TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies

Sergey Korneev 193311IAIB

NILM: COMBINING NSGA-2 WITH DIFFERENT TRAINING TECHNIQUES

Bachelor's thesis

Supervisor: Juri Belikov
PhD

Co-Supervisor: Margarita Spitšakova
PhD

Tallinn 2021

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Sergey Korneev 193311IAIB

MODIFITSEERITUD NSGA-II ALGORITM ELEKTRIENERGIA DISAGREGEERIMISEKS

Bakalaurusetöö

Juhendaja: Juri Belikov
PhD

Kaasjuhendaja: Margarita Spitšakova
PhD

Tallinn 2021

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: Sergey Korneev

20.05.2021

Abstract

Modern policies that fight climate change regulate carbon dioxide emissions and make energy more expensive each year. This challenge for the society can be solved in two ways: either to generate more energy with renewables or stimulate society to consume less energy. Stimulating society to consume less energy is unpopular but an innovation like a modern technology could help with this.

Modern technologies generate large amounts of data with the potential to be analyzed by different analysis methods. One of this technology pieces is a smart electrical meter which can generate high frequency electricity consumption data. If there was an efficient and cheap technology that could show which devices are enabled or consume a lot of energy it could help people to adjust their behavior to consume less energy.

One of technologies is NILM (Nonintrusive load monitoring). This technology has been researched for the last decade with multiple techniques being proposed. One of these techniques is NSGA-II based NILM implementation. NSGA-II in NILM domain has not been well validated [1]. This thesis aims to explore different details in NSGA-II implementation, compare it with other NILM techniques and finally attempt to validate NSGA-II in NILM domain.

The thesis is in English language and contains 50 pages, 13 chapters, 19 figures, 6 tables, 3 algorithms and 5 equations.

Annotatsioon

Kaasaegne kliimamuutustega võitlev poliitika reguleerib süsinikdioksiidi heitkoguseid ja muudab energia hinda igal aastal kallimaks. Seda väljakutset ühiskonnale saab lahendada kahel viisil: kas toota taastuvate energiaallikatega rohkem energiat või stimuleerida ühiskonda vähem energiat tarbima. Ühiskonna stimuleerimine - vähema energia tarbiseks on ebapopulaarne, kuid uuenduslikkus kaasaegse tehnoloogia ees, võiks sellele kaasa aidata.

Kaasaegsed tehnoloogiad genereerivad suures koguses andmeid, mida on võimalik analüüsida erinevate analüüsimeetodite abil. Üks sellistest tehnoloogia toodetest on nutikas elektriarvesti, mis võimaldab genereerida kõrgsageduslikke elektritarbimise andmeid. Kui oleks olemas tõhus ja odav tehnoloogia, mis suudaks näidata, millised seadmed on lubatud kasutada või millised tarbivad palju energiat, võib see aidata inimestel oma käitumist muuta ehk hakkata kohandama oma energia tarbimist.

Üks sellistest tehnoloogiatest on NILM (mittetungiv koormuse jälgimine). Seda tehnoloogiat on viimase kümne aasta jooksul uuritud, pakkudes välja mitu tehnikat. Üks neist meetoditest on NSGA-II-põhine NILM-i juurutamine. NILM-i domeenis olevat NSGA-II poole kinnitatud [1]. Selle lõputöö eesmärk on uurida erinevaid detaile NSGA-II juurutamisel, võrrelda seda teiste NILM-i tehnikatega ja lõpuks proovida NSGA-II valideerida NILM-i domeenis.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 50 leheküljel, 13 peatükki, 19 joonist, 6 tabelit, 3 algoritmi ja 5 võrrandit.

List of abbreviations and terms

NILM - Non-Intrusive Load Monitoring

NILMTK - Non-Intrusive Load Monitoring toolkit

HMM - Hidden Markov model

CO - Combinational Optimization

REDD - Reference Energy Disaggregation Data Set

AMPds2 - The Almanac of Minutely Power dataset Version 2

TC - Termination Condition

DM - Decision Maker

Table of Contents

Introduction

Nonintrusive load monitoring (NILM) is a process for analyzing data generated by electric meter. It offers a cheaper way to determine an individual energy consumption of devices within a household by analyzing and disaggregating voltage and current changes in the power source cable.

The first NILM technique was patented in 1986 by George W. Hart [2]. It featured a digital AC monitor measuring both real and reactive power and an admittance measurement device whose output would be normalized and recorded. A cluster analysis then would help to determine enabled appliances behind the AC monitor.

In 2010s a large variety of open datasets [3] containing appliance consumption in apartments have appeared. These datasets remove the costly need to handle the measuring of appliance consumption and help researchers to focus solely on energy consumption disaggregation problem. The dataset only requires researcher to have a computer. No doubt this factor has helped for multiple NILM techniques to appear.

Since original patent in 1986 several new techniques have been developed and papers published. Several papers point out that NILM process implementations are feasible for determining individual energy consumptions [4].

NILM solution can be split into two parts: training of individual device consumption model and energy disaggregation.

One of the disaggregation techniques relies on an elitist multi objective genetic algorithm: NSGA-II [5]. This will be a core component of this thesis.

Although NSGA-II in NILM domain is well defined [5] little is written about how it copes with different training techniques and there are little attempts to validate it [1]. This thesis aims to integrate NSGA-II with different training techniques and see how well it performs. Two training techniques were designed during this thesis, and a training component of popular FHMM was also used.

Theis work relies on NILMTK for data collection in a python client.
The learning (except for FHMM) and energy disaggregation would be then performed on a Java server. Client and server communicate via WebSocket. Once client receives disaggregation result it will compare its accuracy and recall with other NILM technique implementations provided in NILMTK. These accuracy test result help with NSGA-II verification in NILM domain.

Problem statement

We are given two datasets: first contains only appliances individual consumption readings and the second has aggregated power consumption that was measured by the site meters to which these individual consumers are connected. These datasets are measured at different timeframes. The first goal of this thesis is to write a software capable of learning from the first dataset and then determining on or off state of these consumers given only the second dataset. This is also known as an energy consumption disaggregation problem. We prefer if the software had a scalability potential, thus it comes as a server that receives data with TCP based protocols.

Several datasets contain both load power consumption and a reactive power consumption. Despite reactive component may play a crucial role in improving an accuracy of an output, for the sake of simplicity this will be ignored.

Figure 1 illustrates how both datasets are collected. Readings of both site meter featuring a total power consumption and appliances individual consumption are stored in data storage.

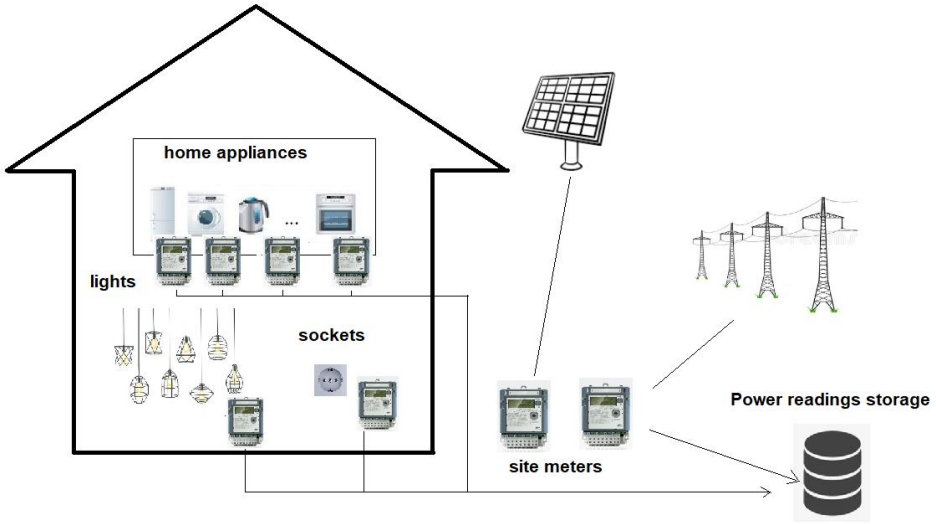

Figure 1 Energy consumption dataset collection site

Figure 2 is a graphical representation of one of the samples from REDD [6] dataset. Sum of site meters load power consumptions represent aggregated power consumption. It also includes

individual appliance power consumption. The Figure shows that a single appliance may have several discrete power consumption modes.

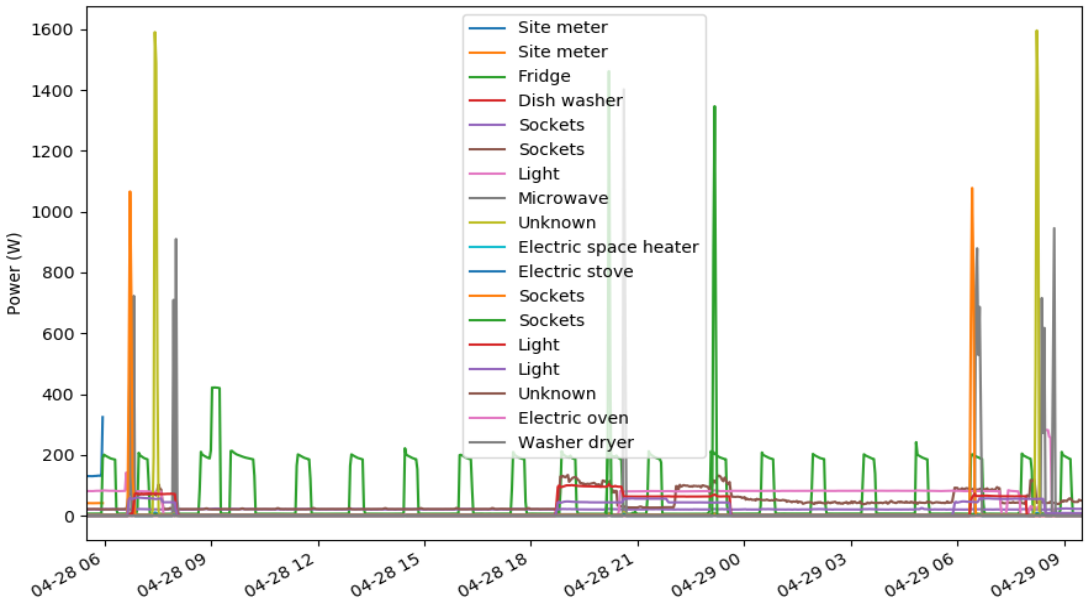

Figure 2 REDD dataset sample

NSGA-II was picked as a key algorithm for this thesis. On its own it cannot solve problems in NILM domain, it needs a proper consumption model. Second goal of this thesis is to find an algorithm that can generate this model that can be used by NSGA-II to solve NILM domain problem.

We also aim to answer these questions in the thesis:

- How well NSGA-II copes with different training techniques that generate consumption model and what is the resulting accuracy
- What are the benefits of NSGA-II when compared to disaggregation techniques from NILMTK

- How do results of disaggregation change when NSGA-II Decision maker (fitness) function changes slightly
- Are NSGA-II disaggregation accuracy and recall homogeneous between different datasets
- How easy is it to integrate with NILMTK

We plan to answer these questions by analyzing the disaggregation results based on different datasets with different properties.

NILMTK

NILMTK (Non-Intrusive Load Monitoring toolkit) is an open-source set of tools that help to organize a NILM based technique [3]. The toolkit is hosted on GitHub [7] and there have been several contributions recently in the repository. It interfaces to available open-source datasets such as REDD, AMPds2 [8] providing a single and simple API for each of them. NILMTK also provides instructions to write a custom dataset converter [9]. Both real and reactive power is supported. Moreover, it contains tools for data set diagnostics, statistics, and chart generation. It also gathers statistics about each appliance in the dataset, like minimum device off state duration, appliance type. This appliance metadata could be helpful when increasing accuracy of energy disaggregation. Several learning and disaggregation techniques are also implemented in the toolkit allowing to compare the solution of this thesis to other techniques. The toolkit uses python environment and depends on pandas and Matplotlib for data handling and visualization. The variety of tools has motivated me to create a client-server application where client mostly relies on NILMTK to load and analyze data. This toolkit will be often used and mentioned in this thesis.

