objectives update (calculation) for NSGA-II. Significant part of Population class functionality depends on Individual class methods and variables. As a result, Population class is used to store individuals and call individual's methods for fitness and objectives update.

8.6 Simple Genetic Algorithm

SGA realization is based on materials from [9] and theory described in the "Theoretical background" section of this thesis. In addition, some changes were made to improve search characteristics of SGA.

8.6.1 Selection

PAS was chosen as a selection operator for SGA. This selection operator gradually moves selection criterion from exploration to exploitation, what allow to obtain advantages of both techniques [14]. Exploration is more prior at the start of GA process and helps to investigate new areas in the search space [14]. Exploitation is more prior at the end of GA process and deals with solution improvement using best found solutions [14]. In other words, population diversity is preferred from the start of GA and selection pressure grows closer to the end of GA to check combinations of best solutions. PAS realization is based on [14].

8.6.2 Crossover

MPC was chosen as a crossover operator for SGA. According to the crossover operator's comparison in [21], MPC is an efficient technique for chromosome recombination. In MPC realization exact number of crossover points is not determined. It dynamically depends on number of time points encoded in the chromosome. Offspring will get evenly distributed genes from parents' chromosomes. For example, if chromosome length is 7, then an offspring will get genes with indexes 0, 2, 4, 6 from parent A and genes with indexes 1, 3, 5 from parent B. Crossover rate is set to 90% and if random value is higher than crossover rate value, then offspring get genes only from one parent.

8.6.3 Mutation

Mutation operator described in [9]'s "A SIMPLE GENETIC ALGORITHM" section was used. According to this mutation operator work process, every gene in the chromosome has "mutation probability" to change. If random value is less than mutation probability, then random appliance from gene is chosen and appliances change_mode method is called. Purpose of this method is to change appliance current power values to a random chosen allowed power values.

8.6.4 Elitism

Despite the fact that SGA from [9] does not require to use elitism operator, it was decided to use such operator for a best solutions preservation throughout SGA work process. Using "elitism proportion" parameter, it is possible to specify what part of best individuals (solutions) will be transferred into next generation. This approach can

accelerate convergence of population depending on “elitism proportion” size, but, in case of bad mutations, it helps to avoid population’s fitness value decrease.

8.6.5 Workflow

Workflow of implemented and adapted SGA is following:

1. Generate random population.
2. Calculate fitness values of every individual in population.
3. Do until generations limit is achieved:
    a. Add best individuals, according to elitism proportion size, into new generation using elitism operator.
    b. Select individuals for reproduction and add them into mating pool.
    c. Do until new generation is filled:
        i. Choose two random parents from mating pool.
        ii. Create an offspring using crossover operator.
        iii. Use mutation operator on the offspring.
        iv. Add offspring into new generation.
    d. Replace current population with new generation.
    e. Calculate fitness values of each individual in population.

8.7 Non-dominated Sorting Genetic Algorithm II

NSGA-II realization is based on materials from [10], [22] and theory described in the “Theoretical background” section of this thesis. Some slight adaptation was made to make it possible to use NSGA-II with described earlier chromosome representation.

8.7.1 Selection

Selection in NSGA-II is realized in two phases. First one is a possible parents selection based on usage of fast non-dominated sorting for fronts forming and crowding distance sorting for maintenance of diversity [22]. It is possible read more detailed information in “Theoretical background” section of this thesis. Second phase is a mating pool selection, which is like TOS. This selection process is following:

1. While mating pool is not filled do:
    a. Choose two random individuals from possible parents
    b. Decide which of them is better and add this individual to mating pool.

8.7.2 Crossover

UNC was chosen as a crossover operator for NSGA-II. According to [22] crossover rate was set to 90% and crossover distribution proportion was set to 50%. If random value is less than crossover rate value, then genes from both parents will be used for offspring’s chromosome creation. Otherwise, offspring will get only one parent’s genes. Crossover distribution proportion is used to decide which parent’s gene will get offspring. If random value is less than crossover distribution proportion, then offspring gets parent’s A gene and otherwise parent’s B gene.

8.7.3 Mutation

Mutation operator described in [10]’s “Genetic Algorithm” section was used. According to this mutation operator work process, every gene in the chromosome has “mutation probability” to change. If random value is less than mutation probability, then random appliance from gene is chosen and appliances change_mode method is called. Purpose of this method is to change appliance current power values to a random chosen allowed power values.

