Exploring Change Point Detection in Network Equipment Logs

Tim Björk, tim.bjork@live.se

Faculty of Health, Science and Technology

Master thesis in Computer Science

Second Cycle, 30 hp (ECTS)

Supervisor: Dr. Johan Garcia, University of Karlstad, Karlstad, SWE <johan.garcia@kau.se>

Examiner: Dr. Leonardo Martucci, University of Karlstad, Karlstad, SWE <leonardo.martucci@kau.se>

Karlstad, August 12, 2021

Abstract

Change point detection (CPD) is the method of detecting sudden changes in time series, and its importance is great concerning network traffic. With increased knowledge of occurring changes in data logs due to updates in networking equipment, a deeper understanding is allowed for interactions between the updates and the operational resource usage. In a data log that reflects the amount of network traffic, there are large variations in the time series because of reasons such as connection count or external changes to the system. To circumvent these unwanted variation changes and assort the deliberate variation changes is a challenge. In this thesis, we utilize data logs retrieved from a network equipment vendor to detect changes, then compare the detected changes to when firmware/signature updates were applied, configuration changes were made, etc. with the goal to achieve a deeper understanding of any interaction between firmware/signature/configuration changes and operational resource usage. Challenges in the data quality and data processing are addressed through data manipulation to counteract anomalies and unwanted variation, as well as experimentation with parameters to achieve the most ideal settings. Results are produced through experiments to test the accuracy of the various change point detection methods, and for investigation of various parameter settings. Through trial and error, a satisfactory configuration is achieved and used in large scale log detection experiments. The results from the experiments conclude that additional information about how changes in variation arises is required to derive the desired understanding.

Keywords

Change point detection, log change detection, time series data, signal processing

Sammanfattning

Förändringspunktsdetektering är en metod som går ut på att detektera plötsliga förändringar i tidsseriedata, och är en viktig del inom nätverkstrafikområdet. Med ökad kunskap om förändringar i dataloggar på grund av uppdateringar i nätverksutrustning, tillåts en djupare förståelse för interaktioner mellan uppdateringarna och den operativa resursanvändningen. I en datalogg som återspeglar mängden nätverkstrafik finns det stora variationer i tidsserien, detta på grund av orsaker som anslutningsantal eller externa förändringar av systemet. Att kringgå dessa dåliga variationer, och sortera fram de avsiktliga variationerna, är en utmaning. I detta arbete använder vi dataloggar som hämtats från en leverantör av nätverksutrustning för att upptäcka förändringar i dessa, för att sedan jämföra de upptäckta förändringarna mot när programvara/signatur-uppdateringar tillämpades, konfigurationsförändringar gjordes osv. Detta görs med målet att uppnå en djupare förståelse för all interaktion mellan programvara/signatur/konfigurationsförändringar och operativ resursanvändning. Utmaningar i datakvaliteten och databehandlingen hanteras genom datamanipulation för att motverka avvikelser och oönskad variation, samt genom experimentering med parametrar för att uppnå de mest ideala inställningarna. Resultaten produceras genom experiment för att testa noggrannheten hos de olika metoderna för ändringspunktdetektering samt undersökning av olika parameterinställningar. Genom försök och misstag uppnås en tillfredsställande konfiguration som används i storskaliga loggdetekteringsexperiment. Från resultaten av experimenten dras slutsatsen att ytterligare information om hur förändringar i variation uppkommer krävs för att få den önskade förståelsen.

Nyckelord

Förändringspunktsdetektering, loggförändringsdetektering, tidsseriedata, signalbehandling

Acknowledgements

First I would like to thank Sandvine for the opportunity to do this thesis work, and a big gratitude to Anders Waldenborg from Sandvine for the assistance.

I would also like to give a special thanks to my supervisor from Karlstad University, Johan Garcia, for the continuous help throughout the project. Thank you for a great cooperation.

Last but not least I would like to express my gratitude to family and friends for the support and keeping my spirit up.

Contents

Chapter 1

Introduction

This thesis work concerns the analysis of log data, using change point detection to detect changes over time in the data and comparing them to updates or changes in the systems. The purpose is to improve the understanding of any interaction between firmware/configuration changes and operational resource usage.

1.1 Background

Network traffic has been increasing for a long time and continues to do so. To be sure the networks operates well, various network equipment is required. This equipment needs to be as effective as possible and to achieve this, you need to know the effects of performance when firmware or signature updates are applied in the systems. The study of change point detection is a method of detecting when, at a certain point in time in a series of data with time stamps for each value, a change in the behavior occurs. Both the theoretical and the practical aspects of change point detection is covered in many works on the subject. New methods and algorithms are provided frequently, for instance by Anastasiou et al. [4], Arlot et al. [5] and Killick et al. [18], but also variations of already existing methods Fryzlewicz et al. [14]. However, Van den Burg et al. [11] is one of the earlier works that evaluates the different change point detection methods on real world time series. Most of the papers introduces new methods and algorithms that are generating data with known change points where the accuracy and model fit is evaluated, but the downside of such an evaluation is that the generated synthetic data is very often particular to the paper which which makes comparisons hard. To properly evaluate and understand the outcome of the evaluations, one must evaluate them on a large set of the same sequential data. This paper focuses on time series derived from data logs and analysis with a statistical approach.

1.2 Problem Description

There are a couple of challenges regarding this work, for example choosing the best methods for the change point detection or finding the correct parameter settings in the various functions. However, the main challenge is the unpredictable changes in the variation of the data, which can occur for example because of noise, because of the user impact or some other external impact. In this thesis, we strive to answer the question of ”how can we manipulate the features to reduce the effect of unrelated value and variation changes, and still be able to produce as good results as possible and gain a better understanding from the produced results ?”

1.3 Thesis Objective

Sandvine develops equipment to ensure high-quality connectivity for a very large amount of users. When deployed, this equipment generates a considerable amount of log data, reflecting for example the processing load and memory usage. The objective of this thesis is utilizing these logs to detect anomalies/changes with a statistical approach, then comparing the detected anomalies/changes to when firmware updates were applied, configuration changes were made, etc.

1.4 Thesis Goals

The goal of this thesis work is to achieve a deeper understanding of any interaction between firmware updates/configuration changes and operational resource usage.

1.5 Ethics and Sustainability

Technical equipment placed in data communication networks could potentially be used for both good and bad things, and there is always a risk of misuse of technical equipment by various sources. In this thesis work, though, it seems like there are no ethical issues that has to be considered, seeing that the features dealt with in the data logs are segments like for example CPU usage which does not induce any ethical issues. However, in a more general sense, this work could potentially provide improvements to network monitoring equipment, which in turn could affect the ethical point of view if the equipment is used for ethically dubious things.

1.6 Methodology

Initially, synthetic data is generated for experimentation with the different methods and algorithms that Ruptures(see Section 1.6.1) has to offer. These different functions

are to be applied to the actual data sent from Sandvine. Pandas is used for the data analysis [25] and Numpy for arithmetic computations [16] (see Section 1.6.2). Plotting functions of different sorts are created to be used for observational purposes. The first log to be analyzed is for one month of activity from December 2020 to January 2021. The different features in the log are controlled, analyzed and plotted to get a picture and a feel for what it looks like. However, the logs received are big and has many features that does not contain useful information (or no information) as well having systems that seems to be shut down for periods of time, which means that the logs has to be cut down and fixed. After the log handling, the different columns (or features) are manipulated, plotted and visualized to be studied and discussed. Further, the manipulated features are experimented with through Ruptures, and the results are then be used for further feature manipulations and testing. This process repeats itself successively when new data files are received. However, the rest of the data files that are to be analyzed varies in both time range, number of system IDs and features, meaning several changes are made for each one received until a more general solution is produced. The combinations of metrics with parameters are gradually optimized and when a satisfactory setting is achieved, Ruptures is executed on a large scale log which provides the final results to be analyzed and discussed.

1.6.1 Ruptures

Ruptures is a Python library for offline change point detection, which provides methods for analyzing and segmentation of signals [26]. Ruptures is also very focused on a comprehensive interface with a lot of documentation. The different algorithms implemented include both approximate and exact detection for different models. Additionally, thanks to the modular structure of the interface, algorithms and models can be connected and extended within this package. Ruptures includes many different cost functions (such as kernelized mean change, Gaussian process change, least absolute/squared deviation etc.), several search methods (like Pelt, Binary segmentation etc.) and signal generation functions (Linear, constant, Gaussian or sinusoidal) as well as evaluation methods to evaluate the results of experiments with combinations of the other metrics. Ruptures can also provide plots of the change point detection results.

1.6.2 NumPy and Pandas

NumPy is a package for Python, a fundamental package for scientific computing. It is a library that provides a large number of different features such as a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more [16].

Pandas is also a package for Python, however its focus is to provide assistance when

working with for example databases. A table in pandas is called a ”DataFrame”, which is frequently used for this work. Pandas provides support for integration of many different file formats into DataFrames. It is also great for filtering/selecting data with respect to different conditions, rows, columns etc. Pandas has great support for time series and has an extensive set of tools for working with dates, times, and time indexed data as well as many more great features [25].

1.7 Stakeholders

This project is done for the company Sandvine. Sandvine needed assistance for analysis of their log data to better understand the relationship between operational resource usage and system updates/configurations. Being able to map changes in operational resource usage give rise to a deeper understanding of the influence of system updates/configuration changes.

1.8 Delimitations

The methods and functions at use in this thesis work is exclusively the ones implemented in Ruptures. As a consequence, a number of other existing approaches will not be examined.

1.9 Outline

This thesis is structured as follows. In chapter 2, we describe the background works of the change point detection topic as well as related topics, and also discuss about which of the earlier works are useful for this thesis. In Chapter 3, an overview of the data sent from Sandvine is described, such as analysis of the original features in the data logs. Further, plots are created to visualize the features as signals, and the features are manipulated and re-plotted to visualize what differences the feature manipulations make. Lastly, the initial change point investigations are introduced in the form of plots. Chapter 4 includes most of the Ruptures evaluation experiments where the aim is to achieve the best parameter settings and filter the amount of metrics used. Also, some additional feature manipulations are included. Chapter 5 contains the results and discussion about the results of the final few experiments as well as the large scale log detection test where all the earlier optimizations are utilized. In chapter 6 there are conclusions, discussions and explanation of the possible future work of this thesis.

Chapter 2

Background and Related Work

This chapter will provide a background of the thesis subject area as well as a detailed description of the earlier works, which varies in relevance to the specific subject area along with a discussion about what is useful and what is not.

2.1 Background

This section will introduce change point detection on a more advanced level.

The initial change point detection experiments go back to the 50s according to Truong et al. [26]. It was for industrial quality control purposes to locate a shift in the mean of independent identically distributed Gaussian variables. However, the subject of change point detection was introduced by Shewart [23] where he invented a new statistical tool which nowadays is known as a control chart. A control chart is graph which is used to study changes of a process over time. A control chart has a central horizontal line which represents the average, a horizontal upper line that represents the upper control limit and another for the lower control limit. Using these lines and comparing them to the data it is possible to draw conclusions about the variation and whether it is consistent or not.

Change point detection is closely related to the well-known problem of change point estimation or change point mining according to Aminikhanghahi et al. [2]. Unlike Change point detection (CPD), change point estimation tries to model and interpret known changes in time series rather than identifying that a change has occurred. The focus of change point estimates is to describe the nature and degree of the known change. Change point detection is the task of finding changes in the underlying model of a signal or time series, which can occur at any point. Change point detection methods are divided into two main branches: online methods, that aim to detect changes as soon as they occur in a real-time setting, and offline methods that retrospectively detect changes when all samples are received. This paper will consider the offline methods.

Change point detection is built as a combination of three different factors: Cost function, search method and constraint where constraint relates to the number of change points.

- The cost function is described as a measurement of how homogeneous the signal is, meaning if the signal is homogeneous the value of the function is low, and high if it is heterogeneous.
- The search method is aiming to resolve change point detection problems mentioned in Truong et al. [26], where depending if the number of change points are known, two optimization problems occur which has to be resolved with partly different methods. Each respective method strikes a balance between both computational complexity and accuracy. More generally, a search method is used for finding the best set of change points to optimize the cost function at use.
- The constraint is only applied when the number of change points is unknown and is added as a form of complexity penalty. The constraint directly affects the optimization functions and is therefore very important for the result. With a too low penalty, there will often be too many change points detected including noise. However, with a too high penalty only the most significant changes will be detected if not none.