Datasets

Several datasets were used in this thesis. Using different datasets helps to see a broader picture of NILM solution. Both datasets contain separate appliances with the same name. In order to distinguish these appliances, we feature appliance label and id in logs and charts.

These datasets have got different frequency of power measurements. Higher frequency allows to apply a wider range of techniques for NILM domain problem [10]. Lower frequency does not come with real benefits, but it is suitable for this thesis: it will allow to cover wider time range of measurements.

REDD

Introduced in 2011 this dataset is the first publicly and freely available dataset aimed at NILM research.
It contains detailed power usage information from several homes. The dataset [11] contains power consumption data from both site meters and individual devices. Thus, making the dataset friendly to both supervised and unsupervised NILM training. Data was collected at the frequency close to 15kHz for the whole house electricity signal. Individual circuits and plug monitors were recorder at 0.5 and 1 Hz respectively. Since AC waveform was collected, both real and reactive power is present in the dataset [6].

A low frequency sub samples with only real power consumption were used in the thesis. Individual appliance is available at 0.33 Hz. This data will be used during training phase and during accuracy measurement. The site consumption meters recordings are available at 1Hz. This data is used during disaggregation phase. REDD dataset contains two site meters for some buildings and NILMTK sees them. Thus, client will aggregate site meter data into single source during disaggregation. Dataset from building 1 is used in tests.

FHMM requires at least 64GB ram to train and disaggregate with this dataset with no limits.

A specific timeframe from REDD dataset was used during disaggregation and its length is 900 measures. Figure 3 and Figure 4 provide an overview of REDD dataset sample.

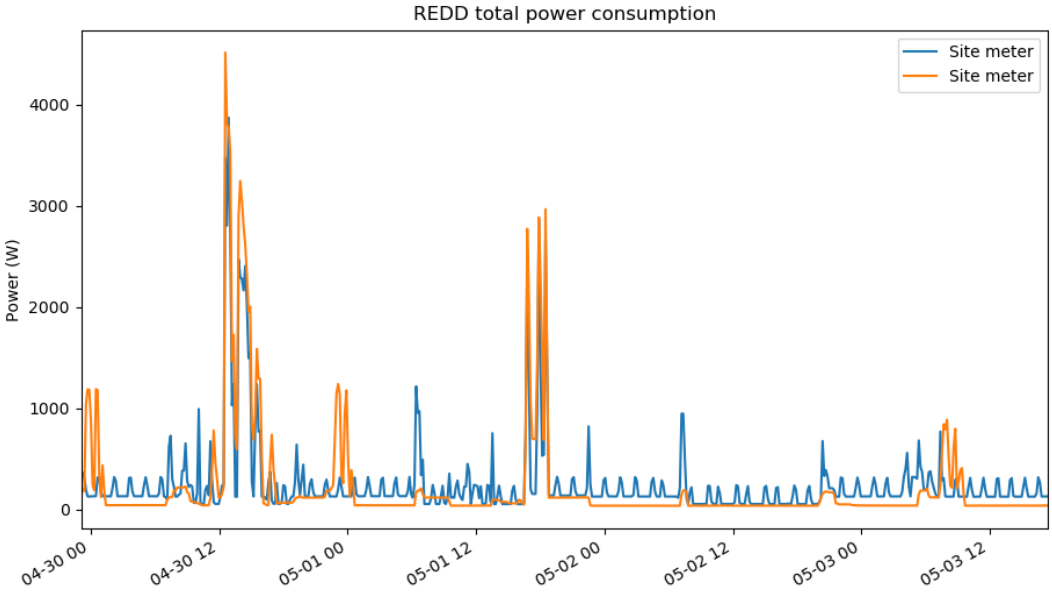

Figure 3. REDD dataset total power consumption

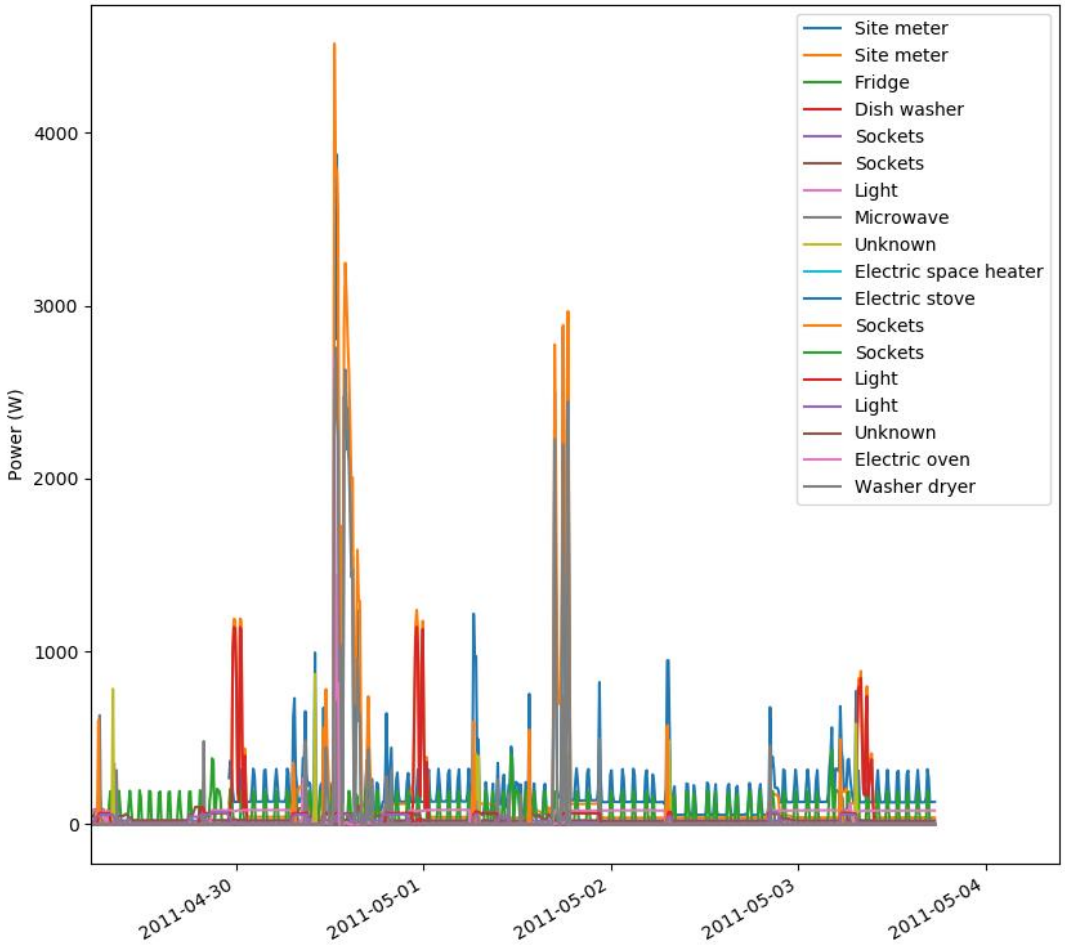

Figure 4. All appliances power consumption in REDD

AMPDs2

The Almanac of Minutely Power dataset Version 2 has been released to help computational sustainability researchers, power, and energy engineers, building scientists and technologists and other organizations. It contains data with electricity, water, and gas consumption over an extended period of two years [12]. Despite data being collected at 1Hz per site meter, electricity consumption of measures with a period of 1 minute are only available. Higher frequency is unavailable for neither NILMTK HDF5 bundle [8] nether for CVS file [13]. The 1Hz data collected here was included in RAE dataset [14]. Tests performed in this thesis mentioning AMPds2 reference 1/60 Hz measurements.

During runs with AMPDs2 with all appliance's consumption data FHMM training phase required 128GB of memory. Thus, FHMM was only provided with a limited number of appliance consumption data while testing on AMPDs.
A specific timeframe from AMPDs2 dataset was used during disaggregation and its length is 840 measures.

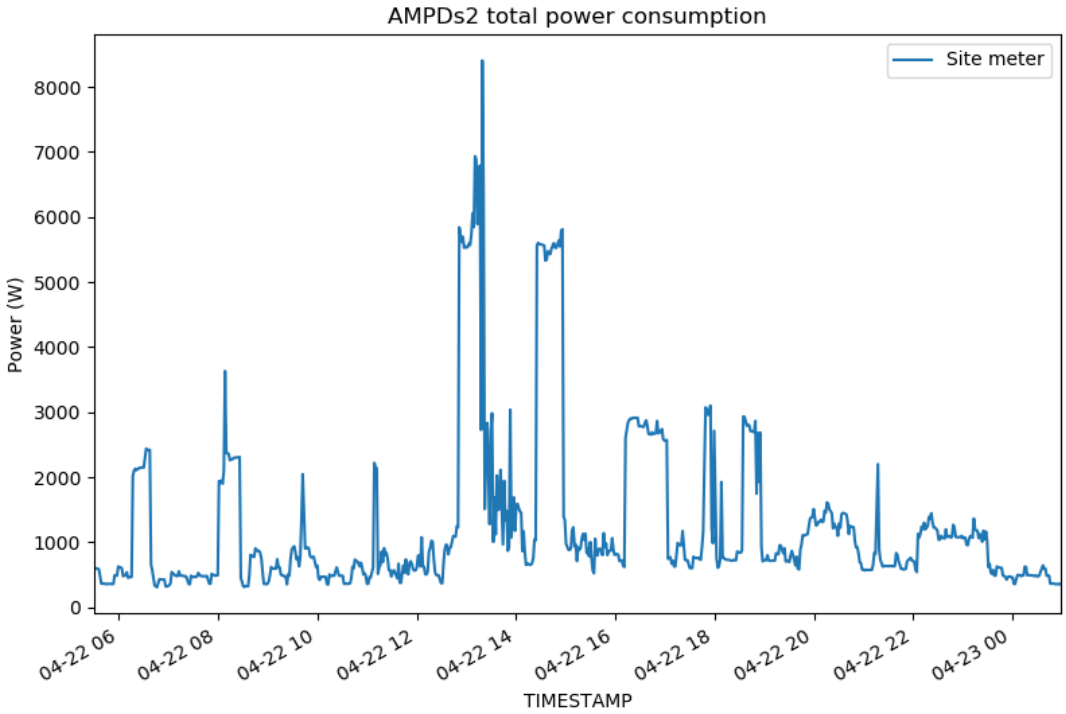

Figure 5. AMPDs2 total energy consumption

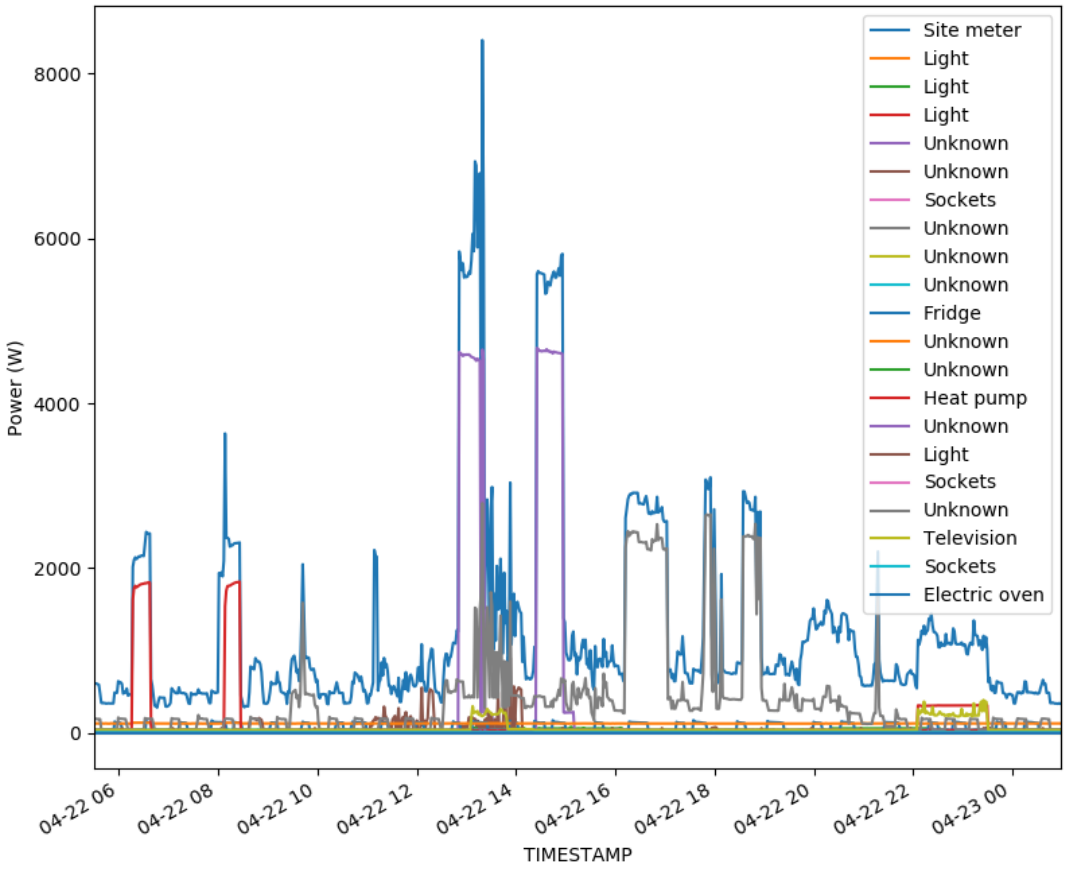

Figure 6. All appliances power consumption in AMPDs2

Overview of NILM techniques

Solving energy disaggregation consists of two subtasks: learning and disaggregation [15]. First learning phase is conducted which results in individual device consumption model. This model is then used during disaggregation phase. This phase is sometimes mentioned as prediction phase in this paper.