It was decided to change only one appliance’s current power values in a gene, because according to [2] number of appliances’ state changes should be minimal.

8.7.4 Reproduction

Reproduction operator is a crossover and mutation operators' fusion. This operator uses mating pool generated with selection operator. Parents from mating pool are chosen by pairs. For example, first offspring's parents are with indexes 0 and 1, then next offspring's parents will be with indexes 1 and 2. Last offspring's parents will be with indexes "last index" and 0. Crossover and mutation operators are used, after parents are chosen. And finally, offspring will be added to the population.

8.7.5 Elitism

There is no separate elitism operator in NSGA-II. Because of that some fitness value fluctuations can be noticed. However, because fast non-dominated sorting is used and only best fitness individuals are transferred to a next generation, fitness value of population tends to improve throughout NSGA-II work process.

8.7.6 Workflow

Workflow of implemented and adapted NSGA-II is following:

1. Generate random population.
2. Evaluate every individual against objective functions.
3. Select possible parents from individuals (selection first phase).
4. Select individuals for mating pool from possible parents set (selection second phase).
5. Use reproduce operator to generate offsprings set.
6. Evaluate each offspring against objective functions.
7. Do until generations limit is achieved:
    a. Form union set by merging parents and offsprings.
    b. Select possible parents from union (selection first phase).
    c. Select individuals for mating pool (selection second phase).

d. Replace current individuals with offsprings.

e. Create new offsprings set using reproduce operator.

f. Evaluate each individual and offspring against objective functions.

9 Business Logic Layer implementation

Business Logic Layer consists of Disaggregator class. This class is a connecting link between Services Layer and User Interface Layer. Disaggregator deals with input data processing and preparation of GA, which should be used with chosen fitness function. Fitness function name, which is selected by user, has following structure: criteria for evaluation, type of measurements data, name of genetic algorithm to use.

SGA is used for “Minimum active power error (basic, SGA)” and “Minimum active power error and minimum state changes (events, SGA)” fitness function options. NSGA-II is used for “Minimum active power error and minimum state changes (events, NSGA-II)” and “Minimum active and reactive power errors (events, NSGA-II)” fitness function options.

GA preparation process is following:

1. Extraction of appliances set and appliances’ loads from appliances’ information received from GUI.

2. Extraction of following information: data source files’ paths, position of start line and amount of lines to read from data source.

3. Preparation of PowerData object for power measurements data storing. Also, PowerData internal methods are used for data conversion from basic format to events format if needed.

4. Extraction of fitness function name and GA’s parameters from genetic algorithm setting information received from GUI.

5. Preparation of FitnessCalculator object using already created PowerData object and extracted fitness function name.

6. Preparation of appropriate GA according to extracted fitness function name and, also, extracted GA’s parameters are used.

In addition to GA's data processing and preparation of GA, Disaggregator class has functionality for GA's result analysis. This functionality includes counting of every appliance's errors number and calculation of appliance estimated load accuracy. This methods of Disaggregator are used, if "fitness functions' efficiency evaluation mode" is enabled, for actual and estimated loads comparison in GeneticAlgorithmResultPage of GUI.

10 User Interface Layer implementation

Graphical User Interface consists of four views: StartPage, ProgramSettingsPage, GeneticAlgorithmProcessPage and GeneticAlgorithmResultPage. The first one is a StartPage, where user can choose which program mode to use. This choice is represented with two buttons and their descriptions. StartPage view can be seen on [Figure 4].