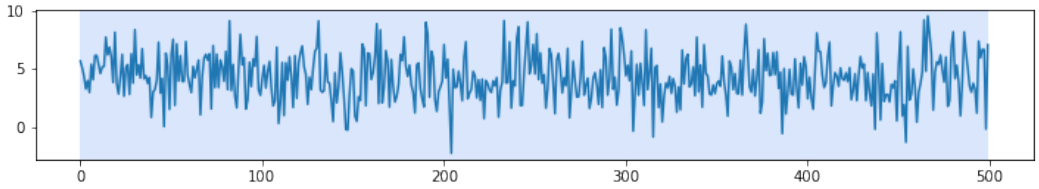

Figure 2.1.1: Synthetic data with no change points, one dimension and low noise(sigma=2).

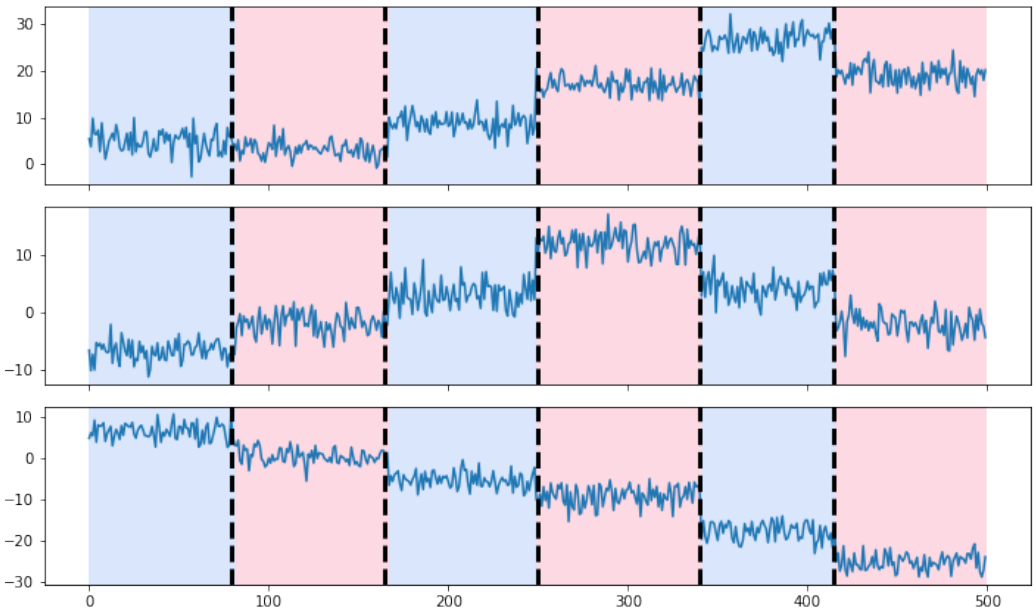

Figure 2.1.2: Synthetic data with five change points (all found), three dimensions and low noise.

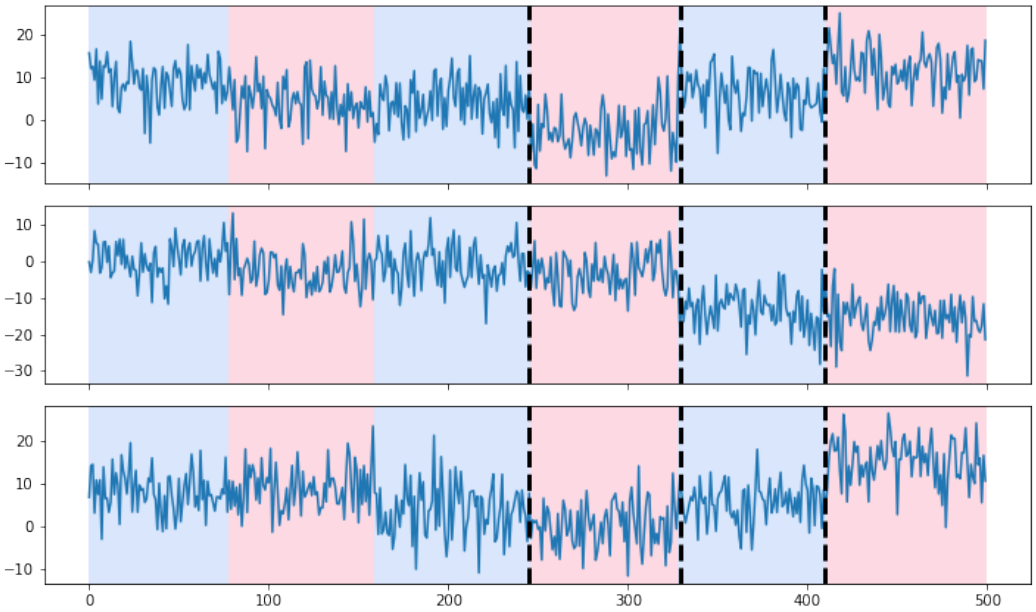

Figure 2.1.3: Synthetic data with five change points, three dimensions and higher noise, causing the algorithm to not find all change points.

Truong et al. [26] is linked with a library (Python specific) called Ruptures. Conveniently, Ruptures will be used for the change point detection which means as a consequence, this paper considers the methods Ruptures contains. Ruptures are also used for the generation of synthetic data testing, which is demonstrated in the figures 2.1.1, 2.1.2, 2.1.3 and 2.1.4. The four figures shows four different stages/settings of the change point detection with the generated synthetic data.

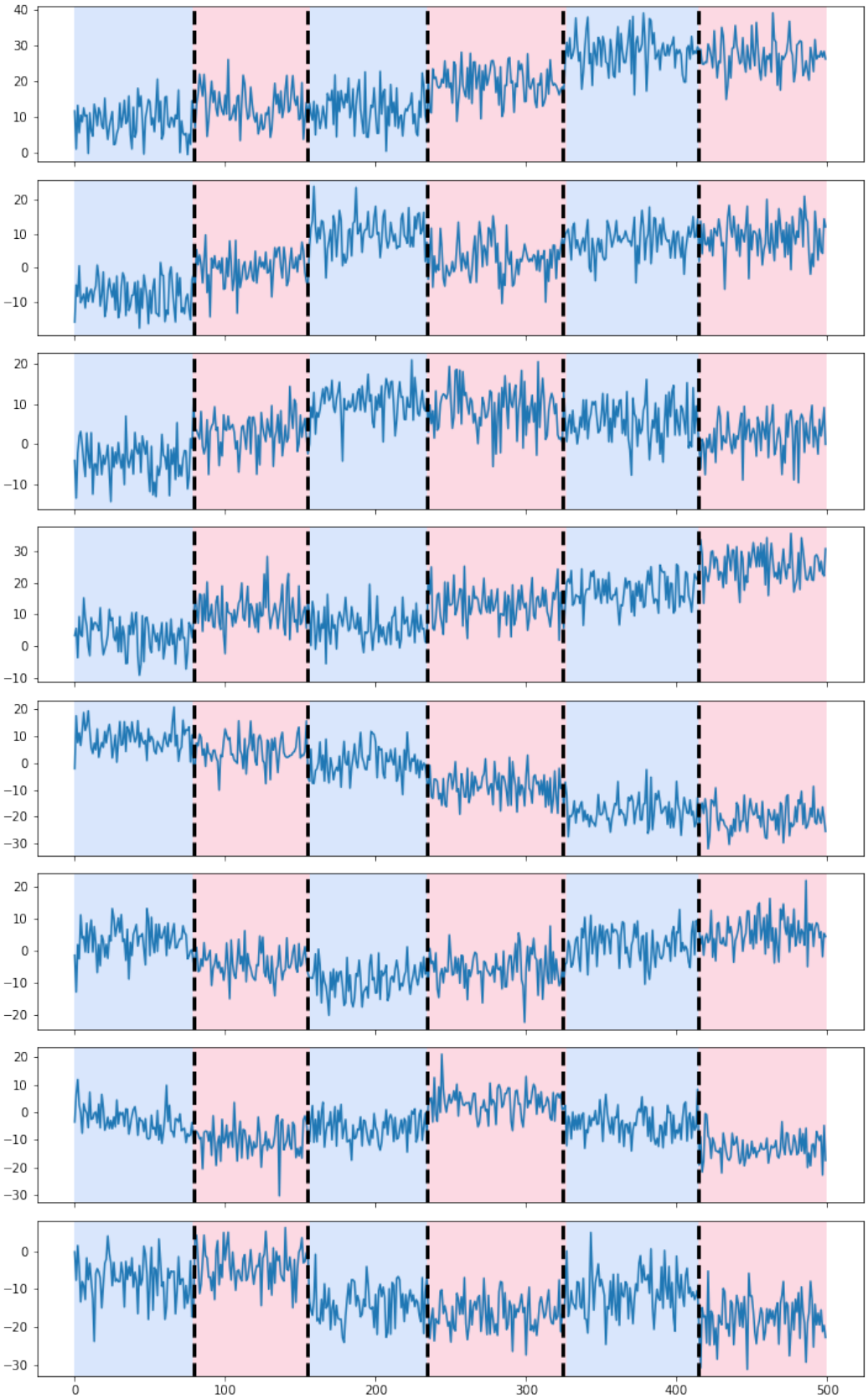

Figure 2.1.4: Synthetic data with five change points, still high noise but increased number of dimensions.

2.1.1 shows a graph of a signal where there are no change points, only a signal with low noise (sigma=2) where the x-axis is time and the y-axis represents a varying signal value. Figure 2.1.2 shows a signal with three dimensions where the noise is still low (sigma=2). The change points are represented by a change in color and the dotted lines represents the change points which has been found by the change point detection algorithm. As shown Figure 2.1.2, all the change points are found. Figure 2.1.3 shows a signal with three dimensions where the noise is increased (sigma=5). As can be observed in the figure, two of the five change points are not detected (since two dotted lines are missing) which is a result of the increased noise. Figure 2.1.4 shows an increase in the number of dimensions as compared to 2.1.3 but the noise is still the same (sigma=5). Evidently, the increase in dimensions helps the algorithm to find all the change points even though there is much noise.

Evaluation of the results is also done through the rupture evaluation metrics [26]. There are three different metrics: ”Hausdorff metric”, ”rand index” and ”precision and recall”. The three metrics are useful for the same reason but the results are presented in different ways. The Hausdorff metric [10] present the evaluation results in the form of the largest distance between the known change points and the Ruptures-detected ones. Rand index [22] produces the evaluation results as a measure of similarity between the change points as a value between 0 and 1. If the known change points and the detected ones are at the exact same spot, the returned value will be 1, and if no change points are detected the value will be 0. The Precision and recall metric returns both the precision and the recall of an estimated segmentation compared with the true segmentation. While precision refers to the percentage of relevant results, recall refers to the percentage of total relevant results correctly identified by the algorithm. Both of these values are returned as a value between 0 and 1, and the goal is for both of them to be as close to 1 as possible. If there is a low precision and a high recall it means that the algorithm finds many change points, although many of them are incorrect. If there is a high precision but a low recall it means just the opposite. The algorithm does not find many change points, but the ones found are correct. Both of these values can be used together to compute the F1-score, i.e. the harmonic mean of precision and recall. The F1-score can be used as a single value measurement of how well the algorithm performs.

2.2 Related Work

There are many interesting works on this particular subject. Most of them concerns the fundamental change point detection aspects. Aminikhanghahi et al. [2] presents a variety of methods for change point detection and algorithms that are commonly applied to the CPD problem, including techniques that are both supervised and unsupervised and based on which outcome is desired. These methods are analyzed and the paper presents their advantages and disadvantages and summarizes some challenges that arise for change point detection. The article also compares online CPD methods vs offline CPD methods as well as additional discussion about scalability,

constraint and performance.