Overview of modern NILM training techniques

The NILM techniques first rely on learning techniques to determine a model of consumer devices. The learning technique can be either supervised or unsupervised [15].

Supervised techniques require training data to include individual appliance consumption. The process of collecting this data is expensive, time consuming and thus have scalability limitations [16]. There are different supervised techniques researched. These include Support Vector Machine (SVM), k-Nearest Neighbors, graph signal processing and HMM [15] [16]. HMM is considered to be a state-of-the-art unsupervised training algorithm. A variation of HMM, factorial HMM is implemented in NILMTK [3].

Unsupervised learning does not need individual appliance consumption data, only aggregated energy consumption is needed. Since this technique is more cost effective it has more potential in everyday usage and thus a lot of NILM research focuses on unsupervised learning. A lot of unsupervised learning techniques are a subset of HMM [15] [17]. Other techniques rely on Deep learning algorithms [15].

Overview of disaggregation techniques

Second phase of NILM solution involves applying individual device power consumption model acquired during training phase against the power output values of site meters to get a list of enabled devices. This paper will call this phase a disaggregation phase. The data that is used in the disaggregation phase will be called test data.

The simplest implementation of disaggregation phase would be a combinational optimization. It would find the best combination of appliance states where the difference between observed aggregate power and the sum of predicted appliances power consumption is minimal [3]. Although an easy implementation it becomes an overly complex computation task if the number of appliances is high.

Second solution would be to implement a set of genetic algorithms. One of the algorithms is an NSGA-II [5]. Lastly, FHMM being a full solution to NILM domain problem, also has a disaggregation phase.

Methodology

Picking training and disaggregation techniques

Amount of training techniques is abundant. Most of them are complex and focus on accuracy.

During review of training techniques, I wanted to see a simple, fast algorithm that would create an individual device consumer model. Since the datasets have power consumption data per each device supervised learning was selected. Two training algorithms were chosen. Both focus on clustering each individual appliance consumption dataset.

K-means based clustering

Firstly k-means clustering algorithm [18] was picked for training. It aims to split N electricity consumption measures for each device into K clusters. It is a complex, NP-hard algorithm [19] but simple to implement and widespread. At its core k-means will pick k random points in the dataset. Then each point in the dataset will be assigned to the nearest randomly generated point. This results in k clusters. After all points are assigned, centroid of each cluster is calculated. The centroid becomes the next point and new cluster is calculated again using the same technique. The centroids are recalculated until the list of assigned points from dataset stop switching clusters.

Number of power consumption modes per each consumer appliance in training dataset was manually analyzed by observing power consumption density charts per each device. This data then was provided to the training model. The consumed power values then would be clustered. Each cluster power consumption centroid would be then saved and used in disaggregation phase. In order to ignore rare power fluctuations or noise the solution look for 2*n clusters and ignore the less populated clusters before ending algorithm. As seen in Fig 7, k-means was directed to use 8 clusters, of which clusters 1-4 were picked. And crossed clusters are not used during disaggregation phase.
One of the problems of k-means is that sometimes a cluster is empty, and it cannot be repopulated during re-clustering. The solution is to delete this cluster since the number of clusters is larger than initial n. The result will still contain K clusters in this case.

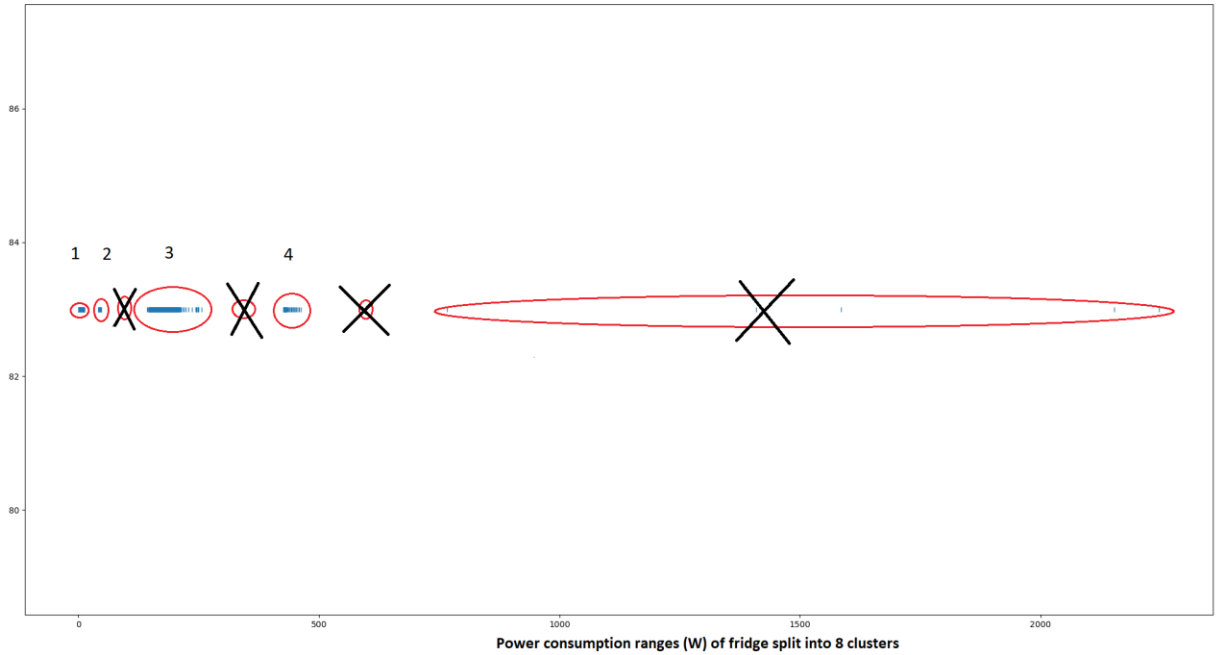

Fig 7 K-means based cluster illustration

Algorithm 1. K-means based clustering

1 K-means based clustering

Algorithm 1 clustering based on predefined cluster size

```
1: procedure FINDCLUSTERS(n, D)
2:     clusters ← generateRandomClusters()
3:     ce ← 0                          ▷ a number to track how often cluster was empty
4:     while clusterMoved = False do ▷ recluster until no elements change cluster
5:         clusterMoved ← False
6:         for d ∋ D do
7:             nearestCluster ← findNearestCluster(clusters, d)
8:             if nearestCluster ≠ previousNearestCluster(d) then
9:                 put d into nearestCluster
10:                clusterMoved ← True
11:        for c ∋ clusters do
12:            if c = ∅ then
13:                incr ce
14:                if ce ≥ 10000 then
15:                    ce ← 0
16:                    remove c from clusters      ▷ remove a cluster if it is still empty
   after several recluster cycles
17:            else
18:                recalculateCentroid(c)
19:        sort clusters by size
20:        return first n clusters
```

The accuracy of this method was not measured, but when visually compared with density chart of a single chart consumption this algorithm mostly produces adequate numbers.

E.G. Dish washer cluster average were calculated as [5.95, 227.25, 1129] and they correspond to Figure 8.

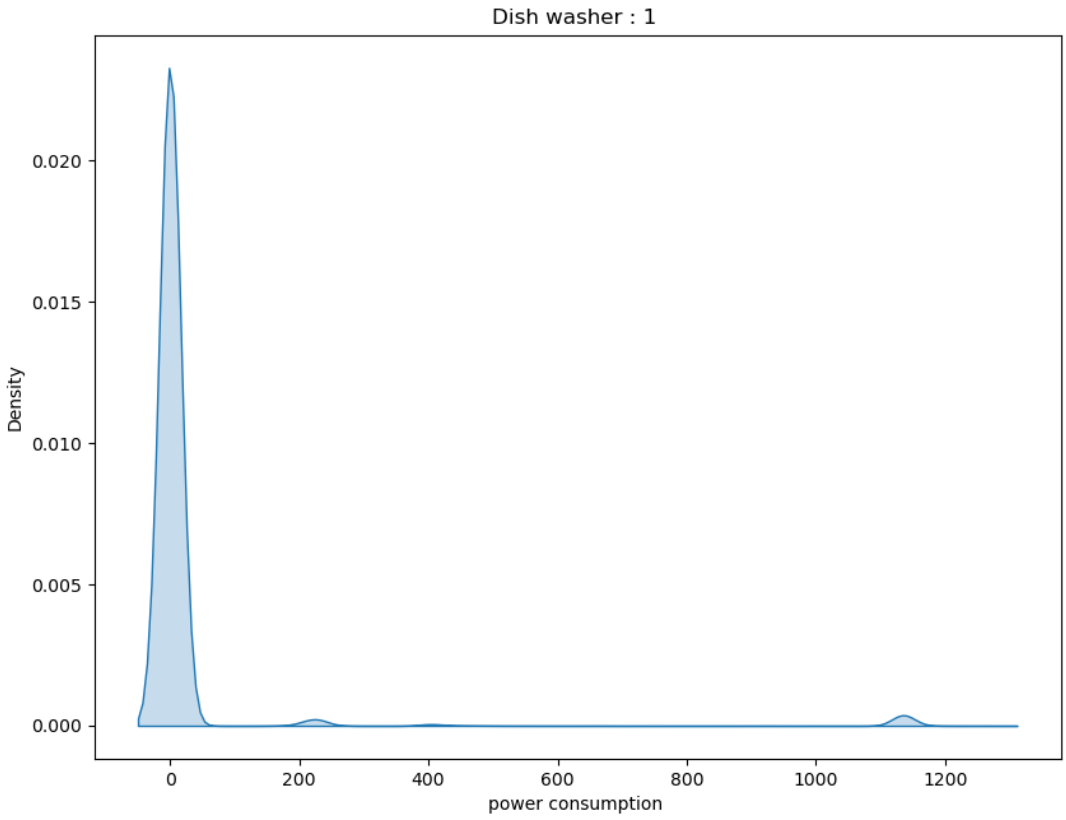

Figure 8 Dish washer power consumption density in REDD building 1

But the algorithm will struggle when there are density local maximums: it cannot detect local maximum when it has a lot of plateau properties.

E.G. Cluster average for sockets in REDD building 1 were calculated as [21.74, 44.46, 67.90] and Figure 9 is generated based on the same dataset. It is seen that the local maximum at around 100 was not identified correctly, but 67.90 was picked instead.