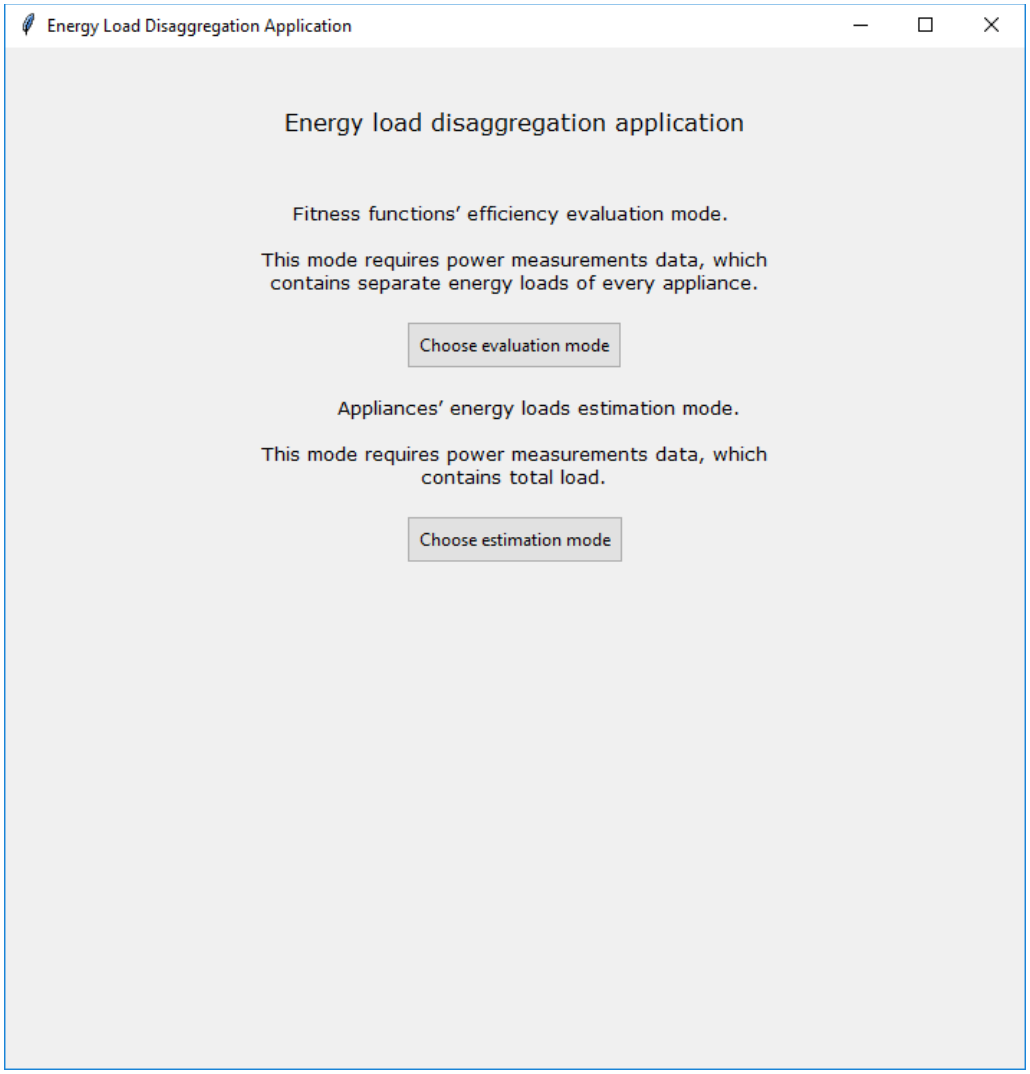

Figure 4. StartPage view.

Regardless of selected button user will be redirected to a ProgramSettingsPage. This page includes entries for genetic algorithm parameters, entries for appliances' names and average power values, entries for data source parameters, button for launch of GA's work. Whole page can be divided into two parts: genetic algorithm settings and data source settings.

According to fitness function name parameter, ProgramSettingsPage view can change. This is caused by usage of different GA implementations and, also, by usage of different power types (active and reactive power) depending on chosen fitness function. SGA requires fitness function name, generations limit, population size, mutation probability, mating pool size, elitism proportion entries as parameters. On the other hand, NSGA-II requires only first four parameters mentioned above. Moreover, if "Minimum active and reactive power errors (events, NSGA-II)" fitness function is chosen, then in data source settings part, in addition to active power entries, reactive power entries appear. Such entries include average reactive power inputs for every added appliance and reactive power data source file selector. ProgramSettingsPage view can be seen on [Figure 5].