Polunchenko et al. [21] describes some recent techniques of sequential change point detection in 2012 including Quickest change detection, sequential analysis, different procedures and so forth. The paper assumes that the pre- and post-change distributions are known and that the time is discrete. However, the paper is not easy to grasp in the sense that most of the article is equations and the text requires a very deep understanding of the subject.

In the article ”Detecting multiple generalized change-points by isolating single ones” by Anastasiou et al. [4], a new approach is introduced which is ”Isolate-detect (ID)”. This method is based on an isolation technique which does not consider intervals that has more than one change point. It is focused on the estimation of the number and location of multiple generalized change points when there is a lot of noise in a data sequence. This method, according to the tests mentioned and visualized in this article, is at least as accurate as the other state of the art methods and in many cases outperforms them. The paper is very informative and visualizes evaluation tests of methods of interest for this task, as well as compares them.

Aminikhanghahi et al. [3] presents a new algorithm for novel real-time non-parametric change point detection called SEP. SEP makes use of separation distances as a divergence measure to detect changes in time series which are high-dimensional. The usefulness of this method is demonstrated in comparison to the other existing methods. Just as many other papers, this one explains the other existing change point detection methods. However, there is not as much evaluation and comparison to other methods as desired. SEP focuses on high-dimensional data and performs very well amongst the other density ratio based methods because it provides a more sensitive change score, but since the target focus is quite limited, the paper becomes quite limited in the amount of desired information for this task.

Just as Aminikhanghahi et al. [3], Liu et al. [20] also presents a new algorithm, however, it is a couple years older and the paper is even more limited than the previously mentioned. This algorithm is based on non-parametric divergence estimation between two retrospective segments. For the divergence measure it uses Pearson divergence and is estimated by a method of direct density ratio estimation. This paper contains some interesting evaluations but lacks the desired quantity of both text and evaluations.

In the article ”Change-point analysis as a tool to detect abrupt climate variations” by Beaulieu et al. [7], an extension of an existing method known as the informational approach for change point detection is introduced. The purpose is to take into account the presence of auto correlation in the model. The method is focused on climate monitoring which means it is desired to detect shifts soon after they occur. The paper explains change point methodology based on the informational approach in detail which is very good and interesting information. It also shows the flexibility of the approach with applications.

Garreau et al.[15] does not introduce a new algorithm but evaluates an already existing one, namely ”KCP” or ”kernel change-point algorithm” introduced by Arlot et al.[5]. Both of these articles go into depth of KCP and evaluates it in various situations. They are both very informative, a lot of visualization of the evaluations and a lot of examples if further depth in the topic is needed. KCP allows for handling complex data (such as graphs or DNA sequences), multivariate or uni-variate data. The papers also presents the KCP algorithm as one of the best, as well as promoting the fact that it can focus on changes in specific features by considering the appropriate kernel.

The article presented by van den Burg et al. [11] is an article focused on evaluation of already existing change point detection methods rather than creating a new method. This is done by creating a data set (including 37 time series) specifically for evaluating them, analyzing the consistency of five expert human annotators and describe evaluation metrics for measuring algorithm performance in the presence of multiple annotations. At the same time a benchmark study is presented where the different algorithms are evaluated, all of them on each time series of the data set. This article grant a great insight in the different existing change point detection methods.

In the article ”Bayesian online change point detection” by Adams et al. [1], online change point detection is described instead of offline, which is useful in various application areas where modelling and prediction of time series is of importance. While frequentist methods have yielded online filtering and prediction techniques, most Bayesian papers have focused on the retrospective segmentation problem. In this paper, the case where the model parameters before and after the change point are independent is examined, and an online algorithm for exact inference of the most recent change point is derived.

Chen et al. [12] introduced a new approach which was utilizing graphs representing the similarity between observations. This graph-based approach is non-parametric, and is flexible in the sense that it can be applied to any data set with the requirement that there has to be an informative similarity measure on the sample space. The graph based approach requires fewer assumptions in comparison to the parametric but at the same time it makes less use of the data. This means that the graph based approach has a wider applicability but also leads to loss of power in low dimensions. This paper is interesting because of the innovative and different method.

Cho et al. [13] proposes a time series segmentation algorithm based on CUSUM, called ”Sparsified Binary Segmentation” (SBS), which is specifically designed for high-dimensional time series. SBS aggregates CUSUM statistics by just adding those that pass a certain threshold. This step is the ”sparsifying” step which is a key part of the algorithm. It reduces the impact of irrelevant, noisy contributions, which is particularly beneficial in high dimensions. This paper is interesting in the sense that the ”Binary segmentation” algorithm will be used in this work and this article presents a variation of it.

Fryzlewicz et al. [14], just as Cho et al. [13], presents a version of the Binary segmentation algorithm called Wild Binary segmentation (WBS). It is assumed in WBS that the number of change points increases to infinity in correlation to the sample size. Compared to the traditional Binary segmentation, WBS has a much lower range of how close neighbouring change points are allowed to be, as well as the permitted magnitudes of the jump parameter. WBS is not in need of a specific span or window parameter and the computational complexity is not significantly increased without it. It is also illustrated in this work that WBS offers a good performance, and they provide recommended standard values for the parameters. This article is very interesting since it provides a lot of insight into the differences between the state of the art Binary segmentation as well as providing additional methods which could potentially be used in future work.

In Haynes et al. [17], a new method is developed, CROPS, that for a range of penalty values optimises the segmentation of data. For the penalized optimisation, earlier works has been focusing on an exact pruning-based approach which is linear in the number of data points under certain conditions. However, for such an approach, to avoid under/over-fitting it naturally requires a specification of a penalty. CROPS is based on minimizing a penalized cost function and they suspect it is a better approach, for many applications, to segment data than to simply use a single choice of penalty, such as AIC or BIC. The work of Haynes et al. is very much related to this thesis, and although the penalty will be decided with BIC or AIC (whichever is the best fitting) in this thesis, it provides very educational information and could be a factor to be investigated in future work.

In the paper presented by Killick et al. [18], a method is introduced which is directly related to this thesis. The method is called PELT, and is one of the different search methods used in this work. The paper compares the differences between PELT and binary segmentation, and comes to a conclusion that PELT outperforms Binary segmentation, not in the computational cost but in accuracy. The paper also compares the computational cost with other existing methods and find PELT to have one of the lowest. Their focus is on applications where the number of change points increase linearly which the amount of data. Since this method is used in this thesis, the paper is very relevant in the sense that it directly explains how the method works. If, somewhere down the line, information is required of this particular method, it is a great to have such a source of information.

There are other usages to change point detection than time series analysis. Barnett et al. [6] uses change point detection in a correlation network instead, adapted using a computational framework. This framework utilizes the bootstrap, avoiding the computational assumptions that are usually made. The framework is extended for multiple change points, where the data is split into two segments at a found change point and then the process is continued on both subsegments. The method and its generalizable nature is demonstrated by applying it to stock price data as well as fMRI data, where fMRI is a method of measuring brain activity by detecting changes

associated with blood flow. It is important to know about all the different usages of change point detection to understand its importance and how it can be used.

Tartakovsky et al. [24] considers the problem of efficient on-line anomaly detection in computer network traffic. The algorithm proposed in the paper is a novel score-based multi-cyclic detection algorithm, which is based on the Shiryaev-Roberts procedure. It is confirmed in the paper through experiments that the Shiryaev-Roberts procedure performs better than other detection schemes. Also, the computational complexity of their method is very low and it is easy to implement. This paper is somewhat related to this thesis work and is therefor relatively interesting and educational.

Lindquist et al. [19] introduces a new approach to the subject of change point detection, HEWMA. It can be used for making inferences about both individual and group fMRI activity. HEWMA is an extension of another time series analysis methods called EWMA, which is used to multisubject data. What is interesting about this paper is the fact that even though this method was developed for the fMRI data analysis, it may, according to the paper, still be useful for other areas in the detection of deviation from a baseline state in any type of time series data. These areas could be for example longitudinal studies of brain structure or PET activity.

2.3 Chapter Summary

This chapter explains the fundamental parts of change point detection as well as introduces and discusses many different existing methods and functions and how they are of interest to this thesis work. A short background to change point detection is presented and a description (both visual and textual) of how it works.

Chapter 3

Data Characterization and Preliminaries

This chapter covers the initial analysis of the data, as well as the initial analysis of the features and feature manipulations.

3.1 Feature Analysis

3.1.1 Data Source

The source of the data used in this thesis work comes from the equipment used by Sandvine, and is received as data logs. The data logs received varies in size, and as the work progresses, the bigger the logs are which is because of the simple reason that we initially do not require larger data sets. The logs themselves includes many system IDs, where each system ID represent one of Sandvines equipment(or one system). The logs are then converted into a data frame to be analyzed and used for the change point detection.

3.1.2 Data Frame Analysis

Tables 3.1.1 and 3.1.2 represents examples of how a data frame that is used in this work looks like, although in a much smaller scale. A data frame varies in size depending on the time span of the data logs, the number of features and the number of different systems. The first column is the System ID feature which states the ID of a specific system used by Sandvine, as mentioned in Section 3.1.1. There is a very large amount of different System IDs in total, however, every individual system ID may be present in a large amount rows in the data frame. The timestamp feature represents the time of when a specific row of data was created, and it is present in what is called a ”Timestamp format”. The timestamp is the most crucial feature of change point detection together with the System ID, they are what you would call primary keys in a regular database. On each row, every data field on that specific row is connected to

that timestamp. Furthermore, the signature version feature displays the version of the signature database that is operating at a specific point in time. A signature database holds a set of characteristics such as IP address, port numbers, TCP flags etc. which are used to classify different types of network activity. In line with the signature version, the firmware version displays the current version of the firmware at a specific point in time, and the system model displays the system model. The rest of the columns (features) are various different generated data from the systems at hand. The few examples in Table 3.1.2 displays a big variation of values and represents what the rest of the data frame looks like. Each row of the data frame is directly connected to the system ID, timestamp etc. on that specific row. Furthermore, each of the features in Table 3.1.1 are constant for a system ID, whereas the rest are varying data. The features ending with ”rate” represents incoming or outgoing packets over a five second period measured each hour, whereas those ending with ”tot” represent the total amount of packets/drops/power-saves etc.

System ID	Timestamp	Signature version	Firmware version	System Model	...
000BAB4D7BD9	2020-12-05 23:51:38	S-440.baseline-2018-10-22	16.2.10.2	PL8720-INT-FWA6500	...
000BAB4D7BD9	2020-12-06 00:51:38	S-440.baseline-2018-10-22	16.2.10.2	PL8720-INT-FWA6500	...
000BAB4D7BD9	2020-12-06 01:51:38	S-440.baseline-2018-10-22	16.2.10.2	PL8720-INT-FWA6500	...
E4434BF36FB0	2020-12-21 00:51:00	S-552.baseline-2020-11-13	21.60.06.1	iQ42300-INT-DELLR740	...
E4434BF36FB0	2020-12-21 01:51:01	S-552.baseline-2020-11-13	21.60.06.1	iQ42300-INT-DELLR740	...
E4434BF36FB0	2020-12-21 02:51:01	S-552.baseline-2020-11-13	21.60.06.1	iQ42300-INT-DELLR740	...

Table 3.1.1: Six rows of data from the data frame with the columns: System ID, Timestamp, Signature version, Firmware version and System model.

System ID	PacketProcess RXPackets rate	PacketProcess RXPackets tot	PacketProcess CPULoad tot	PacketProcess CPUIrqs rate	PacketProcess CPUIrqs tot
000BAB4D7BD9	743236	2487667180435	15	29337	102562805220
000BAB4D7BD9	434670	2489659414734	10	29294	102667430737
000BAB4D7BD9	355586	2491053383420	8	29067	102771845924
E4434BF36FB0	0	0	0	0	0
E4434BF36FB0	2697144	6717645673	6	132494	307855867
E4434BF36FB0	2827941	26508265792	6	131431	1259870801

Table 3.1.2: Six rows of data from the data frame with variating values on each feature for a specific system ID.