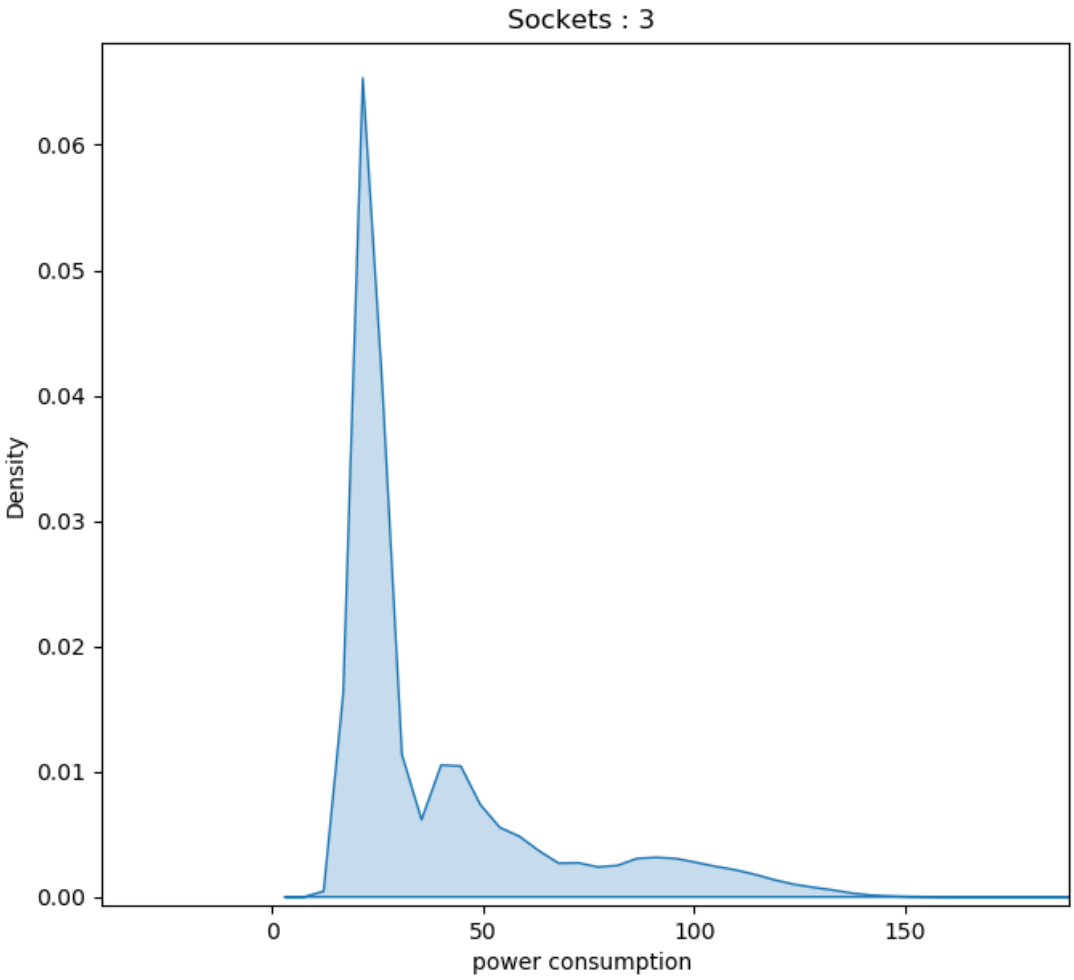

Figure 9. Socket #3 power consumption density in REDD building 1

To combat this inaccuracy, careful cluster numbers per each device was picked. This training technique but its accuracy under certain conditions and the need for human interaction means that it should be either updated or replaced by another algorithm in the future. But we still will conduct accuracy test with this training technique as it will allow to evaluate NSGA-II.

The future replacement algorithm could look for local maximums of energy consumption densities in energy datasets.

Fast clustering

Second training algorithm (Algorithm 2) is a straightforward O(n*ln(n)) algorithm where n is a size of the dataset. No academic papers were researched: this is a quick and rough attempt to cluster data. Its most complex operation is dataset sorting. There are no expectations about accuracy: it may only work with a primitive consumption condition and will only be referenced as a benchmark. This has also struggled to provide a good model and was inferior to FHMM.

Algorithm 2.Fast clustering

2 Fast clustering

Algorithm 2 Fast but inaccurate clsutrering algoritm

```
    procedure FASTFINDCLUSTERS(D)
2:      for d ∋ D do
            if d < 3 then
4:              remove d from D
        sort D ascending
6:      clusters = ∅
        currentCluster = ∅
8:      for d ∋ D do
            if diff(centroid(currentCluster), d) > 10 then
10:             currentCluster = ∅
                clusters ∪ {currentCluster}
12:         currentCluster ∪ {d}
        sort clusters by size
14:     return first 3 clusters
```

Hidden Markov Model

Last training algorithm used in this thesis is FHMM. HMM is a Markov model when states of the Markov model are unknown. Instead, only observables are known. In case of NILM, the power consumption at the site meter and its fluctuations are observables and individual appliance on/off states are hidden variables [15].
HMM is characterized by finite number of states for each appliance and finite number of observables per each state. The transition matrix would then represent a probability of moving between states of the appliance (between previous and current power consumption measure). The factorial HMM (FHMM) is an extension of the HMM with multiple independent hidden state sequences and each observation is dependent upon multiple hidden variables [15] [20]. FHMM is a preferred method for modelling time-dependent processes.

NILMTK has got an implementation of this algorithm. The generated power consumption model will be used both with NSGA-2 and FHMM disaggregation implemented in NILMTK. After

training phase NILMTK FHMM model transforms transition matrix into a dataset where each appliance power consumption modes and their power consumption can be observed. The NSGA-2 implementation then can use this dataset during disaggregation algorithm. One of the downsides of the FHMM is a monstrous appetite for memory of the transition matrix. The more appliances there are in the model the larger memory consumption. NILMTK will warn about this and suggest using fewer appliances in training set if the training fails. Though memory constrains may be negated with modern hardware and a more optimized software: during test runs 64GB of RAM was enough to run FHMM with a one of the datasets. But another dataset required 128GB RAM.

Disaggregation

NSGA-II

NSGA-2 would be used in disaggregation phase due to its simplicity and the fact that it is not that researched. Papers [21] [5] were used as a reference during implementation. Offspring generation, initial random population and NSGA-II termination conditions were not mentioned in papers; thus, following algorithms were implemented: (3) for offspring and algorithm (4) for termination condition.

Initial random population generation and offspring generation have a common feature: each product of these functions is checked for duplicates by a fast, hash table-based algorithm. If a similar individual was or about to be checked by NSGA-II for elitism, meaning its duplicate is already in the population, it will be considered a duplicate and will not be inserted into the population.

Random population generation is a simple function that generates individual with some devices enabled. It is more likely to generate and individuals with a lot of appliances enabled. This bias ensures the variety of population. After all there are more potential individual combinations with N +1 devices enabled than with N devices enabled.

Offspring generation is described by Algorithm 3. If you look at Figure 10. Small appliance dataset you will notice the dataset may contain a appliances (E.G. light, sockets and other small electronics) that have small power consumption while enabled. It is possible that during offspring generation enabling some of these devices will not drastically increase total power consumption by an individual and thus offspring still features main traits of its parents. Because of this fact it was decided that offspring generation algorithm should be able to add a lot of traits (enabled devices) with a small probability.

3 Generate offspring

Algorithm 3 generate offspring given two parents and a set of all devices in the model

```
    procedure GENERATEOFFSPRING(p1, p2, D)
2:      DS = ∅
        DS ∪ {getDeviceStates(p1)}
4:      DS ∪ {getDeviceStates(p2)}
        for ds ∋ DS do
6:          if DS.hasAnotherDevice(ds) then
                if rand50%() then
8:                  remove d from DS
                else
10:                 remove other parent d from SD
        if rand50%() then
12:         if rand10%() then
                remove random device from DS
14:         l = |D| − |DS|
            if l > 0 AND rand50%() then
16:             for random 0 to l times do
                    rd = pickup random device mode not in DS
18:                 DS ∪ {rd}
        return DS
```

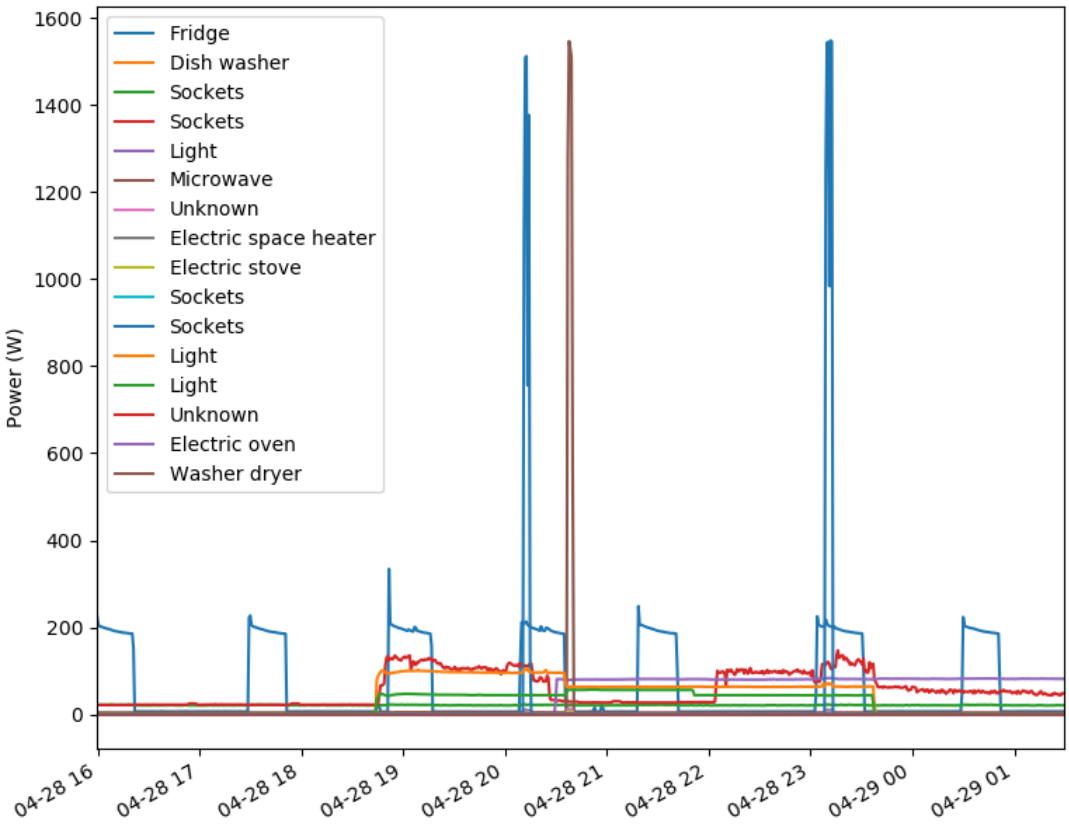

Figure 10. Small appliance dataset

Termination condition (TC) for NSGA-II impacts how long the disaggregation will run. Once TC is triggered the best individual is selected, and we start disaggregating the next measure. When the NSGA-II cycle starts, the probability of the best individual being in the population is low. If the best individual would be selected during the first cycles and the NSGA-II terminates, the accuracy of the result would be small. On the other hand, once the best individual is in the population there is no point of issuing more NSGA-II cycles. Termination condition duty is to detect this moment once the best individual has entered the population and stop NSGA-II cycle as soon as possible.

There are two termination conditions implemented in thesis:

Firstly, after reaching a certain DM function score the TC is triggered. Since tests are conducted with different training techniques and different datasets the TC needs to be agile. It is not possible to set the score of DM after which termination occurs: the best TC trigger scores will be different for each dataset. Instead, the score is extremely strict at the beginning and gets less strict for each next set of NSGA-II cycles. If the best DM value does not change for some number of cycles, the TC trigger score becomes even less strict increasing the chance of TC being triggered.

Secondly, there are some training techniques and dataset combinations that provide a small number of possible individual combinations. For these reasons if the population already contains ninety percent of possible individuals the TC will trigger. This is more likely to occur if NIKMTK FHMM was used during training.

NILMTK Disaggregators

NILMTK contains an implementation of combinational optimization. It will be used as a reference benchmark for NSGA-2 implementation. There are also other disaggregation techniques but current thesis will not use them.