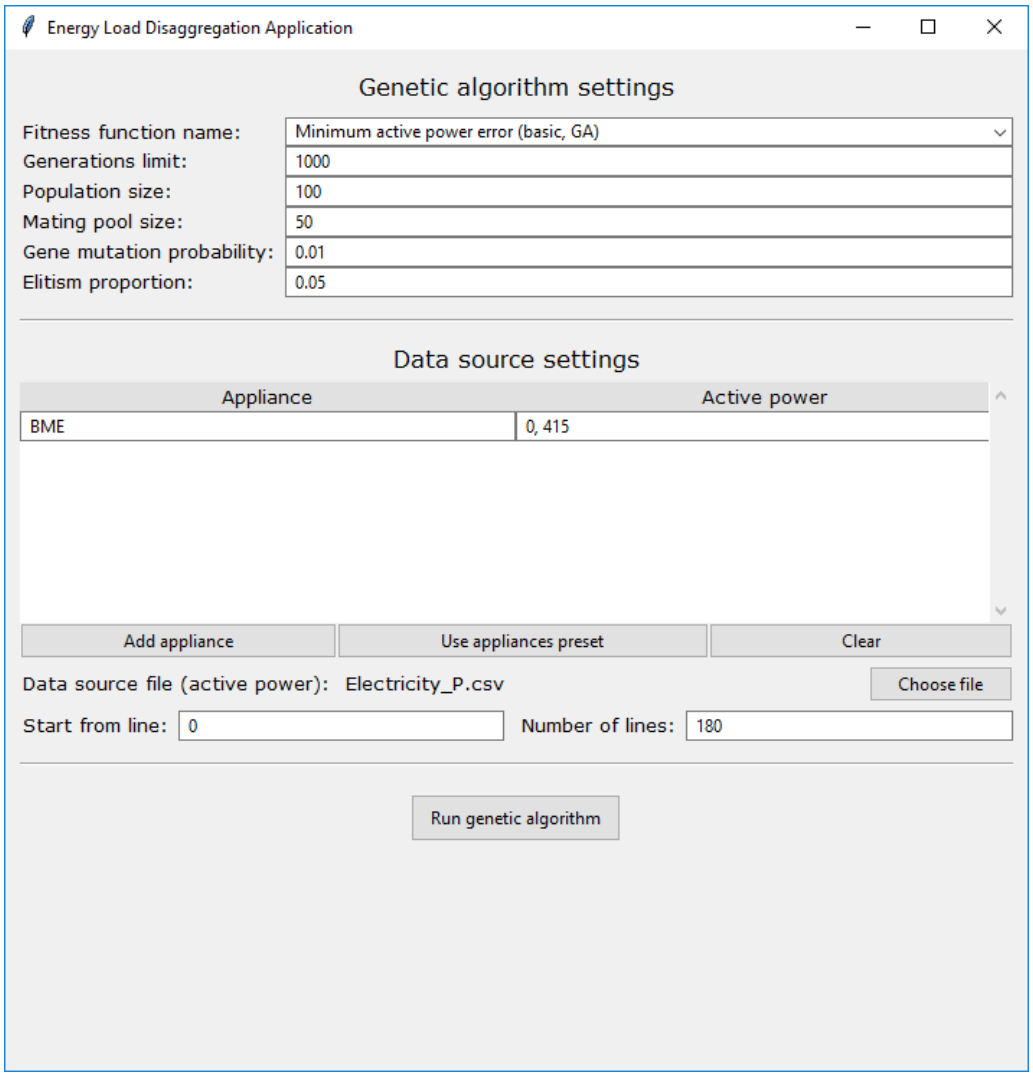

Figure 5. ProgramSettingsPage view.

After all settings are entered and GA launch button is clicked, user is redirected to GeneticAlgorithmProcessPage. This page consists of a scrollable text area and a set of

buttons. Text area is designed for depiction of GA's work process. Fitness of best individual from population and generation's number is printed at the end of each generation processing. Furthermore, chromosome of best individual is printed when GA stops work, and time elapsed from start of GA work is printed. Set of buttons includes navigation buttons to ProgramSettingsPage and to GeneticAlgorithmResultPage. Also, there is a stop button for manual halt of GA process. "Stop" button is active and navigation buttons are disabled during work of GA. Otherwise, "Stop" button is disabled and other two buttons are active. GeneticAlgorithmProcessPage can be seen on [Figure 6].