3.2 Graph Generation

To get an initial overview of the data, graphs of the features are generated with different characteristics. The first one to be generated is Figure 3.2.1. This figure contains a total of nine system IDs and each system ID is plotted with the rate of packets against time. As can be observed in the figure, some of the lines are showing a very strange behaviour. Systems that has large segments with straight lines like that are deemed to be testing systems and are not relevant for the change point detection in this thesis work. The Feature PacketProcess_RXPackets_rate represents the packet rate of the system for

each timestamp. There are timestamps every hour between 2020-12-05 and 2021-01-05 for this particular data set. Figure 3.2.2 is generated with the exact same feature

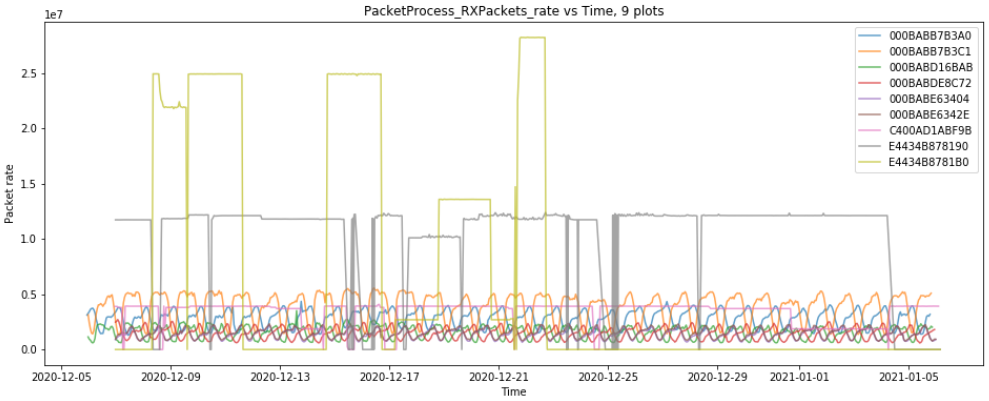

Figure 3.2.1: Illustration of the packet rate vs time for all system IDs (including those with bad values).

as Figure 3.2.1, but now the system IDs with bad values are removed. This is achieved through removing the system IDs that are considered testing systems. Not only does the graph look better but also, it is much better scaled which means that it is easier to analyze the variation and notice change points.

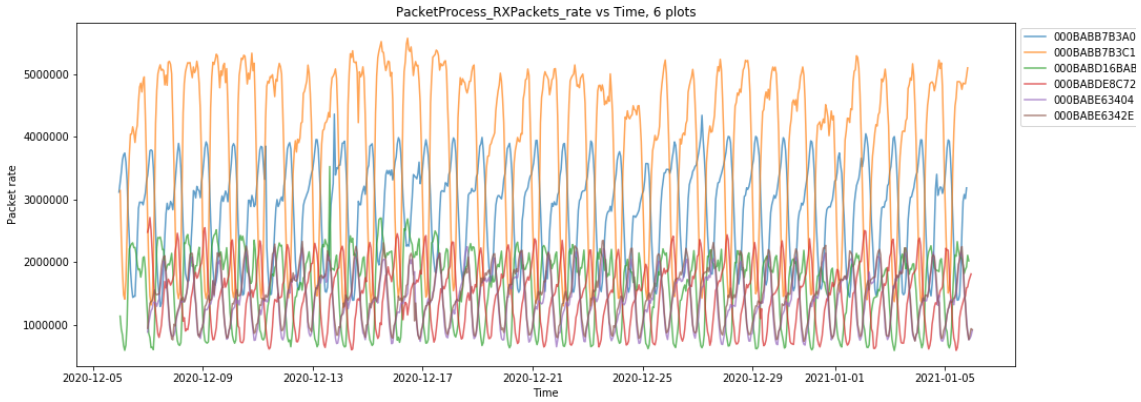

Figure 3.2.2: Illustration of the packet rate vs time, where bad system IDs has been removed. It is clear that the variation is very different for the different system IDs.

The next figure, Figure 3.2.3, represents the total amount of packets for the specific system for each specific timestamp. The drops down to zero that can be seen in the figure is due to a shut down/reboot of the system. The data as shown in this figure (Figure 3.2.3) is not particularly useful, in the sense that the variation between each timestamp is very hard to keep track of considering the values are monotonically increasing.

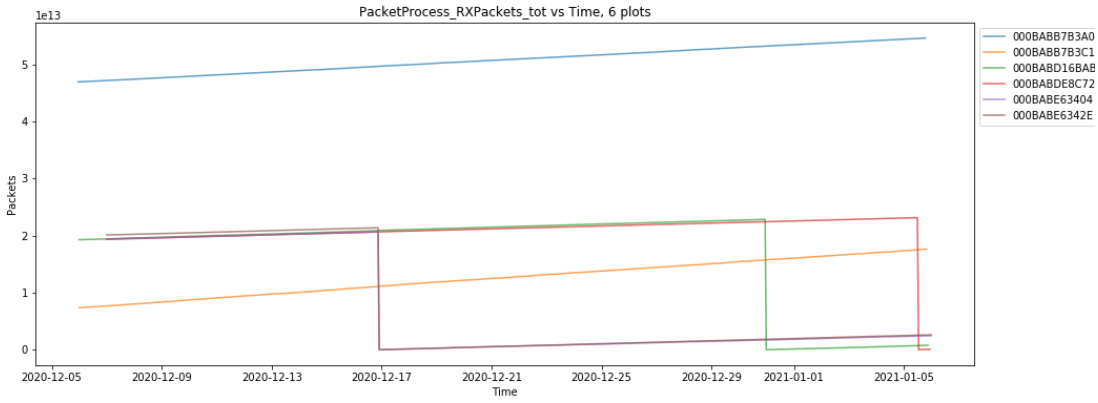

Figure 3.2.3: Illustration of the total amount of packets vs time, where the values are monotonically increasing. The variation is impossible to see.

Figure 3.2.4 is created to be able to see the variation of total packets for each timestamp. This is done through a method called that is referred to as the ”diff method”, by choosing the value for each timestamp of the ”PacketProcess_RXPackets_tot” subtracted by the value of the same feature but one earlier timestamp. This creates a new feature which represents the change in the total amount of packets at the system between two timestamps. This new feature is necessary for further analysing, but the ”diff” method is not applicable on every feature, only the ones with monotonically increasing values.

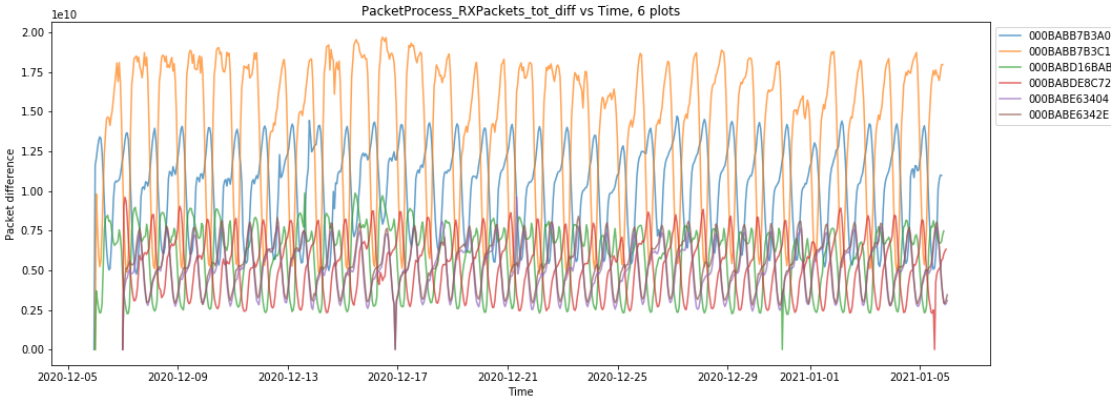

Figure 3.2.4: Illustration of the total amount of packets vs time, where the ”diff” method causes a much more clear picture of the variation for each system ID.

Figure 3.2.5 shows how much of the total load of the CPU (in %) is used for each timestamp, meaning how many percent of the total capacity of the CPU used over time. This feature is part of another class of features, namely those that are not directly related to packet counts. As can be observed, this feature does not have monotonically increasing values and will therefore not be subject to processing with the ”diff” method.

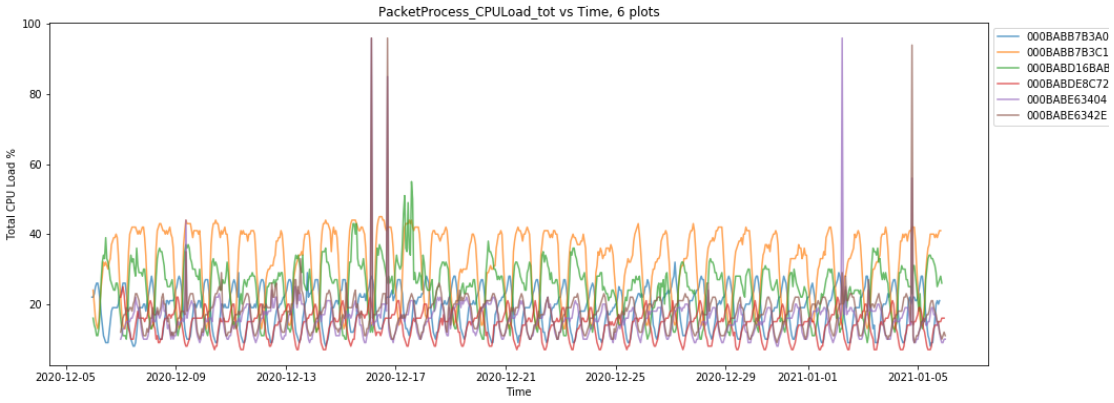

Figure 3.2.5: Illustration of how much of the total load of the CPU (in %) is used for each system ID at each timestamp.

The next Figure 3.2.6 is a combination of two features, where the values of ”PacketProcess_RXPackets_rate” is divided by the values of ”PacketProcess_CPULoad_tot which creates a feature that represents the packet rate of the system per CPU percentage. This is called ”load normalizing” and will be used a lot in further investigations. Each value of the RXpacket rate of each timestamp is divided by the corresponding value of the total CPU load for that timestamp. Also, there will occur certain situations where for example the system of interest is down for a specific timestamp causing the value to be 0, and for those cases the value of the denominator is set to 1.

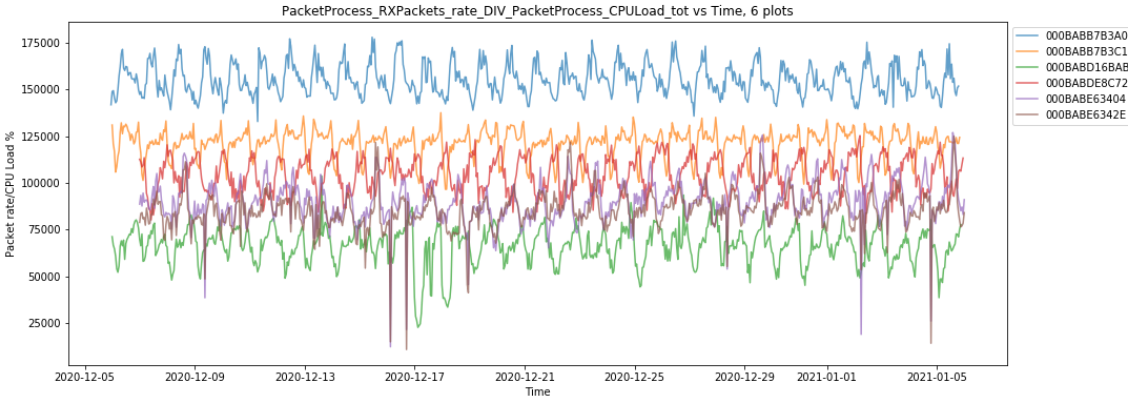

Figure 3.2.6: Illustration of the packet rate, load normalized by the total CPU load for each system ID.

Just as the previous figure, Figure 3.2.6, 3.2.7 is created through division of features. ”PacketProcess_RXPackets_tot_diff” is divided by the ”PacketProcess_CPULoad_tot feature. This creates a new feature which represents the difference in total packets per CPU load percentage, between each timestamp. It is observable that figures 3.2.6 and 3.2.7 are very similar, which is very much intended. It is clear that the ”diff” method is successfully created and does exactly what it should.