Technique summary

In Table 1 you will find all learning and disaggregation techniques used in this thesis. Table 2 describes pros and cons of each technique discovered during tests.

Table 1. List of training and disaggregation algorithm combination used in thesis.

combination	**training algorithm name**	**training algorithm implemented in**	**disaggregation algorithm**	**disaggregation algorithm implemented in**	**name in charts and logs**
Fast cluster + NSGA2	Fast cluster	Thesis work	NSGA2	Thesis work	Serializer
K-means based clustering + NSGA2.	K-means based clustering.	Thesis work	NSGA2	Thesis work	Serializer_k_means
FHMM+NSGA2	FHMM	NILMTK	NSGA2	Thesis work	FHMM_NSGA_2
CO	CO	NILMTK	CO	NILMTK	CO
FHMM	FHMM	NILMTK	FHMM 1d disaggregation	NILMTK	FHMM

Table 2. Technique pros and cons

Technique	Pros	Cons
Fast cluster	Simple and fast implementation	Limited accuracy
K-means based clustering	Simple implementation	Limited accuracy Human interaction required
NSGA-II	Possible to adjust execution time and accuracy A better genetic algorithm	Slow disaggregation time
FHMM	Better accuracy Widely used in research	High memory consumption Slow disaggregation time
CO	Simple implementation High speed	Low recall in tests

Architecture

File Converter

The solution implementation consists of 3 components: file converter, client and server.

File converter role is to transform any energy consumption dataset into the NILMTK HDF5 file format. HDF (Hierarchical data format 5) is a data format and a set of tools meant to store and organize enormous amounts of data [22]. NILMTK provides enough support for this format [3]. It also proves an API for the format, enough for any NILMTK user to ignore inner workings of the format. It is a script that uses NILMTK package. NILMTK supports several data formats. HDF5 file format is similar regardless of input file format. This allows for the remaining to abstract over the analyzed data format.
In this thesis REDD and … datasets were successfully converted into HDF5 files and later used.

Client

Client is responsible for loading converted HDF5 files into memory, sending it to the server, receiving disaggregation results and then visualizing, comparing and analyzing data. It can also launch other energy disaggregation techniques provided by NILMTK. The client flow chart is visually described in a flowchart Figure 11.

The client is a python-based script. All operations in the client code are blocking and single threaded. Its lifecycle is bound to a single problem solution: once data has been disaggregated and visualized the runtime environment is terminated. The script is based on [23] and starts by loading a subset of dataset (E.G. All power consumption data of a specific building) and its time limitations. There are two categories of data: data meant for training that contains both submeter

and site meter readings and a data for disaggregation: it will contain only information about aggregated power consumption.

WebSocket was chosen as a mean of communication between server and client. Although HTTP protocol is a simpler solution it does not support transport of large amounts of data. A classical TCP socket was considered, but WebSocket message handling functionality made it a preferred candidate.

Once desired data is loaded into memory a WebSocket communication channel with the server will be opened. Once opened a handshake between client and server will be done. As a result of handshake, the client will be granted with an id of the training and disaggregation session. All subsequent WebSocket messages are in json format. Then a training stage begins. Energy consumption of all submeters will be sent through WebSocket to server. Since the training data volume may be large for any http packet or WebSocket message it is sent in small chunks in json format. Once all training data is sent client waits for the server to acknowledge that all training data has been received and processed.

Next stage is for the client to send the data meant for disaggregation. Once all data has been sent the client will send a signal that all data is sent, and it will wait for server to send all disaggregation results.

Client now has the disaggregation data; it now runs other disaggregation techniques provided by NILMTK. This is done so that the work in this thesis can be compared to other NILM techniques.

Last stage for client is to draw charts and run disaggregation accuracy analysis. Data received from server is organized in NILMTK friendly format. This allows to interface with other NILMTK tools.

During chart and accuracy analysis the following is done:

- Generate chart that compares predicted total consumption and real data from the same moment.
- Calculate accuracy, recall and f-measure.

The chart is useful as it displays how well a NILM solution focuses on either preserving power consumption states between each value in test dataset or either picking the ideal combination of device modes so that the difference between estimated chart consumption and given consumption is minimal.

Once this is done the client finishes execution and python runtime terminates.

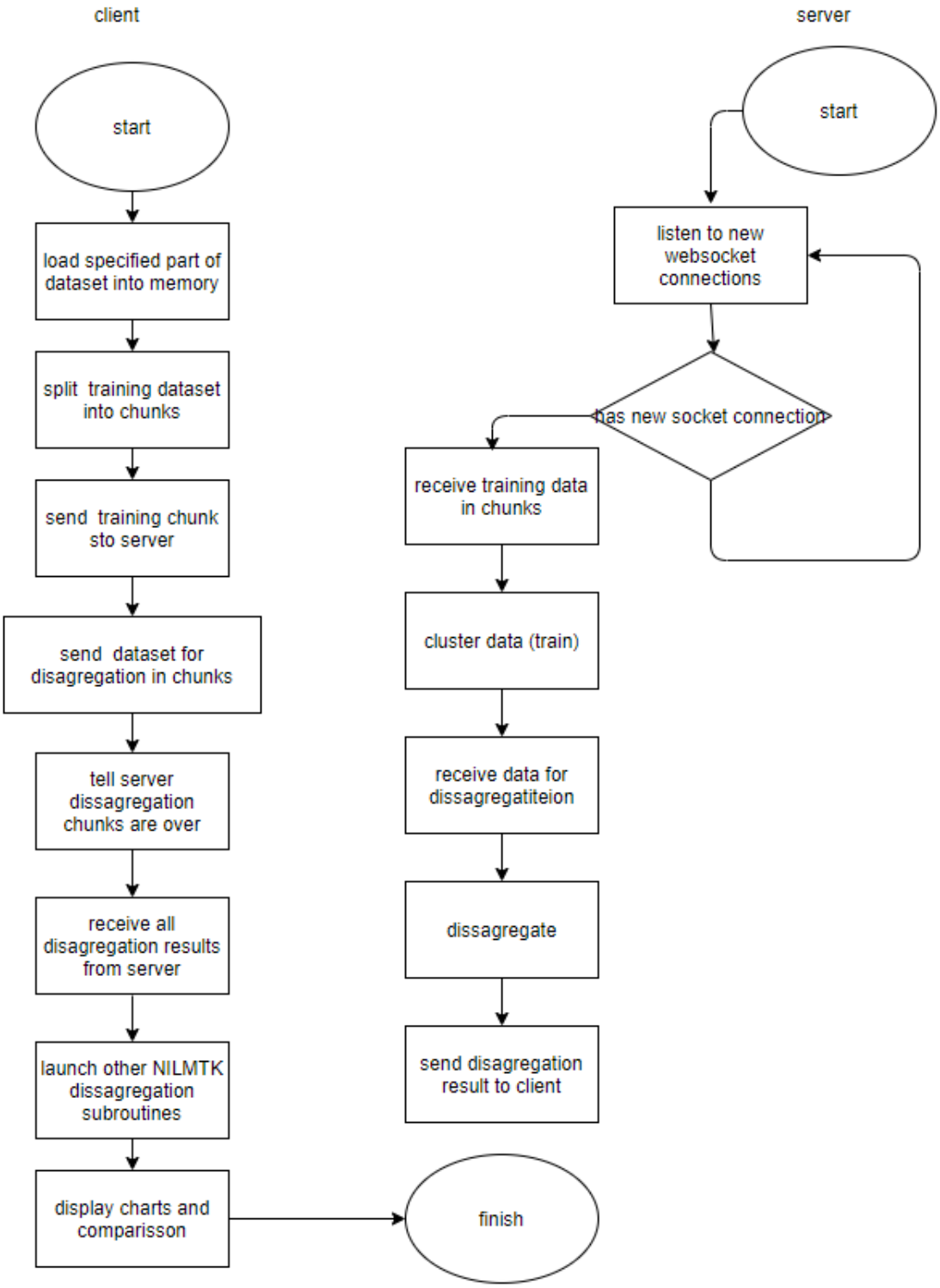

Figure 11. Client and server flowcharts

Server

Server role is to receive data from the client, both training and test data. Server opens a port and actively listens to new connections.

Once a connection is established server will send a unique identification number which will be used to distinguish this client from others. Server then expects data from the client in specific order: train data, test data and a signal to start disaggregation. Last chunk of train data contains a flag so that the server should start training, thus building a consumption model. Ideally, server should start building model right away, but the wish to keep the implementation simple dictates otherwise. Then the test data is sent, and server starts disaggregation phase once a signal is to start it is received. Once disaggregation phase finishes all calculated data is sent back to the client via WebSocket and connection terminates.

Server also logs crucial information like results of fast clustering and k-means clustering training sessions.

Architecture summary

A good question about architecture would be: “Why not to merge both client and server architectures into a single python runtime? This would even make it possible to add NSGA-2 implementation into NILMTK toolkit. “. Though there are some benefits to a monolith application a distributed solution has its own pros. These are:

- Loose coupling: server would not be dependent on data parsing and its errors.
- Server can potentially serve other clients on a remote host or a cloud: a real-world friendly business case.
- I have a lot of experience with Java and wanted to implement a solution using a Java.

Writing a Java only solution was not a realistic since the list of tools for parsing available energy consumer data sources was almost absent. Generating charts was also easier with python.

Technologies and tools

Client is built around NILMTK. It uses python runtime and Anaconda [24] to manage NILMTK library dependencies.
In order to run a client, one should install anaconda and create anaconda environment provided by NILMTK.

NILMTK stores datasets mostly in pandas [25]. In order to be compatible with NILMTK a lot of client produced data is stored in pandas. Matplotlib [26] is used to generate charts illustrated in this paper. Matplotlib is used because it is integrated into pandas [27] which itself is heavily used by NILMTK. Since Matplotlib is widely used in data science, there are a lot of instructions to a lot of chart related tasks. Python built in WebSocket package is used to do communications with server. NILMTK tools are interfaced around Pandas data-frames based format. Thus, client stores train, test and disaggregation data in pandas as well.

Server is run in Java environment. It is picked since I have experience with Java and comfortable with its environment. Gradle [28] is used for building, testing, deploying and library management. It is a well-supported Java tool and offers more minimalistic configuration compared to other Java environment build tools. Server runtime is based on spring boot [29]. Spring boot was chosen because it has integrated got a WebSocket support [30] and dependency injection. Dependency injection helps with component organization. Server has no persistence: all data is stored in memory. Supervised training is performed in a multi-threaded environment (each thread per each device) which makes the process faster.

Details on how to run client and server are in README.md file.

Evaluation

To have a good overview of the NILMTK a range of different accuracy metrics can be used.

True positives, false positives, false negatives, true negatives.

With datasets providing data for supervised learning, it is possible to compare results of the disaggregation with individual appliance on-off states at the same time. Each device in the house on/off state can be compared between control dataset and disaggregation result at each timeframe. These statistical errors and states can be included in statistics:

- True positive (TP): device on state was correctly predicted during disaggregation state.
- False positive (FP) or type 1 error: device on state was identified as on, but it was disabled in control dataset.
- False negative or type 2 error (FN): device on state was identified as on, but it was enabled in control dataset.
- True negative (TN): device was correctly identified as off.

The device was considered on when its power consumption was greater than certain margin.

Per each device and electric measure in the dataset used in disaggregation we calculate number of each of these metrics' occurrences. Based on these statistics precision (1), recall (2) and F-score (3) can be calculated [31].

$$Precision = \frac{TP}{(TP + FP)} \qquad (1)$$

$$Recall = \frac{TP}{(TP + FN)} \qquad (2)$$

$$F - score = \frac{2\ Precision\ Recall}{(Precision + \ Recall)} \qquad (3)$$

The larger each of these scores and closer to 1 the better disaggregation technique performs.

These scores were calculated separately for each device in the dataset and combined for each disaggregation technique.

For a more detailed analysis, the test in thesis work collects metrics not only for all devices combined but per each device as well.

Results and analysis

We publish recall and accuracy results against all NILM techniques in Table 3 and Table 4.