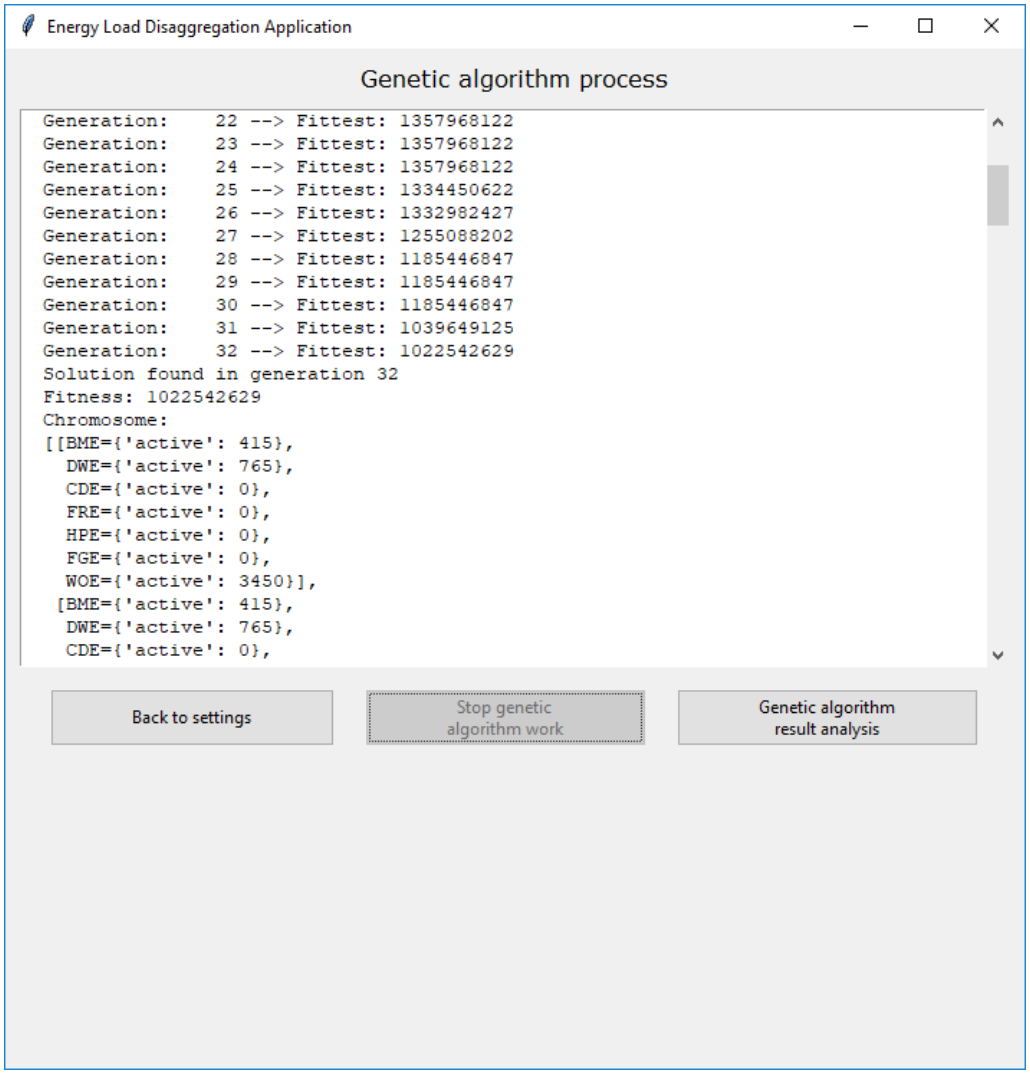

Figure 6. GeneticAlgorithmProcessPage view.

The last view of program is GeneticAlgorithmResultPage. This page consists of a notebook widget and a set of buttons. Notebook widget contains estimated total load and appliances' separate load plots. Also, if "fitness functions' efficiency evaluation

mode" is chosen, then accuracy value of estimated loads is shown above plots. Thanks to Matplotlib's functionality mentioned plots are interactive. It is possible to zoom in / out, to pan axes and to save picture of the plot. The set of buttons contains two buttons for navigation to ProgramSettingsPage and GeneticAlgorithmProcessPage, and a button for result saving into json file. Saved result is represented as dictionary, where "UNIX_TS" key indicates list of timestamps and other keys are names of appliances for lists of their estimated power values. GeneticAlgorithmResultPage can be seen on [Figure 7].