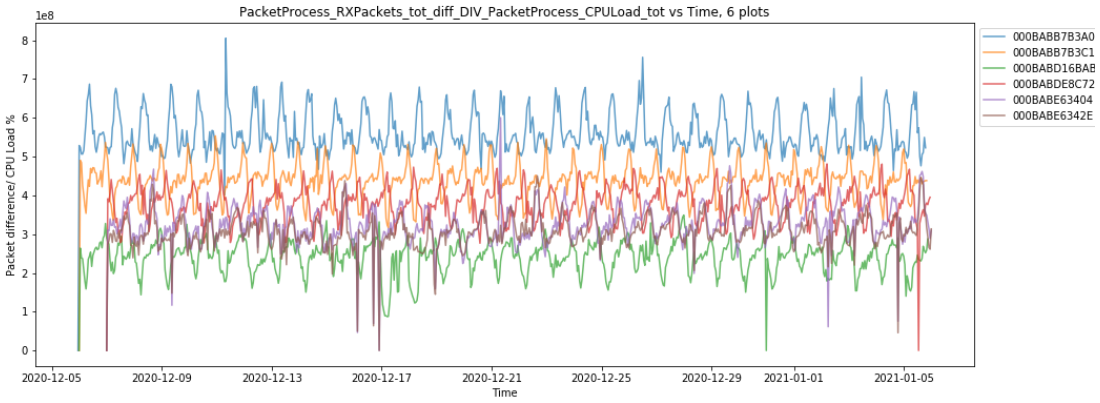

Figure 3.2.7: Illustration of the total packet count, load normalized by the total CPU load for each system ID.

The last figure of this section, Figure 3.2.8, shows the memory available for a system over time. It is neither monotonically increasing nor varying.

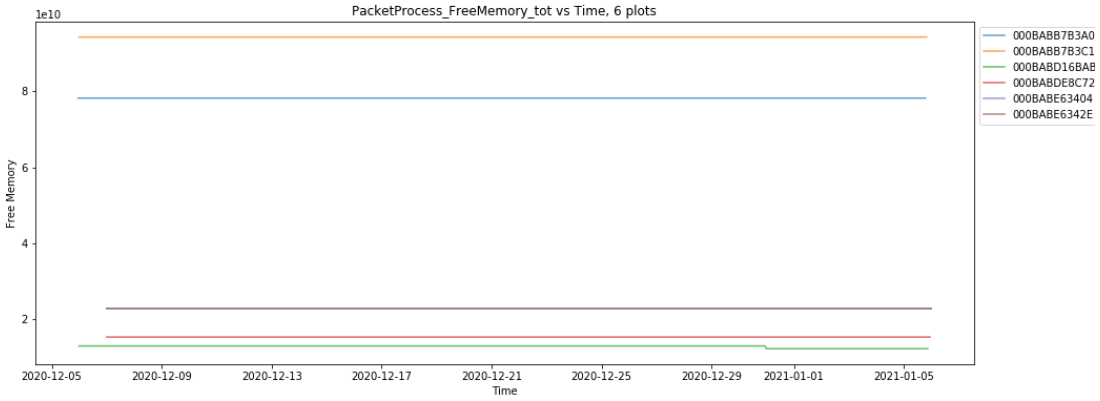

Figure 3.2.8: Illustration of the memory available for each system over time.

The variation is of upmost importance for this work, not the daily variation but the variation caused by changes in signature database version or firmware version, or potential parameter changes. To be able to spot the reasons for the changes in variations, a function is created to add vertical lines, at the timestamps where either firmware version updates or signature version updates occurs, to the plots. The goal is to be able to explore the extent it is possible to visually see when updates affects the

system, and the vertical lines provide great assistance when trying to observe change points which occurs due to these updates/changes.

3.3 Initial Investigations

The initial investigations revolved around purely observing, but observing the previously shown graphs is relatively difficult considering the variation of scaling per system ID and per feature. To simplify observations, normalization of the graphs are applied making the y-axis values scale from 0 to 1. This way, regardless of the exact value of a certain feature over time, it is easier to see the exact variation. Figure 3.3.1 shows an example of this, where the feature ”PacketProcess_RXPackets_tot_diff_DIV_PacketProcess_CPULoad_tot” is in focus. The 0 to 1 normalization method is mostly (if not only) used for observational studies.

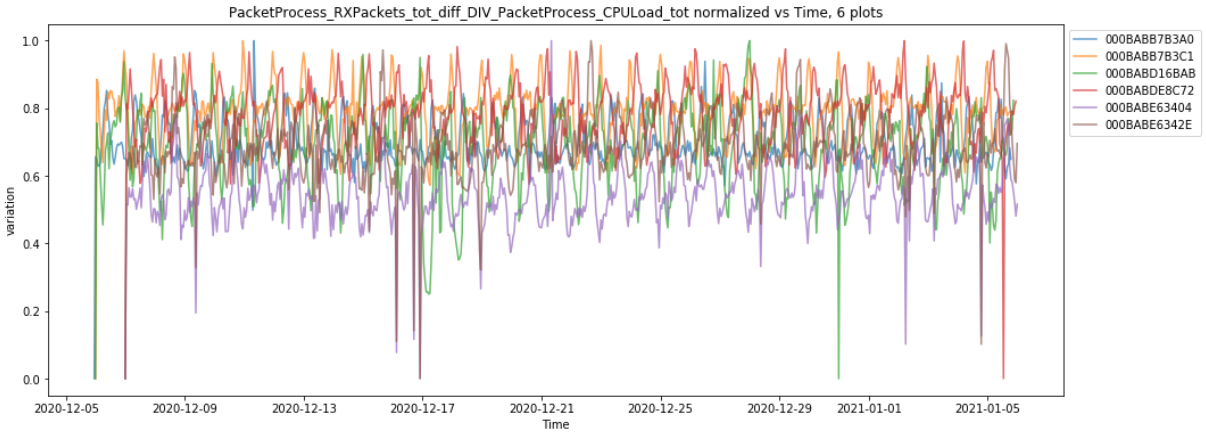

Figure 3.3.1: Illustration of the packet count vs time, diff method applied, load normalized by the CPU load and applied normalization of the values, scaling from 0-1, for each system ID.

It is noticeable that some graphs stand out in certain ways. For example, the graph of available memory, Figure 3.2.8 in Section 3.1, there are no variations or spikes at all as opposed to most of the other features. This is only due to the fact that Sandvine pre-allocates memory for ”PacketProcess”. Moreover, the spikes the can be seen in many of the graphs are there for different reasons, with the main reason being system reboots/shut downs. To solve this issue(or part of it), before converting the next data logs into data frames, the data logs are filtered to only contain good system IDs, meaning system IDs that does not contain sections with no values (Nan-values).

3.3.1 Moving Average

For most of the features, values vary according to the day/night cycle, meaning that there is more activity during the day because there are more people online. The cycles

make any potential change point harder to visually detect, therefore additional features are created using a ”moving average” function. The moving average function creates an average value for a time span of choice and sets that value in the time stamp which corresponds to the last time stamp of the chosen span. For example, for the series [(1,1h),(2,2h),(3,3h),(4,4h),(5,5h),(6,6h)] and the time span is 3h, the resulting moving average calculation would be [(nan),(nan),(2,3h),(3,4h),(4,5h),(5,6h)]. These new features if plotted, creates a graph which vary not as a consequence of the day cycle, but of the actual variation. A demonstration of this is shown in the Figure 3.3.2.

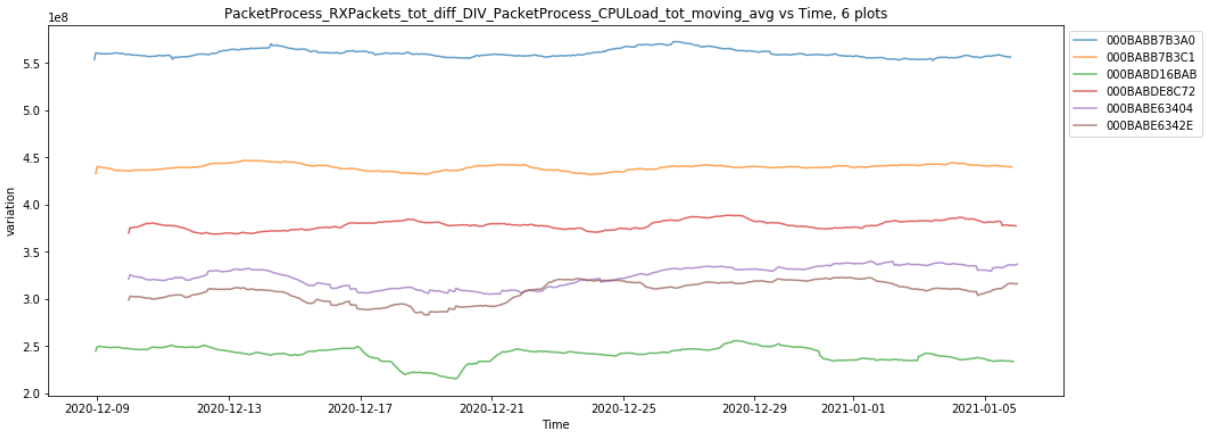

Figure 3.3.2: Illustration of the packet count, diff method applied, load normalized by the CPU load and the moving average method applied with 72h time span.

The newly produced graph, Figure 3.3.2, is substantially better for finding change points by the eye, although there are additional visualized data to be added. As mentioned in Section 3.1, vertical lines at the time stamp where either firmware updates or signature updates occur can be generated. Such lines are a huge benefit when looking for change points and are therefore added to the graph. This is displayed in Figure 3.3.3. However, as can be observed in the figure, there are no firmware changes for the specific system IDs, only signature changes.

3.3.2 A Change of Features

The plots are looking good and the vertical lines provide great assistance. However, there are many more features to consider and further investigations and observations are required, resulting in additional feature creations from the remainder of the data received from Sandvine at the time. Taking a look at the plots of the few features, most of those where the feature name end with ”tot” are monotonically increasing, meaning they increase in a low but steady pace and investigating them with regards to variation is pointless. Therefore, just as introduced in Section 3.2, ”diff” features are created to be able to further analyze the features. Also, from analyzing the many graphs, it is clear that the number of firmware updates is very low. The resulting graphs are similar to Figure 3.3.4. The vertical lines represents changes in signature or firmware,

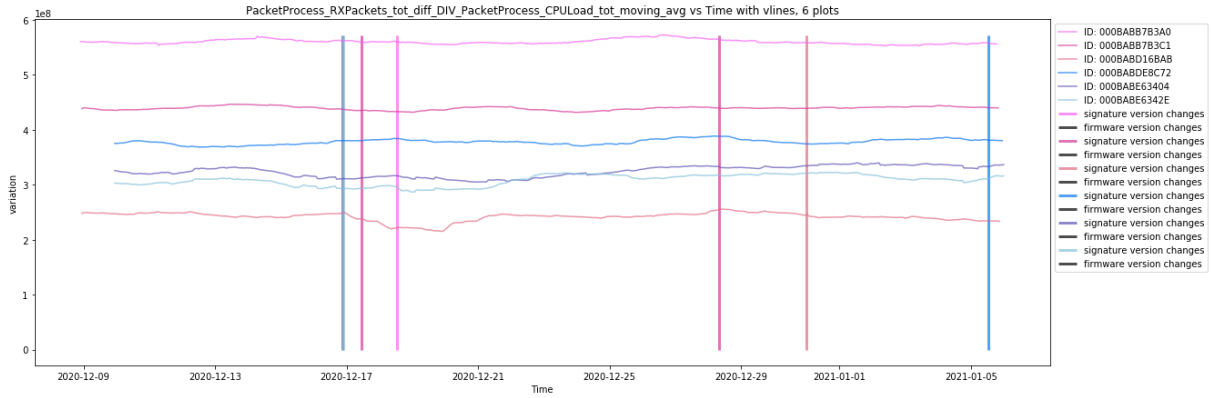

Figure 3.3.3: Illustration of the packet count, diff method applied, load normalized by the CPU load and the moving average method applied. The vertical lines represent change points (in this case only signature changes).

and each color of the plot is represented by a system ID. A specific color of the plot directly corresponds to the same color on the vertical lines.