Table 3. REDD disaggregation results

	accuracy	recall	F-score	Total time (sec)	Training time (sec)
K-means + NSGA II	**0.44**	**0.69**	**0.53**	**1797**	**18.59**
Fast cluster + NSGA II	**0.48**	**0.55**	**0.51**	**1775**	**17.62**
FHMM + NSGA II	**0.66**	**0.33**	**0.44**	**1079**	**34.20**
Combinational optimization	**0.87**	**0.31**	**0.46**	**6.66**	**5.26**
FHMM + FHMM	**0.83**	**0.86**	**0.85**	**14467**	**43.75**

Table 4. AMPDs2 disaggregation results

	accuracy	recall	F-score	Total time (sec)	Training time (sec)
K-means + NSGA II	**0.62**	**0.73**	**0.67**	**2489**	**4.92**
Fast cluster + NSGA II	**0.66**	**0.70**	**0.68**	**2634**	**4.60**
FHMM + NSGA II*	**0.85**	**0.79**	**0.81**	**1756**	**19.49**
Combinational optimization	**0.73**	**0.47**	**0.57**	**4.42**	**4.02**
FHMM + FHMM*	**0.94**	**0.97**	**0.96**	**3919**	**19.85**

*Not all devices were used in training set.

Let us compare full NILMTK FHMM with NSGA-II implementation + FHMM training phase since they use similar training technique and have best results.

Test results show that FHMM is a beast when it comes to accuracy and recall. However, high memory and CPU time consumption do not let it scale well for everyday usage. This is especially true the more devices there are in training dataset (test trials with limited number of devices in dataset showed small execution times for the same datasets). While it is possible to run REDD based disaggregation without limits it is impossible to do so with AMPDs2. Set of training devices was thus limited.

It takes 10 more times to disaggregate REDD dataset by FHMM in NILMTK than by NSGA-II.

On the other hand, NSGA-II can be easily scaled since it can be configured to consume less CPU and memory. But it does however struggle with REDD dataset when it comes to accuracy and recall. In Figure 12 and Figure 13 you can notice that NSGA-II is more likely to change states between two disaggregation measures. This is especially true for appliances with low power consumption as there are less problems with high consumption devices as seen in Figure 14.

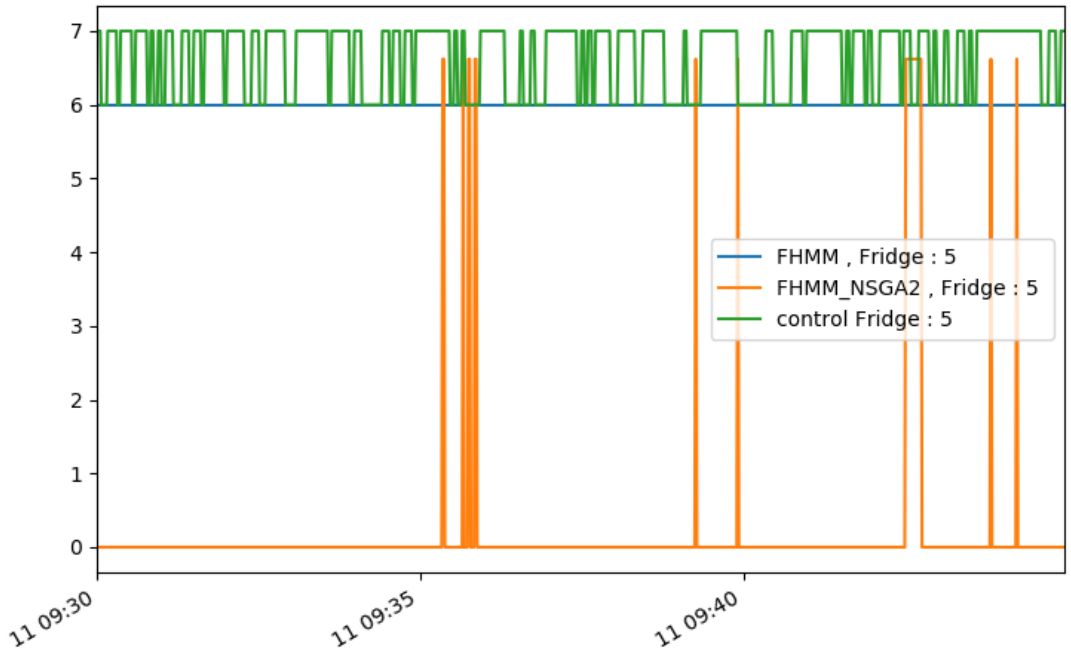

Figure 12. Predicted result and control value for fridge power consumption IN REDD dataset

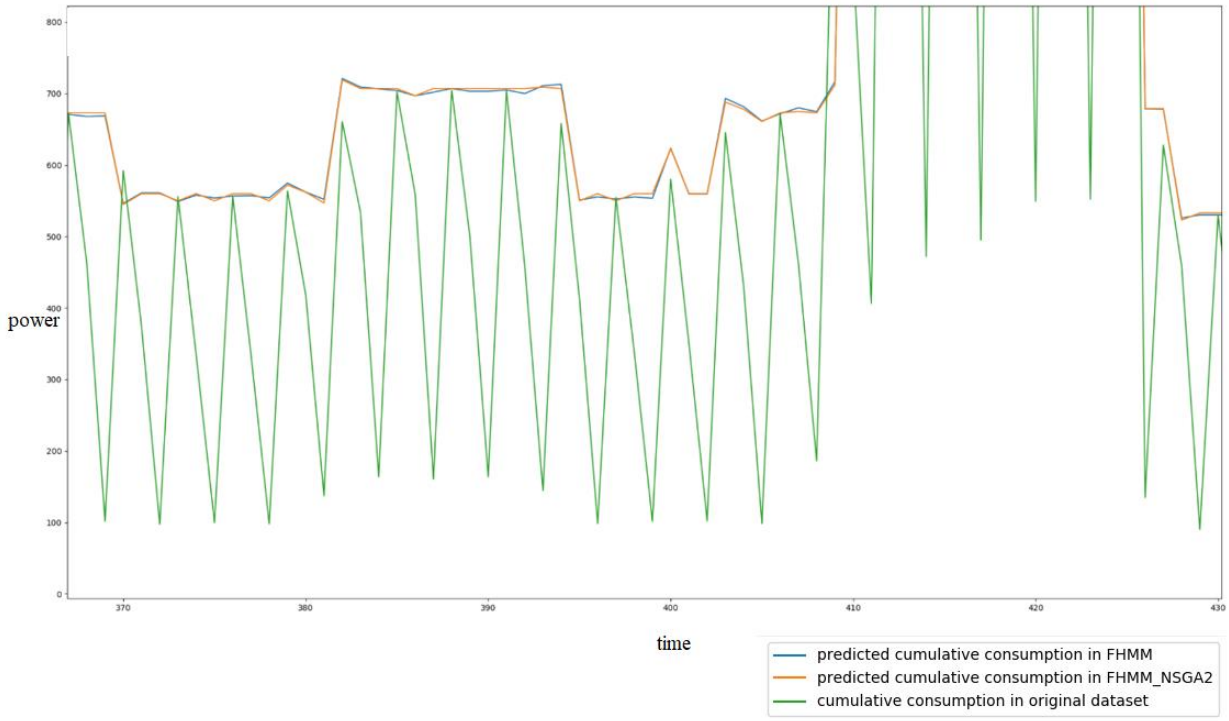

Figure 13. Predicted total result and control site meter value in REDD dataset

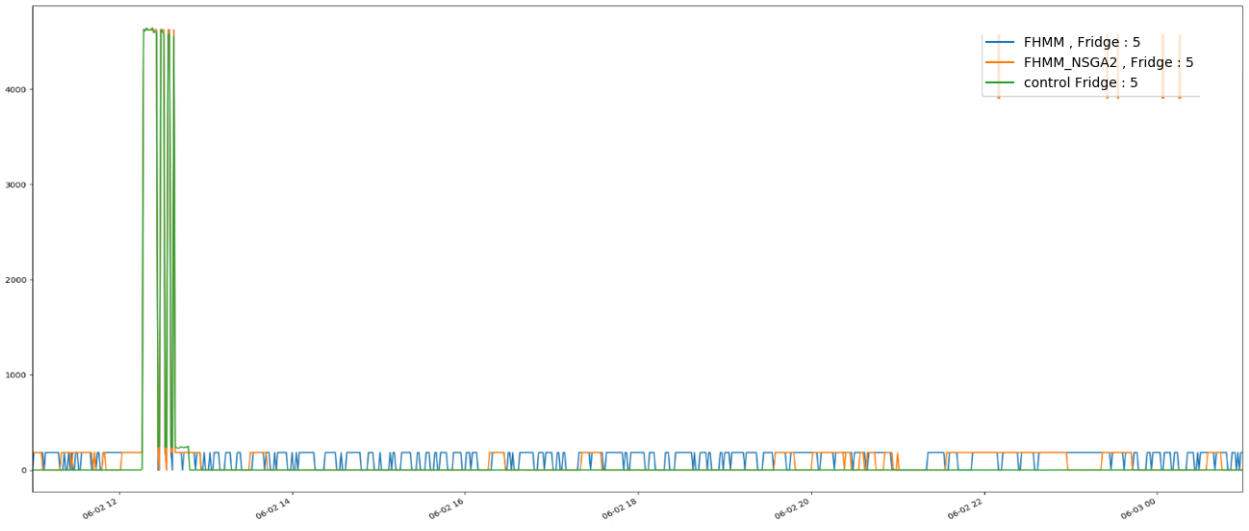

Figure 14. Predicted and control power consumption for high consumption device in AMDs2 dataset

Once tests were conducted it was clear that despite decision making function [5] penalizes for switching devices states between measures, it would still generate predictions that have a sum of all enabled device states close to the measure in the dataset. This is observed among all datasets and their devices.

This can also be seen in Figure 15 with other NSGA-II implementations, when compared with CO.

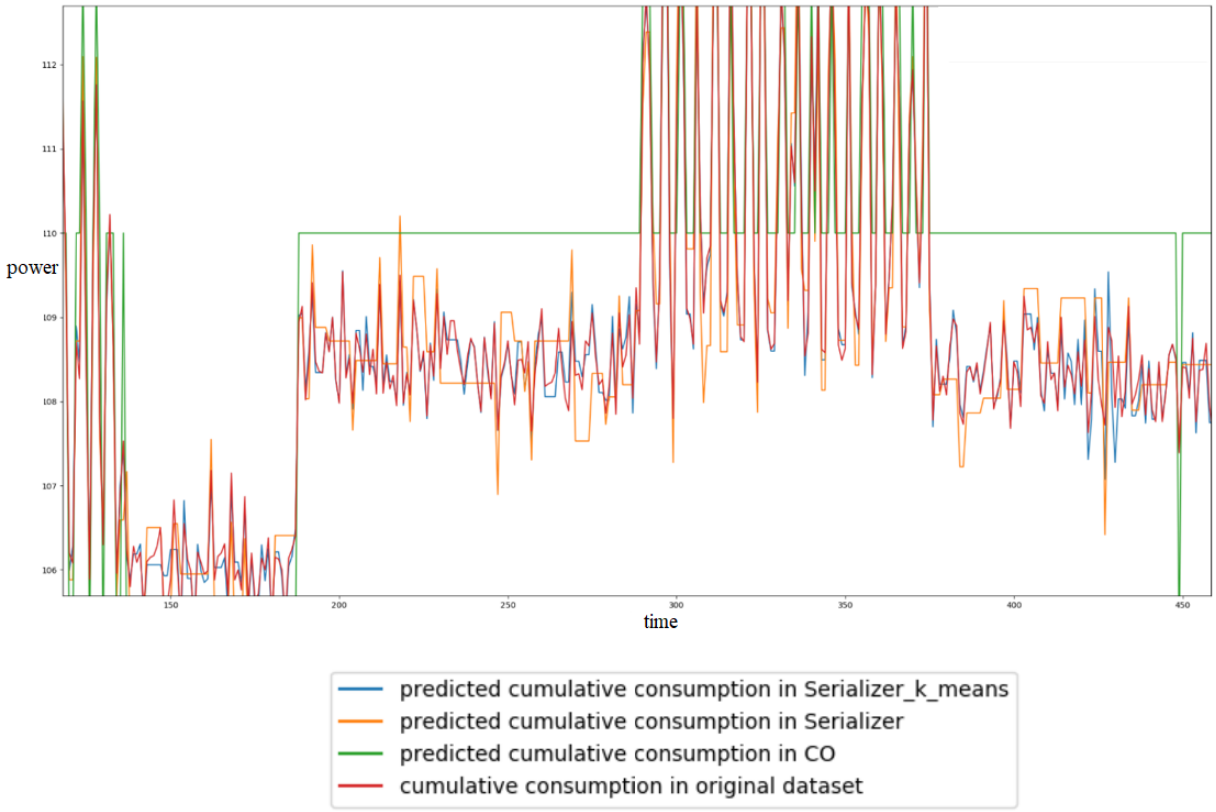

Figure 15. Predicted total result and control site meter value in REDD dataset

Although same pattern can also be observed in CO for individual devices. Figure 16 and Figure 16 show that both CO and NSGA-II tend to actively switch on/off states of individual devices.