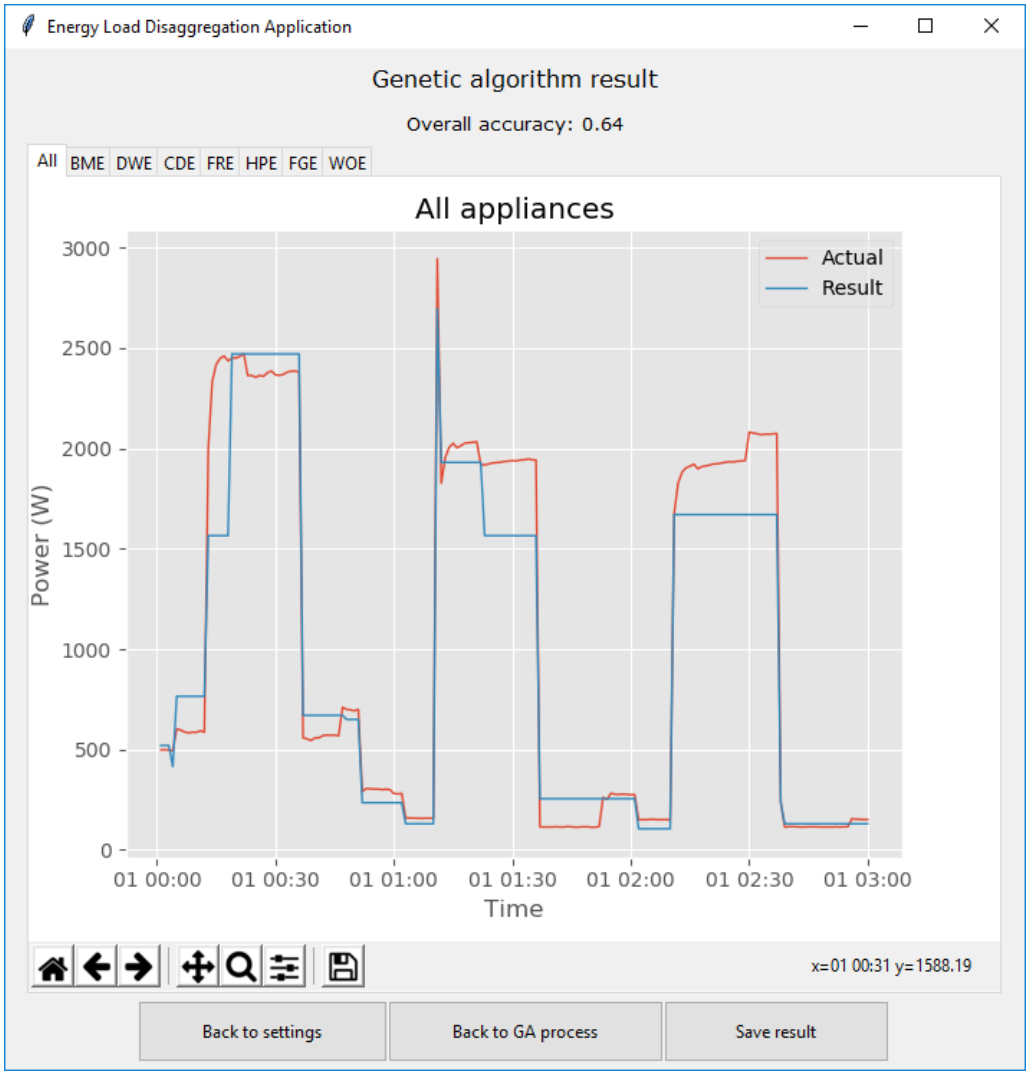

Figure 7. GeneticAlgorithmResultPage view.

11 Evaluation of fitness functions

It is possible to use four fitness functions for the energy load disaggregation. Names of these functions are following: “Minimum active power error (basic, SGA)”, “Minimum active power error and minimum state changes (events, SGA)”, “Minimum active power error and minimum state changes (events, NSGA-II)”, “Minimum active and reactive power errors (events, NSGA-II)”. First fitness function is based on Hart’s idea [3]. This function is considered as baseline for further comparison of fitness functions usage.