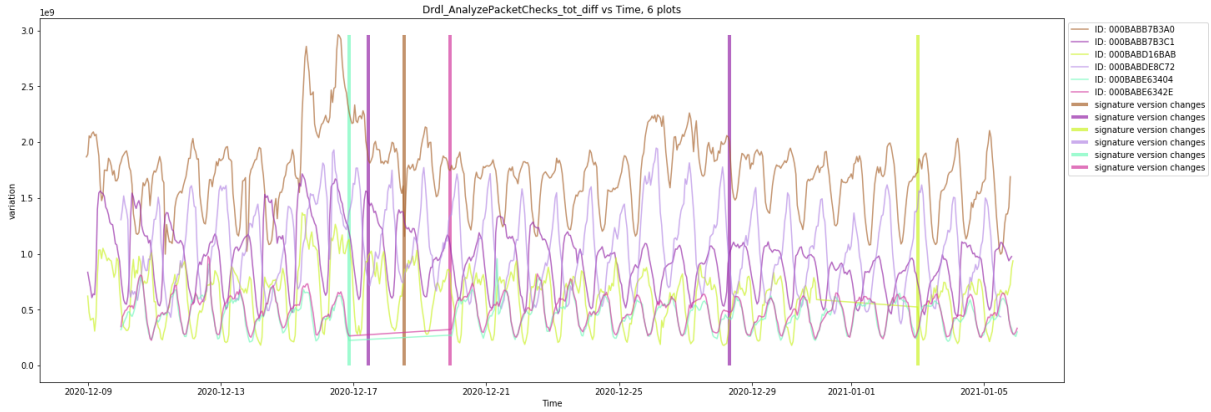

Figure 3.3.4: Illustration of the feature ”AnalyzePacketChecks_tot_diff” plotted against time with vertical lines representing the change points for the different system IDs.

The features created using the ”diff”-method now looks exactly like those whose names end with ”rate”, meaning that they have the same characteristics which means all features that are to be analyzed can be treated equally. The features analyzed up to this change of features has been load normalized with the CPU load feature, however, the next step is to change the load normalization feature. Additional creation of new features is done by taking each one of the approximately 30 features and divide them (load normalize) with the feature ”Connections_inbound_plus_outbound_tot”, resulting in graphs where the total variation is reduced. Although, as can be observed in the previous Figure 3.3.4, the day/night-cycle is still present and is handled by the ”moving average” method mentioned earlier in this section. After combining the load normalization and moving average, the graphs looks like Figure 3.3.5.

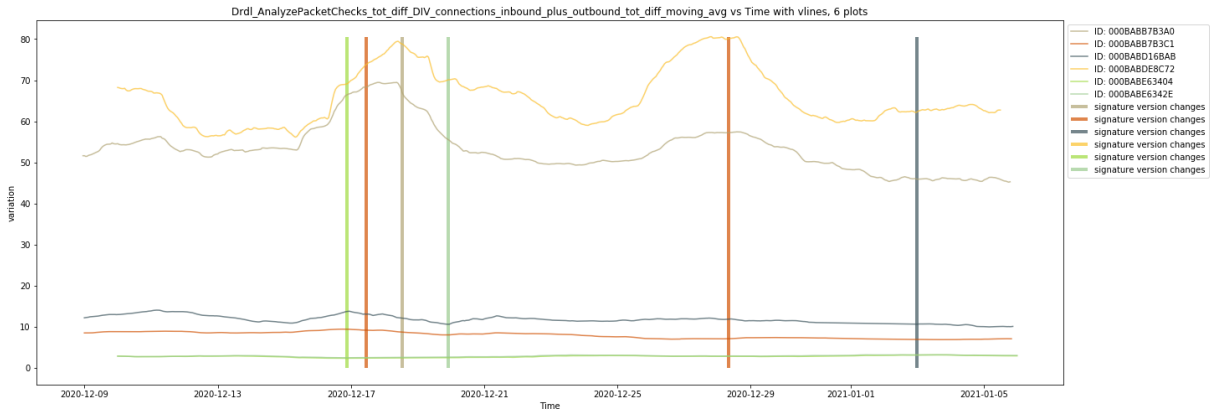

Figure 3.3.5: Illustration of the feature ”AnalyzePacketChecks_tot_diff”, load normalized by total number of connections, moving average method applied, plotted against time with vertical lines representing the change points for the different system IDs.

3.4 Chapter summary

In this chapter we explain the source of the data logs as well as the data itself. Further, we analyze the data through graphs and tables, and some initial feature manipulation is introduced along with changing the set of features and investigating the new set.

Chapter 4

Experimental design survey

This chapter contains different settings, feature manipulation and parameters for Ruptures evaluation experiments with the experiments plotted as heatmaps to be analyzed. Also, the different parts of the change point detection through Ruptures is explained more thoroughly as well as the evaluation metrics of the change point detection results.

4.1 Overview of Ruptures

Possessing about 30 features where the scaling, structure and plotting is relatively satisfactory, there is now need for further investigations with Ruptures. As mentioned in Section 1.6.1, Ruptures provides methods for analyzing signals and most of these methods are used, including a variety of the different search methods, cost functions, signal creation and evaluation metrics. The signal creation is somewhat obviously not to be considered since the signals used will be the ones from the data logs sent from Sandvine. To test all of the different methods and functions, multiple nested for loops has to be created to run through every different combination. However, to be able to test with different parameters and some specific combination within a reasonable amount of time, not every method is tested directly, but a selected few. Sections 4.1.1, 4.1.2 and 4.1.3 will explain more about the various existing methods, functions and metrics.

4.1.1 Search methods

- ”Dynamic Programming” is a method which computes the cost of all subsequences of a signal, where the number of computed costs depend on the number of change points and the number of samples. This means that the number of change points has to be defined beforehand.
- ”Pelt” tries to enumerate as many partitions as possible, and since it is impossible to enumerate all of them for a given signal, Pelt relies on a pruning rule. The goal

of the pruning rule is to reduce the computational cost while still retaining the capability of finding the optimal segmentation. For a more detailed description, see Killick et al. [18].

- Binary segmentation is a sequential approach which is used to perform fast signal segmentation. In the first step, it detects a change point in the complete signal. Secondly, it splits the series around that change point and the operation is repeated on both sub signals. A visual illustration is presented in Figure 4.1.1. Binseg comes with benefits such as a low time complexity and the perk of being able to extend any of the existing change point detecting method to detect several change points.
- Bottom-up segmentation is a sequential approach which starts with detecting several change points and progressively removes the less significant ones. This method can also extend any of the existing change point detecting method to detect several change points. A visual illustration can be observed in Figure 4.1.2.
- The window based algorithm, i.e. Window sliding segmentation, uses two windows which both slide across the data stream. It first converts the signal to a discrepancy curve and then a sequential peak search is performed on the curve to find change points. The window-size should be pre-defined, otherwise it defaults to 100 samples. A visual illustration is presented in Figure 4.1.3.

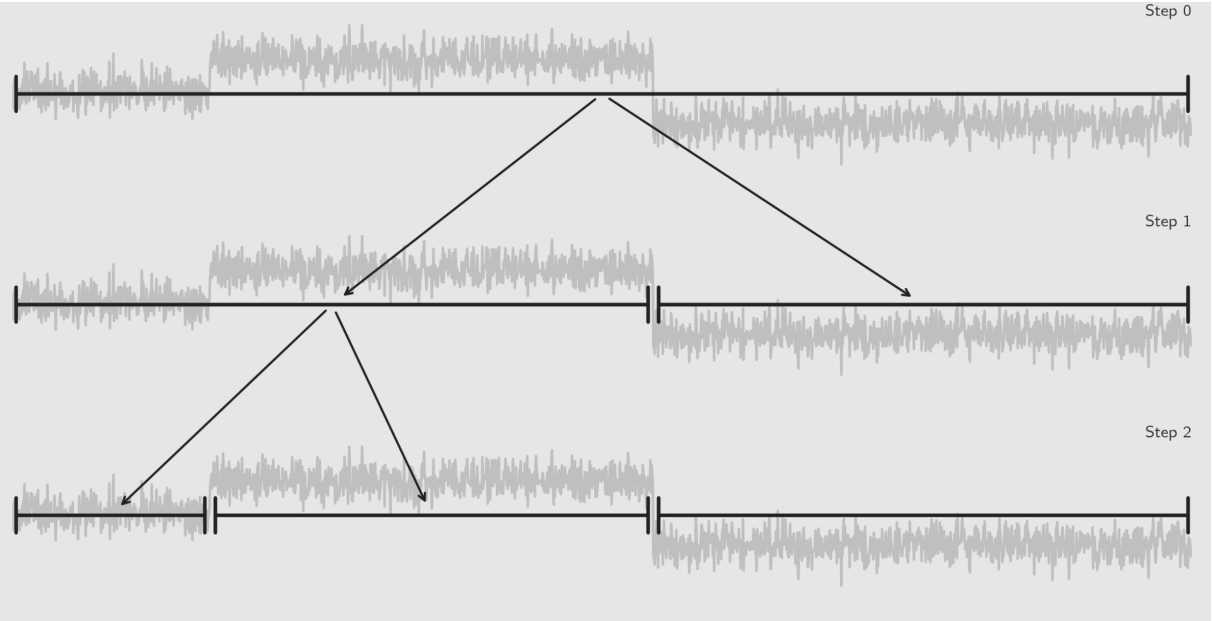

Figure 4.1.1: Illustration of the Binary segmentation search method. Image from [8]

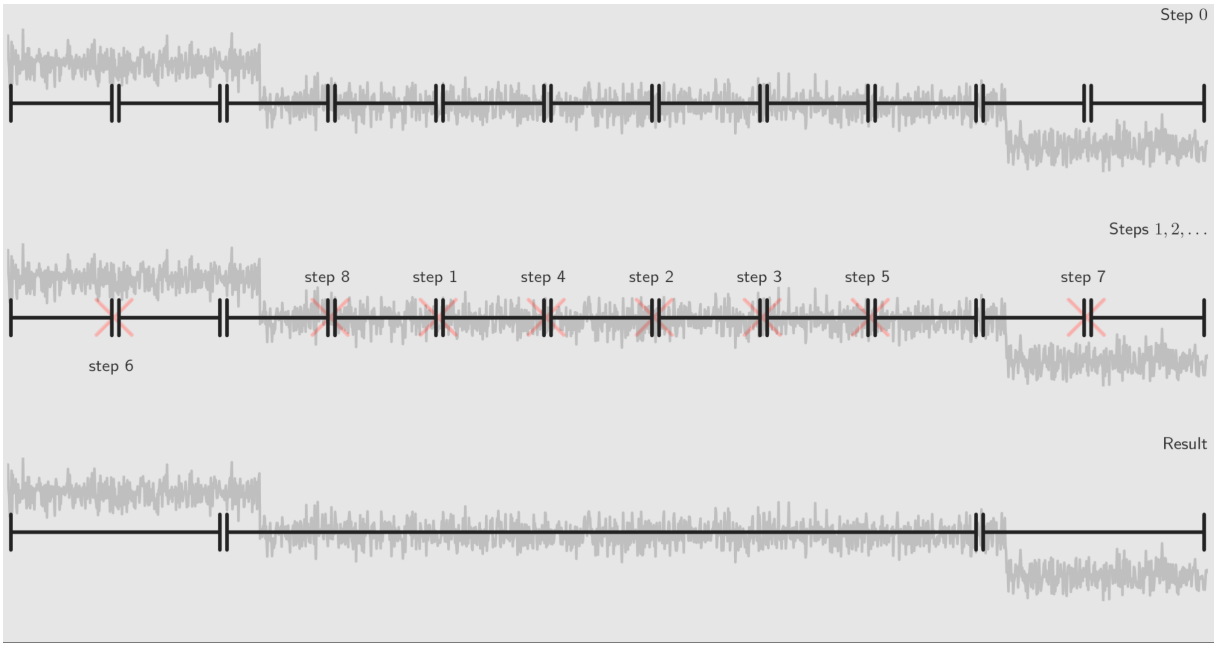

Figure 4.1.2: Illustration of the Bottom-up segmentation search method. Image from [9]