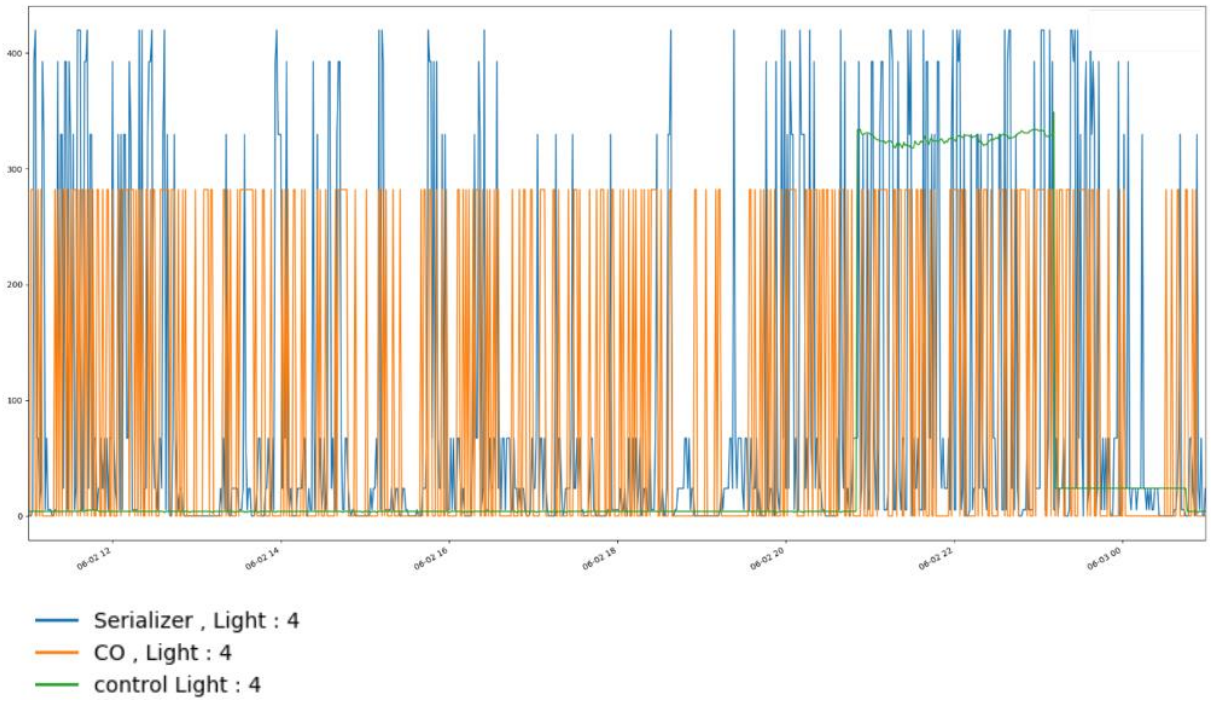

Figure 16. Predicted CO and NSGA-II and control Light power value in AMPDs2 dataset

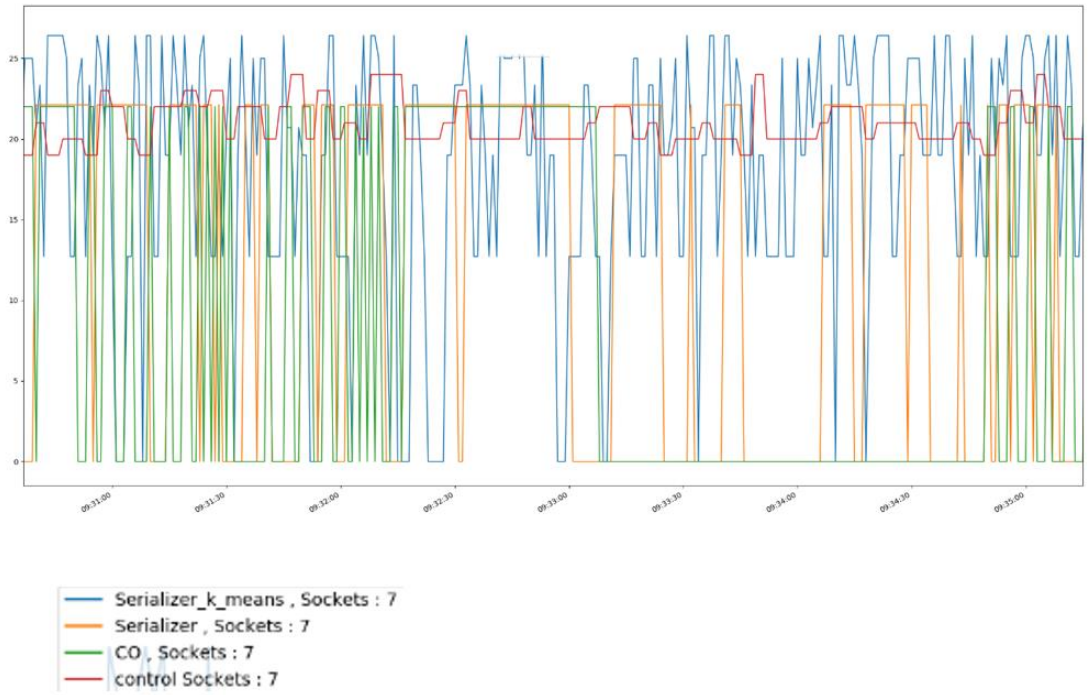

Figure 17 Predicted CO and NSGA-II and control socket power value in REDD dataset

This could be happening because not enough appliance consumption states are known:

1. NILMTK CO code analysis shows it can generate maximum 3 device consumption states.
2. Low count of states (0-6) was manually specified while generating consumption model with K-means.
3. Fast cluster algorithm design ensures it will not generate separate states with a difference more than 10 W

2 and 3 could be changed to ensure more device consumption modes are generated. This could be done in future research.

This fact of rapid appliance mode change combined with test results from [32] suggest there is a chance to improve the accuracy of NSGA-II if DM function would penalize more for switching appliances' states between measures. Let us test this hypothesis by slightly altering the DM (4) function for NSGA-II.

$$\mathrm{DM}_{cost} = f_1(X[n]) + \left[(1 + 2f_2(X[n]))\sqrt{|f_1(X[n]) - f_1(X[n-1])|}\right] \quad (4)$$

After running the tests with new DM function on REDD dataset, we can see in Figure 18 that NSGA-II is less likely to switch device modes.

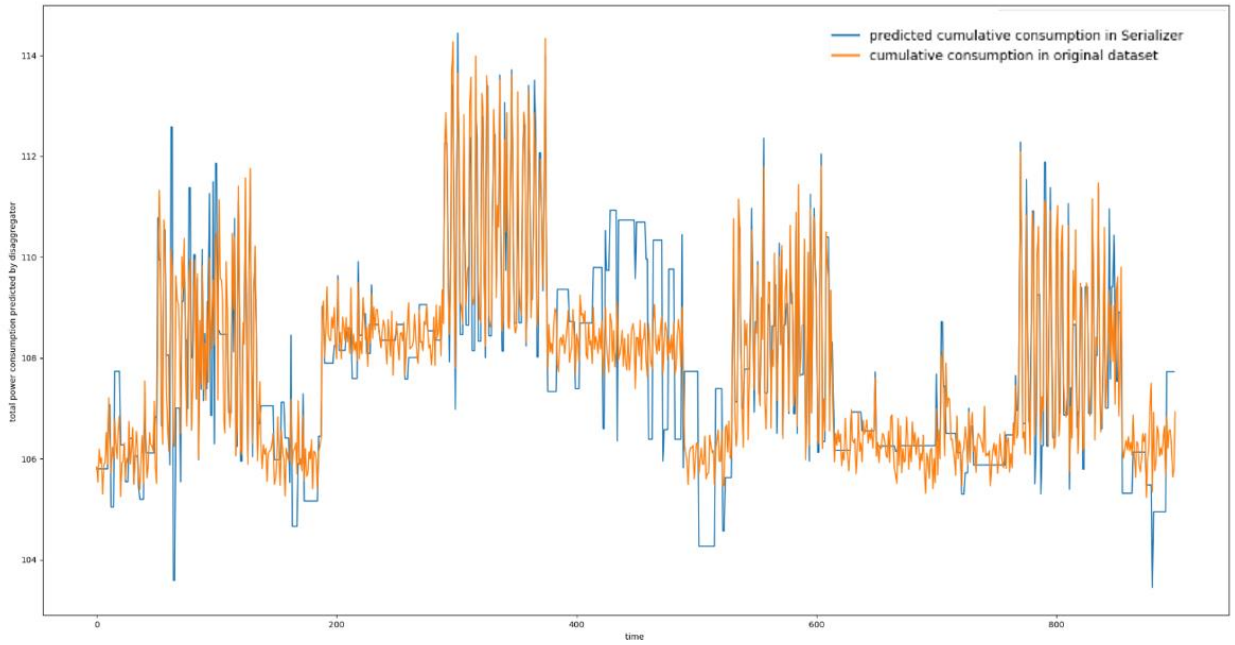

Figure 18. Altered DM disaggregation result

Table 5. Modified DM results with REDD dataset

	accuracy	recall	F-score	Total time (sec)
Fast cluster + NSGA II	**0.48**	**0.55**	**0.51**	**1775**
Fast cluster + NSGAII with (4)	**0.50**	**0.63**	**0.56**	**1661**
FHMM + NSGA II	**0.66**	**0.33**	**0.44**	**1079**
FHMM + NSGA II with (4)	**0.48**	**0.57**	**0.52**	**1605**
Fast cluster+ NSGA II with (4)	**0.46**	**0.98**	**0.55**	**1640**

As seen in test results with REDD dataset in Table 5 the recall of disaggregation has increased. But accuracy changes differ for techniques.

$$\mathrm{DM}_{cost} = f_1(X[n]) + \left[(2 + f_2(X[n]))\sqrt{|f_1(X[n]) - f_1(X[n-1])|}\right] \quad (5)$$

We have also tried NSGA-II with DM function (5) which has lower impact of f1 of individual with the site meter value. This function has the best results so far.

Table 6. Modified DM results with REDD dataset

	accuracy	recall	F-score
K-means + NSGA II	**0.42**	**0.56**	**0.48**
K-means + NSGAII with (5)	**0.53**	**0.58**	**0.56**
FHMM + NSGA II	**0.66**	**0.33**	**0.44**
FHMM + NSGA II with (5)	**0.75**	**0.36**	**0.48**
Fast cluster + NSGA II	**0.48**	**0.55**	**0.51**
Fast cluster + NSGAII with (5)	**0.46**	**0.56**	**0.50**

As seen in Table 6 the recall and accuracy with new DM function and REDD dataset are either better or the same as the original.

This does not mean that DM proposed in (6) is wrong, but this means that the DM should be further researched with more datasets and more training techniques. After all current test in this paper haves got low F-score to begin with. This could also be a result of large number of appliances in the dataset.