First three hours from AMPds [6] were chosen for testing of fitness functions. This power measurements segment is reasonably diverse, besides the fact that few appliances are working only in one state during whole time segment. However, this fact does not make energy load disaggregation task easier. The more appliances are used for energy load disaggregation, the more possible combinations should be considered and the more time for the task processing is needed. GA parameters for testing are population size - 100, generations limit - 1000, mating pool size - 50, gene mutation probability - 0.01 and elitism proportion - 0.05. Set of appliances used for testing consists of BME (basement plugs and lights), DWE (dishwasher), CDE (clothes dryer), FRE (forced air furnace fan and thermostat), HPE (heat pump), FGE (fridge), WOE (convection wall oven). Each fitness function was tested 100 times. Average work time and average accuracy were calculated for further comparison of fitness functions.

Following average results were achieved:

1. Minimum active power error (basic, SGA) - accuracy 65%, 124.557 seconds

2. Minimum active power error and minimum state changes (events, SGA) - accuracy 72%, 45.863 seconds

3. Minimum active power error and minimum state changes (events, NSGA-II) - accuracy 74%, 175.356 seconds

4. Minimum active and reactive power errors (events, NSGA-II) - accuracy 66%, 127.414 seconds

It is possible to notice that performance of first function in quite low. This function's accuracy is lowest among four fitness functions and time consumption is on the second place along with fourth function. This result was expected, because Hart's idea [3] is very similar to knapsack problem solving process. This approach focuses on minimization of error between estimated total load (sum of estimated appliances' loads) and actual total load. It is impossible to distinguish two similar appliances using this disaggregation method.

Fourth fitness function has a little bit better accuracy than first one, but its working time is longer. This function's idea is the same as in first fitness function, but in addition to active power total load disaggregation, it deals with reactive power total load disaggregation. According to research [7] this approach should improve accuracy. However, in case of thesis program, performance improvement is minimal comparing to first fitness function.

Second and third fitness function have similar accuracy results. Both these functions are based on active power total load disaggregation as in first fitness function, but additionally number of state changes between every time point of measurements is considered. These functions try to stimulate such solutions, where appliance's position in one work state is continuous, and to exclude such solution, where appliance changes its work states frequently. This approach tries to simulate real behaviour of appliances. The main difference of these fitness functions is implementation of GA, which was used. For the NSGA-II appropriate work Third fitness function is divided into two parts: calculation of objectives and calculation of fitness using objectives' values. This behaviour cause increase of processing time. However, disaggregated loads are more precise in case of third fitness function usage.

Best fitness function choice depends on user's priorities. If user needs to disaggregate energy load as soon as possible, then "Minimum active power error and minimum state changes (events, SGA)" is recommended. If user needs the most accurate result, the "Minimum active power error and minimum state changes (events, NSGA-II)" is

recommended. Additionally, it is reasonable to run energy load disaggregation several times, because GA does not guarantee to find best solution and accuracy may vary.

12 Summary

The aim of this work was to develop genetic algorithm-based program for the energy load disaggregation. Firstly, analysis of existing solutions and their functionality was conducted. Secondly, functional requirements were established and subsequently realized.

During this thesis different combinatorial optimization approaches of energy load disaggregation were researched. As a result, four methods were chosen and successfully implemented alongside with two different genetic algorithm realizations. Also, efficiency of these four methods was evaluated and analysed. Basing on analysis results two promising methods were highlighted – “estimated active power total load error minimization and reduction of appliances’ state changes” based on SGA and based on NSGA-II.

For the comfortable usage of the program graphical user interface based on Tkinter library was realized. GUI makes settings input process easier, allows to monitor process of solution search and to review result of disaggregation using plots created with Matplotlib library.

In conclusion, the main goal of this thesis has been successfully achieved. The result is the energy load disaggregation program, which can complete its task and present reasonably precise result. In addition, this program has good potential and ways for further development like addition of new and more effective fitness functions, and support for other data source formats.

References

[1] A. Zoha, A. Gluhak, M. Imran, and S. Rajasegarar, “Non-Intrusive Load Monitoring Approaches for Disaggregated Energy Sensing: A Survey,” *Sensors*, vol. 12, no. 12, pp. 16838–16866, Dec. 2012.