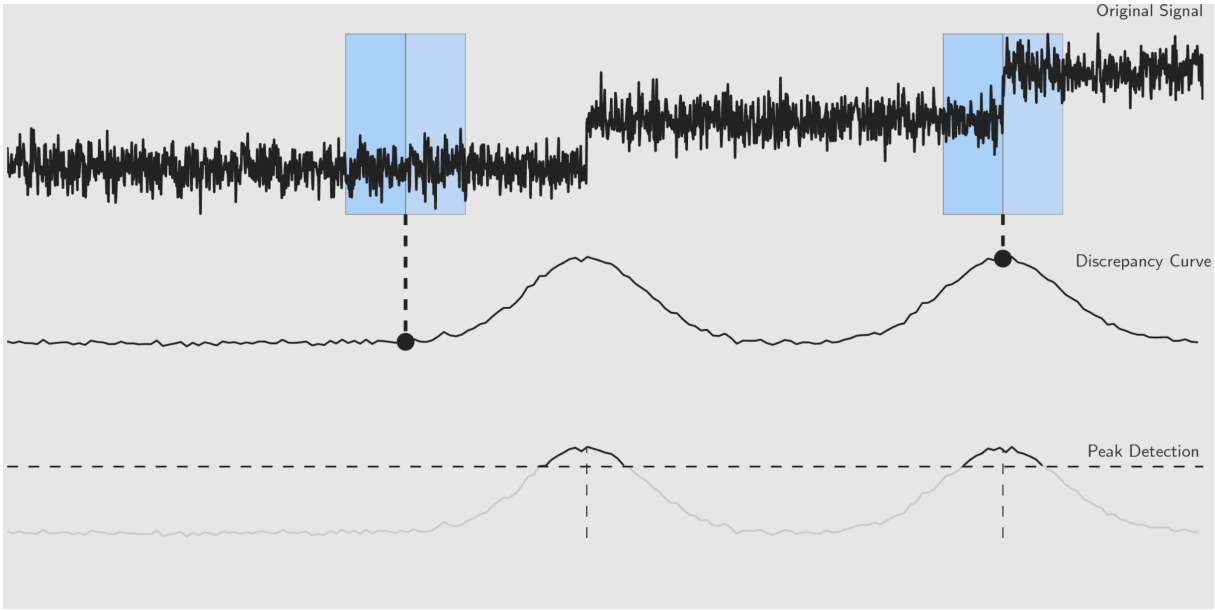

Figure 4.1.3: Illustration of the Window sliding search method. Image from [28]

These methods has a few parameters that needs to be described. First and foremost there is the ”min_size” parameter which is important for helping the algorithm for cases where assumptions about distance between change points can be made. The min_size value controls the minimum distance between change points, meaning that if min_size is set to the value 20, all the Ruptures-detected change points are at least 20 samples apart (default is 10). Ultimately, the min_size parameter facilitates in the algorithms search considering it does not have to look for change points within the range of the min_size. See Section 4.1.4 for an illustration.

Another parameter is ”jump”, which also facilitates the search of change points. For example, if the value of ”jump” is set to x, only changes at x, 2*x, 3*x etc. are considered. However, the jump parameter is not used(in the sense that it is set to the value 1 so that no search restriction is performed)in the initial Ruptures testing considering the change points often occurs in a very irregular pattern.

Last but not least there is the penalty parameter issue which plays a crucial part in the Ruptures prediction algorithm. In the case where you do not know the amount of possible change points (as mentioned in Section 2.1), you can specify the penalty by setting the ”pen” parameter to a value calculated with certain formulas and depending on said formula, too many or too few change points may be detected if the formula is a little off.

However, when the number of possible change points are known, the ”n_bkps” parameter (replacing the ”pen” parameter) can be set to the number of actual change points i.e. a combination of the number of firmware updates and signature updates, to achieve the most ideal setting for the parameter instead of using a formula for calculating a penalty. Although, setting the penalty with ”n_bkps” may be good for testing but can not be used in the final analysis, since the idea is to find the change points without knowing how many there are exactly. Initially, ”pen” parameter is set to 100 since no investigations have been made yet on penalty functions.

The search methods used for the initial testing are Pelt and Binary segmentation.

4.1.2 Cost functions

The different cost functions that exists within Ruptures are the following:

1. Least absolute deviation (Costl1)
2. Least squared deviation (Costl2)
3. Gaussian process change (CostNormal)
4. Kernelized mean change (CostRbf)
5. Kernelized mean change (CostCosine)
6. Linear model change (CostLinear)
7. Continuous linear change (CostCLinear)
8. Rank-based cost function (CostRank)
9. Change detection with a Mahalanobis-type metric (CostMl)
10. Autoregressive model change (CostAR)

The cost functions used for the initial experiments are numbers 1,2,3,4,8 from Table 4.1.2. These functions are chosen solely because of the fact that reduced running time of the tests is of interest. Eventually, more of the cost functions will be tested.

4.1.3 Evaluation metrics

There are three different evaluation metrics in Ruptures. First and foremost, there are the ”Precision and Recall” metrics. The two metrics returns two different values. These two values are defined in two somewhat similar ways, where

$$Precision = TruePositives/(TruePositives + FalsePositives)$$

and

$$Recall = TruePositives/(TruePositives + FalseNegatives)$$

In order to understand these equations, it is essential to know what the factors mean for this particular subject. A true positive is an outcome where the CPD method correctly predicts an existing change point. Similarly, a true negative is an outcome where the CPD method does not predict a change point where it should not be one. A false positive is an outcome where the CPD method incorrectly predicts a change point, and a false negative is an outcome where the CPD method does not predict a change point where it should be one.

The exact description of the two metrics in words are that Precision is the percentage of the algorithms predicted change points that are true positives, and Recall is the percentage of the true positives that are correctly classified by the algorithm. There is also a trade-off between these values. If there is a high Recall, the generated results are many and not very correct, hence lowering the Precision. The other way around, if there is a high precision, there is often very few predicted results. However, to easier be able to maximize the trade-off between these two values, there is a simpler metric to consider: the F1-score. The F1-score is simply the harmonic mean of Precision and Recall, and is defined as:

$$F1Score = 2 * (Precision * Recall/Precision + Recall)$$

Secondly, there is the Hausdorff distance metric [10]. Hausdorff distance is not quite as complicated, but is still an important evaluation metric. It measures the distance between two subsets, meaning in our case the distance between actual change points and predicted change points. Informally speaking, it is the greatest of all distances between any one point in one subset to the closest point in the other subset. In the ideal scenario, the Hausdorff distance should be really low. Lastly, there is the Rand Index metric [22]. Rand Index is a measure of the similarity between two segmentations. The returned value is a number between 0 and 1, if the value is 1 it means that, in our case, the predicted and the actual change points are the same and the reverse for the value 0. All of these three evaluation metrics are considered in the testing. The evaluation metrics also have a certain parameter to consider, namely the ”margin” parameter. Margin determines how precise the algorithm should be. For example, if the algorithm finds a change point at position 200, but the actual change point is at position 192, the evaluation metric does not count that as a true positive unless margin is 8 or higher. See Section 4.1.4 for a more detailed description.

4.1.4 Change point detection illustrations

The Figure 4.1.4 shows an example of what a Ruptures prediction looks like where the four dashed vertical lines represent the positions of the detected change points by Ruptures and the exact positions of the shifts in colors represent the firmware/signature updates for a specific system ID. From this figure the conclusion is that Ruptures detected four change points at the correct locations. In Figure 4.1.5,

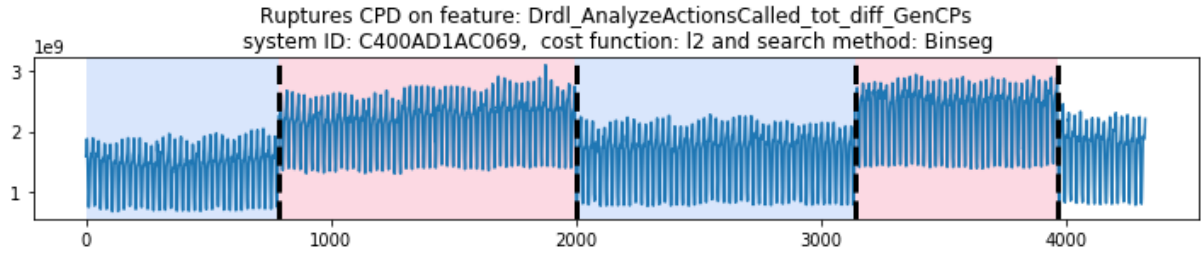

Figure 4.1.4: Illustration of what a Ruptures change point detection output looks like. The dashed lines are Ruptures-detected change points and the change in colors are actual change points.

margin and min_size are visualized. For both graphs, the actual change points are at the same position, the varying factors are margin and min_size (where min_size is unrealistically high for illustrative purposes in the lower graph) which affects the position of some detected change points. In the upper graph, the margin is 36 and min_size is 48 and in the lower graph margin is 2 and min_size is 400. In the upper graph, all actual change points are considered found even though they are not exactly correct, because of the margin parameter being larger than 25 and the distance between two of the actual change points (first and forth) and the location of the detected change points are 25. As for the lower one, min_size at 400 does not allow the algorithm to find a change point at the second actual change points position, instead it is found 123 positions away. Furthermore, since the margin parameter is 2, only two of all the change point are true positives i.e. change points found at the correct location by Ruptures. To make it clear, the last change point in both ”Actual sig CPs” and ”Ruptures detected CPs” are the absolute last position in the data and does not affect the ruptures evaluation at all. It is neither counted as a true positive nor anything else.

Using the chosen features, methods and functions, a new data frame is created. The function used to create it loops through all the different combinations and a new column is added which contains the average value for each evaluation metric for each combination of feature, search method and cost function over all the 25 system IDs of the current data frame. Lastly, the average values over all system IDs are plotted in a heatmap to much easier be able to see which combination is the most efficient for the held data. One heatmap is created for each evaluation method. The parameter settings are: min_size = 36, margin = 24, pen = 100, jump = 1. A heatmap of the experiment for the F1-value of the Precision and Recall evaluation metric is shown in Figure 4.1.6, where the last part of the names of the features, I.E ”_DIV_connections_inbound_plus_outbound_tot_diff_moving_avg” is cut out just