Other observations

- Comparison between REDD and AMPDs2 datasets also hints that accuracy and recall may differentiate between datasets. All training and disaggregation techniques showed better F-score in AMPDs2 dataset. This is also true for NILMTK provided techniques. We cannot explain the reason for this trend.
- CO tends to have a higher accuracy and low recall rate. This means that while a lot of predicted appliances ON states were accurate, there were a lot of cases when enabled appliance was not predicted correctly.
- The more devices there are in training dataset the more CPU time during disaggregation and memory during training FHMM implementation in NILMTK needs. This is seen in Table 3 and Table 4 results. It was also observed during tests that lowering number of devices used in training dataset resulted in lower memory and CPU time consumption by FHMM.
- Despite being not that complex on paper it took a lot of time to do fast cluster training during tests. It is possible that the implementation has got a problem.
- High recall for fast cluster and NSGA-II combination in Table 5 is probably an anomaly. We need more tests and datasets to control this.

Experience with NILMTK

NILMTK has a steep learning curve for newcomers. Its data structures are not documented enough leading to difficulties during implementation. Sometimes same code resulted in NILMTK returning different python types when running with different datasets. I still have a feeling that I would be more productive without NILMTK. NILMTK provides a sample for running disaggregation techniques, but it would be better if this code sample were already in the NILMTK toolkit. NILMTK also does not contain any accuracy and recall tests: in my opinion if NILMTK had these tests in its package then it could be standardized and used by other researchers.

Despite all these shortcomings, NIMTK does come with its benefits like dataset integration and packaged NILM solutions for reference. It would be great if contributions to the project continue.

This thesis shows that NSGA-II is compatible with NILMTK, thus both NILMTK and NSGA-II research would also benefit a lot if NILMTK had a proper implementation of NSGA-II algorithm coming as a part of NILMTK package. This would mean that more researchers would compare NSGA-II to other algorithms, run NSGA-II with more datasets and learn its specifics.

Future work vector suggestions from author

As seen in Figure 19 there are appliances (a socket in the example), that have a continuous fluctuating power consumption in the dataset. This puts stress on the disaggregation techniques as it is hard for trained model to represent such appliance behavior. As seen in tests NSGA-II implementation used in this thesis struggles with such appliances.

It is though possible to mark such devices as volatile, calculate power consumption limits during volatility and pass this knowledge in consumption model. It is also possible to add more dimensions to NSGA-II and change its decision function. The third dimension in NSGA-II could improve such results if it detected power fluctuations in nearby measurements and increased the probability of such devices being enabled in disaggregation result.

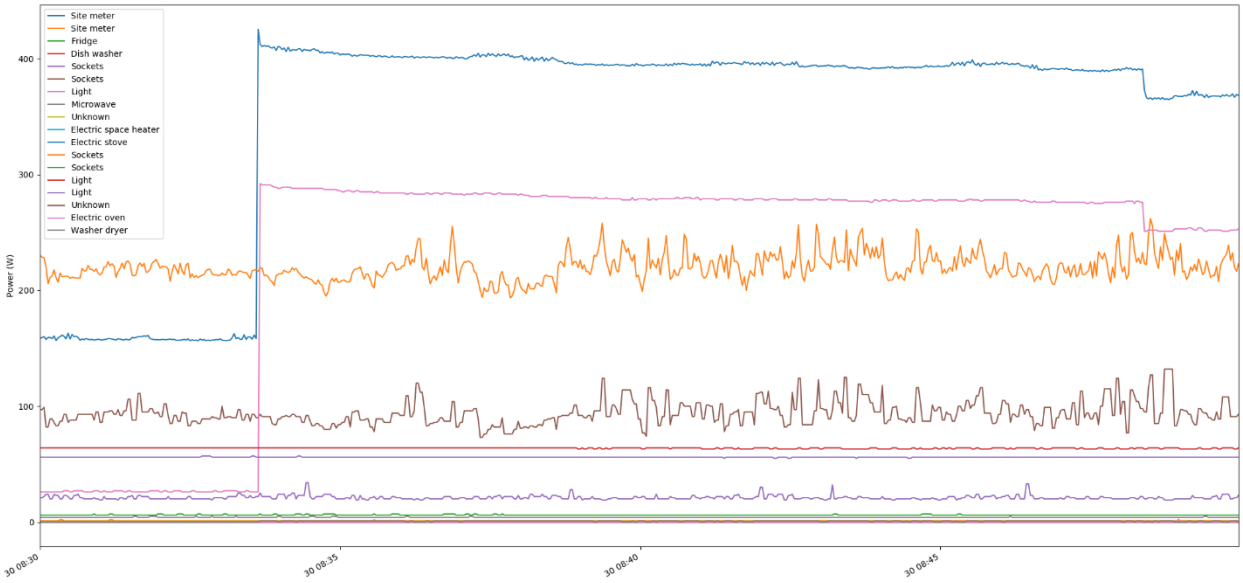

Figure 19. Redd dataset sample with continuous fluctuating power consumption

Speaking of other NSGA-II in NILM domain improvement work we advise to conduct more research regarding DM function declared in [5] as tests in this thesis show it is not guaranteed to be optimal.

Running tests took a lot of time during work on thesis and this has resulted in slow productivity. During test runs only a single thread was used. This may be a problem should NSGA-II be widely used in NILM domain since single threaded algorithm lacks in scalability. The possibility of implementing a parallel NSGA-II in this thesis was observed, though abandoned as it was considered complex. There are papers [33] that suggest that NSGA-II can be run in parallel, thus a parallel NSGA-II for NILM domain research is advised. This would help with productivity and possibly let to implement the NILM solution on the GPU which is good in parallel execution.

Fast cluster and K-means based clustering techniques implemented in this thesis and their consumption had low accuracy and recall compared with FHMM based model. During design of both techniques, different changes to both were considered but the simplest solutions were preferred. It would make sense to tweak both techniques to improve accuracy, recall of NSGA-II and their CPU time.

Lastly, NILM domain would benefit if more NSGA-II compatible training techniques were researched.

All code written for this thesis will be made publicly available in Github.

Summary

This thesis had two goals: one was to implement a software capable of learning and then disaggregating energy consumption datasets and second was to find a compatible training technique for NSGA-II based disaggregation.
The first goal was fully achieved: work on thesis has resulted in client and server combination capable of parsing multiple datasets, solving NILM domain problem and then calculating precision and recall of disaggregation results.
The second goal achievement is arguable. Although several of training techniques have resulted in device consumption model that can be used with NSGA-II, the disaggregation result accuracy and recall have small or comparable margins compared to results published in other papers [3][6]. This suggests that more research needs to be conducted in field of training algorithms that are compatible with NSGA-II.

On the bright side, it was found that the DM function for NSGA-II declared in [6] could not be optimal and that it may need more research.

During work on thesis, I have learned a lot about NILM domain problems, solutions to them and research. Several NILM techniques were run and tested. Moreover, it was learned that different datasets result in different accuracy and recall of these techniques. I also learned about python scientific tools and some data processing algorithms.

NILM domain is relatively young and it has a lot of research to be done until there is a realistic NILM solution to energy problems. A good solution installed in many households may improve population consumer behavior thus helping to stop climate change. Hopefully, this paper helps other people to find and explore better NILM solutions. To help this I plan to make the code used in this project publicly available after this thesis is defended.

References

[1] H. Liu, Non-intrusive Load Monitoring: Theory, Technologies and Applications, 2020.

[2] G. W. Hart, "Non-intrusive appliance monitor apparatus". United States of America Pat US4858141A, 15 08 1986.

[3] N. Batra, J. Kelly, O. Parson, H. Dutta, W. Knottenbelt, A. Rogers, A. Singh and M. Srivastava, "NILMTK: An Open Source Toolkit for Non-intrusive Load," 2014.

[4] B. Y. Karim Said Barsim, "On the Feasibility of Generic Deep Disaggregation for Single-Load Extraction," 2018.

[5] R. Machlev, J. Belikov, Y. Beck and Y. Levron, "MO-NILM: A multi-objective evolutionary algorithm for NILM classification," *Energy & Buildings*, 2019.

[6] J. Z. Kolter and M. J. Johnson, "REDD: A Public Data Set for Energy Disaggregation Research," 2011.

[7] "NILMTK: Non-Intrusive Load Monitoring Toolkit," [Online]. Available: https://github.com/nilmtk/nilmtk. [Accessed 18 05 2021].

[8] "AMPds2: The Almanac of Minutely Power dataset (Version 2)," [Online]. Available: https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/FIE0S4. [Accessed 10 05 2021].

[9] "Writing a dataset converter," [Online]. Available: https://github.com/nilmtk/nilmtk/blob/master/docs/manual/development_guide/writing_a_dataset_converter.md. [Accessed 13 05 2021].

[10] T. P. R. G. L. J.-C. L. B. Y. R. Mohamed Nait Meziane, "A New Measurement System for High FrequencyNILM with Controlled Aggregation Scenarios," 2016.

[11] "Initial REDD Release," [Online]. Available: http://redd.csail.mit.edu/. [Accessed 20 04 2021].

[12] S. Makonin, B. Ellert, I. V. Bajic and F. Popowich, "Electricity, water, and natural gas consumption of a residential house in Canada from 2012 to 2014," 2016.

[13] "AMPds," [Online]. Available: http://ampds.org/. [Accessed 10 05 2021].

[14] S. Makonin, Z. J. Wang and C. Tumpach, "RAE: The Rainforest Automation Energy Dataset for Smart Grid Meter Data Analysis," 2017.

[15] A. Faustine, N. H. Mvungi, S. Kaijage and M. Kisangiri, "A Survey on Non-Intrusive Load Monitoring Methodiesand Techniques for Energy Disaggregation Problem," 2017.

[16] K. S. Barsim and B. Yang, "Toward a semi-supervised non-intrusive load monitoring system for event-based energy disaggregation," 2015.

[17] Y. G. C. J. S. Ruoxi Jia, "A Fully Unsupervised Non-intrusive LoadMonitoring Framework," 2015.

[18] J. MacQueen, "Some methods for classification and analysis of multivariate observations," 1967.

[19] S. Dasgupta and Y. Freund, "Random projection trees for vector quantization," 2008.

[20] M. I. Jordan and Z. Ghahramani, "Factorial Hidden Markov Models," 1997.

[21] K. Deb, A. Pratap, S. Agarwal and T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm:NSGA-II," 2002.

[22] "HDF5," The HDF Group, [Online]. Available: https://portal.hdfgroup.org/display/HDF5/HDF5. [Accessed 18 05 2021].

[23] "Disaggregation and metrics," [Online]. Available: https://github.com/nilmtk/nilmtk/blob/master/docs/manual/user_guide/disaggregation_and_metrics.ipynb. [Accessed 18 05 2021].

[24] "Anaconda," [Online]. Available: https://www.anaconda.com/products/individual. [Accessed 18 05 2021].

[25] "Pandas," [Online]. Available: https://pandas.pydata.org/. [Accessed 18 05 2021].

[26] "Matplotlib," [Online]. Available: https://matplotlib.org/. [Accessed 13 05 2021].

[27] "Visualization," Pandas, [Online]. Available: https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html. [Accessed 13 05 2021].

[28] "Gradle," [Online]. Available: https://gradle.org/. [Accessed 13 05 2021].

[29] "Spring boot," [Online]. Available: https://spring.io/projects/spring-boot. [Accessed 13 05 2021].

[30] baeldung, "Reactive WebSockets with Spring 5," 15 09 2020. [Online]. Available: https://www.baeldung.com/spring-5-reactive-websockets. [Accessed 13 05 2021].

[31] D. Powers , "Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation," 2020.

[32] I. Gretsi, "Energy load disaggregation approach based on heuristic optimization algorithms," 2019.

[33] S. Bandyopadhyay and R. Bhattacharya, "Solving multi-objective parallel machine scheduling problem by a modified NSGA-II," *Applied Mathematical Modelling,* vol. 37, no. 10-11, pp. 6718-6729, 2013.

Appendix 1 – Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I, Sergey Korneev

1. Grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis "NILM: combining NSGA-2 with different training techniques", supervised by Juri Belikov and Margarita Spitšakova
   1.1. to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;
   1.2. to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.
2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.
3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

1 The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.