[2] D. Hock, M. Kappes, and B. Ghita, “Non-Intrusive Appliance Load Monitoring using Genetic Algorithms,” *IOP Conf. Ser. Mater. Sci. Eng.*, vol. 366, p. 012003, Jun. 2018.

[3] G. W. Hart, “Nonintrusive appliance load monitoring,” *Proc. IEEE*, vol. 80, no. 12, pp. 1870–1891, Dec. 1992.

[4] M. L. Marceau and R. Zmeureanu, “Nonintrusive load disaggregation computer program to estimate the energy consumption of major end uses in residential buildings,” *Energy Convers. Manag.*, vol. 41, no. 13, pp. 1389–1403, Sep. 2000.

[5] N. Batra *et al.*, “NILMTK: An open source toolkit for non-intrusive load monitoring,” presented at the e-Energy 2014 - Proceedings of the 5th ACM International Conference on Future Energy Systems, 2014.

[6] S. Makonin, B. Ellert, I. V. Bajić, and F. Popowich, “Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014,” *Sci. Data*, vol. 3, p. 160037, Jun. 2016.

[7] R. Machlev, Y. Levron, and J. Belikov, “MO-NILM: a Multi-Objective Evolutionary Algorithm for NILM Classification,” p. 8.

[8] D. Soni, “Introduction to Evolutionary Algorithms,” *Towards Data Science*, 18-Feb-2018. [Online]. Available: https://towardsdatascience.com/introduction-to-evolutionary-algorithms-a8594b484ac. [Accessed: 08-May-2019].

[9] M. Melanie, *An Introduction to Genetic Algorithms*. 1996.

[10] J. Brownlee, *Clever algorithms: nature-inspired programming recipes*, Revision 2. s.l.: LuLu.com, 2012.

[11] “Pareto Front,” *Cenaero*. [Online]. Available: http://www.cenaero.be/Page.asp?docid=27103. [Accessed: 09-May-2019].

[12] N. Dulay, “Introduction to Genetic Algorithms,” *Imperial College London*. [Online]. Available: https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol1/hmw/article1.html. [Accessed: 09-May-2019].

[13] K. Jebari, “Parent Selection Operators for Genetic Algorithms,” *Int. J. Eng. Res. Technol.*, vol. 12, pp. 1141–1145, Nov. 2013.

[14] R. K. Dcsa and I. J. Dcsa, “Effect of Annealing Selection Operators in Genetic Algorithms on Benchmark Test Functions,” *Int. J. Comput. Appl.*, vol. 40-No.3, 2012.

[15] J. Magalhães-Mendes and A. B. de Almeida, “A Comparative Study of Crossover Operators for Genetic Algorithms to Solve the Job Shop Scheduling Problem,” 2013.

[16] A. Hemmatifar, “Household Energy Disaggregation based on Difference Hidden Markov Model,” p. 5.

[17] A. Marchiori, D. Hakkarinen, Q. Han, and L. Earle, “Circuit-Level Load Monitoring for Household Energy Management,” *IEEE Pervasive Comput.*, vol. 10, no. 1, pp. 40–48, Jan. 2011.

[18] A. G. Ruzzelli, C. Nicolas, A. Schoofs, and G. M. P. O’Hare, “Real-Time Recognition and Profiling of Appliances through a Single Electricity Sensor,” in *2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2010, pp. 1–9.

[19] M. Baranski and J. Voss, “Genetic algorithm for pattern detection in NIALM systems,” in *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, 2004, vol. 4, pp. 3462–3468 vol.4.

[20] “DEAP documentation — DEAP 1.0.2 documentation.” [Online]. Available: https://deap.readthedocs.io/en/1.0.x/. [Accessed: 10-May-2019].

[21] S. P. Saroj, “Evaluation of Crossover operators performance in Genetic Algorithms,” *Int. J. Sci. Invent. Innov.*, vol. 1, no. 1, Jul. 2016.

[22] “Non-dominated Sorting Genetic Algorithm - Clever Algorithms: Nature-Inspired Programming Recipes.” [Online]. Available: http://www.cleveralgorithms.com/nature-inspired/evolution/nsga.html. [Accessed: 12-May-2019].