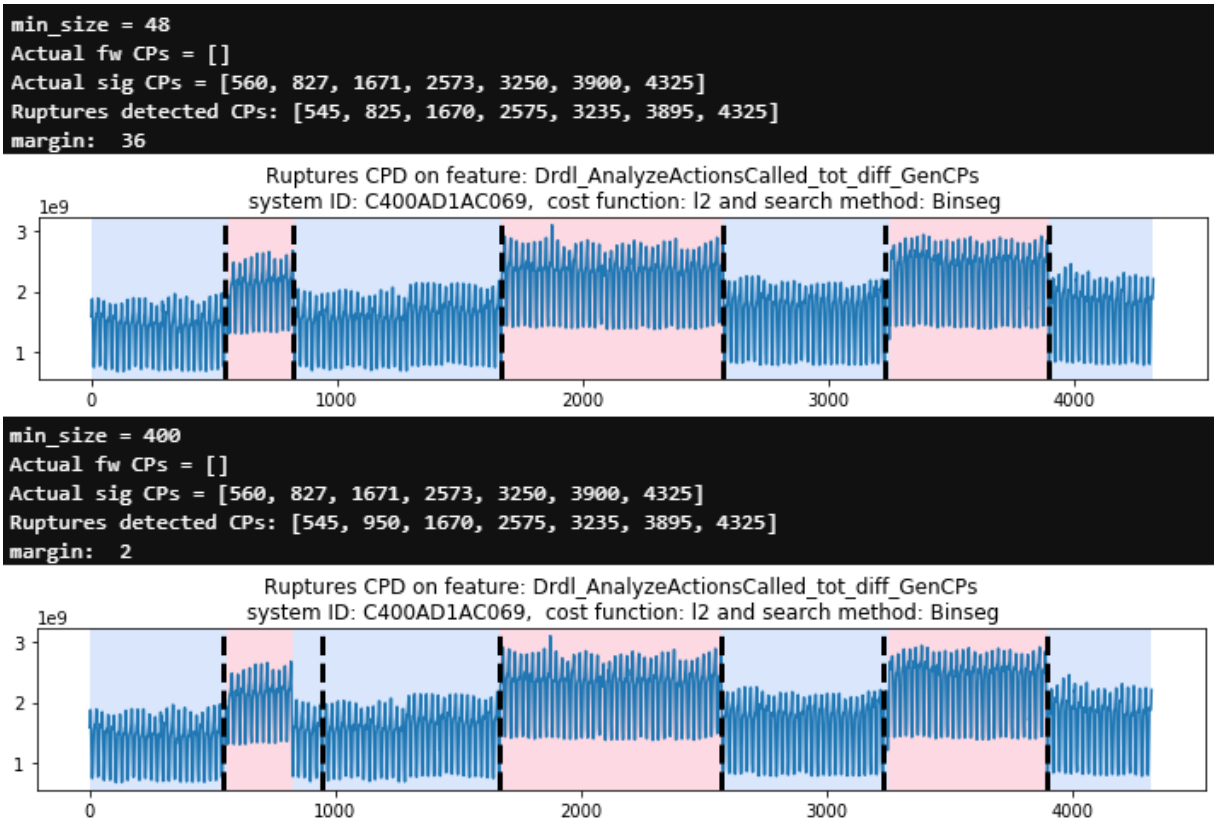

Figure 4.1.5: Illustration of the margin and min_size parameters.

to save space. As expected, these tests results are not particularly good. The average result (from the Precision and Recall evaluation method) of any combination used on any of the features had a F1-score value around 0.2. However, the sole purpose of the experiment is to get an initial understanding of which combinations actually provide results (meaning f1-scores that are not 0), which is achieved. By observing the heatmap, it is clear that a couple of features provide nothing whilst some provide results (even though they are bad) for almost all the features.

4.2 Ruptures testing

Further experiments were performed, taking advantage of the test results from the evaluation methods and heatmap. By observing which of the features/combinations of methods gave the best results, it is possible to reduce the number of features, methods and functions, to the point where only the best combinations are used and better results are shown. From that point, it is much easier to alter the different parameters to find the optimal settings. However, since this is only the beginning of the testing, not too much filtering is applied since it is not certain that the optimal settings for each combination is achieved. The plotting this far looks good and the actual change points seem to relate to the changes in the graphs. However, the Ruptures change point detection results are not as satisfactory. The observed results from Figure 4.1.6 represent the f1 score of the Precision and Recall evaluation metric, where the optimal

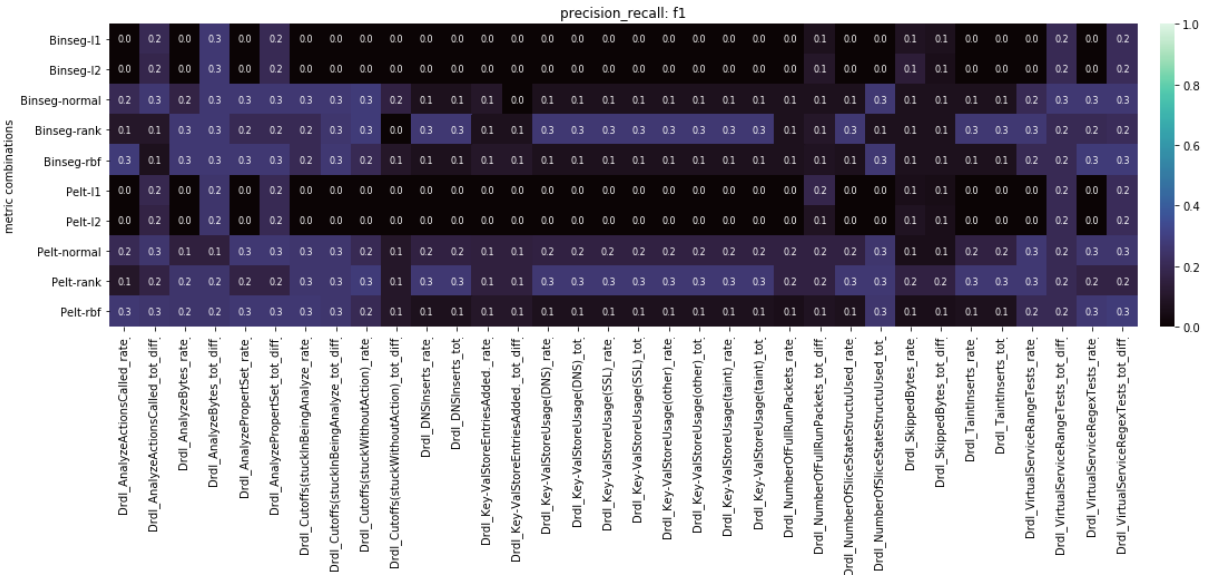

Figure 4.1.6: Illustration of a heatmap, with metric combinations on the y-axis and features on the x-axis. The presented features has been manipulated with load normalization and the moving average function. The resulting numbers are the mean F1-value for one combination of metrics, one feature and all system IDs.

result is the value 1 and the average values of every combinations varies between 0 and 0.3. If the top combinations are chosen and plotted into a new heatmap, the results are still relatively bad. But by taking a closer look as to why that is, the conclusion is that some of the system IDs returns terrible results whereas one or two returns better results.

4.2.1 Change Point Generation and Feature Manipulation

Now, as the testing continued a realization came to appear. By using the ”moving average” function, the algorithm has difficulties locating the exact positions of the change points considering the graph has a somewhat stretched out look. If there is a change point, it is be stretched out over 72 hours which makes them harder to locate. Therefore, a new function was created called ”Auto regression” (AR), which is used instead of the moving average function. To explain the ”AR” function, what happens is that for a certain feature, the 24th value (since the values are updated every hour) is subtracted by the first value to find the difference between every 24 hours, while still counteracting the day/night cycle. The subtraction is continued throughout the entire feature (and the first 24 values positions are set to ”nan” - values) which created a graph looking like the Figure 4.2.1, where as can be observed, the values can be below 0.

To be able to see, test and understand what happens with AR-created features during a change point, a method is created to be able to generate change points and specify the magnitude of the change. This is also a very good method of testing how well Ruptures finds change points with different functions and metrics and will be used frequently in

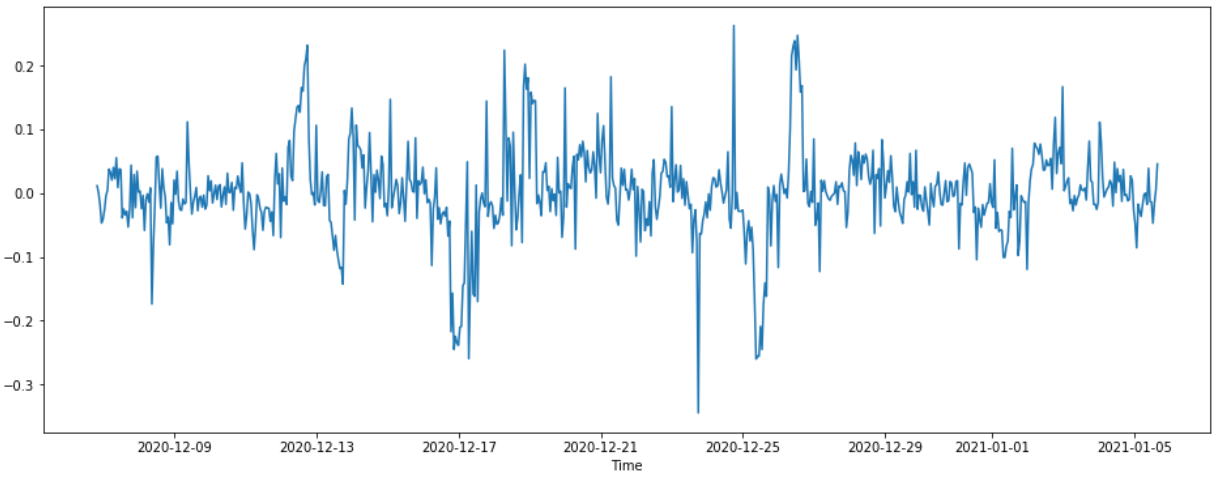

Figure 4.2.1: Illustration of how the data looks after manipulated by the ”AR” method. Which specific feature used is not important

the future testing. An example of a feature with generated change points is provided in Figure 4.2.2, where at the specified positions, all values afterwards are multiplied by an integer of choice. In this case at the first change point the values afterwards are multiplied by 100 and the values after the second change point are multiplied by 0.1. These integers are chosen to exaggerate the changes and clearly show what happens. When a change point occurs, considering the previous explained AR-function, the first new value is subtracted by the value 24 hours earlier which creates the first ”flank”. Since the next 23 values are also changed, the next 23 subtractions creates the ”tower” as can be observed. At the 25th value, both of the values in the subtraction has been multiplied with the same factor which results in the second ”flank” of the tower. For this reason, the margin parameter of the evaluation metrics are set to 28 since it is uncertain whether Ruptures detects the first or the second ”flank” as a change point. For the same reason, the minimal window of how close the change points can be detected with Ruptures, i.e. min_size, is set to 30.

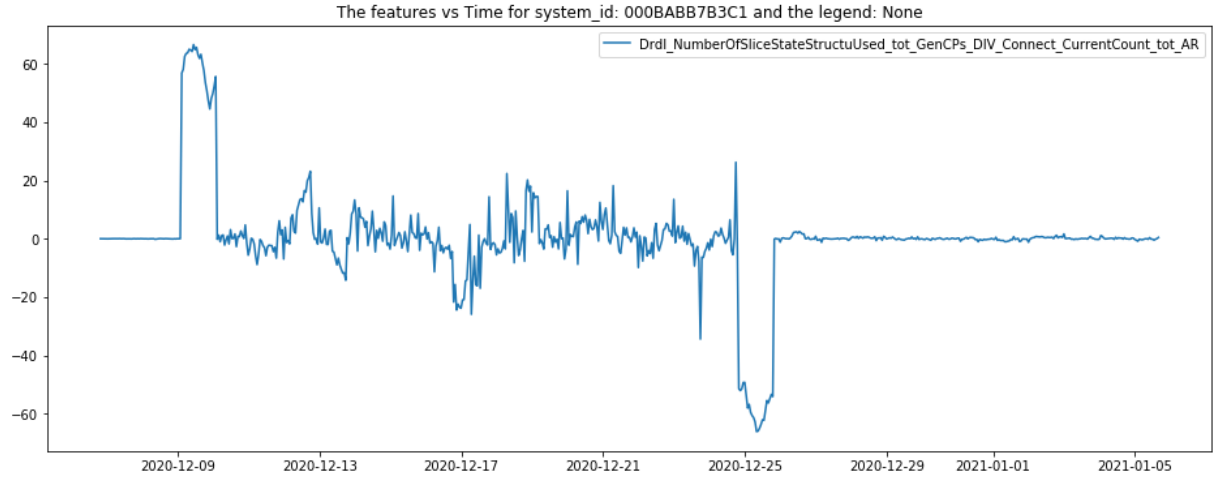

Figure 4.2.2: Illustration of how the data looks after two generated change points are added and then manipulated by the ”AR” method.

With the new functions explained, more Ruptures testing is required. The newly created features with generated change points are used to create a heatmap just as Figure 4.1.6. Some features are filtered out because of bad values(like NaN or constant 0). Two change points are generated at a random time within the time frame, with the multiplying factors 2 and 0.5. The factors are relatively high, however it makes it easier to sort out which metrics are better to keep investigating. The average f1 values of the calculations from Precision and Recall are shown in Figure 4.2.3. As can be observed, there are two combinations that are clear winners, Binseg-normal and Pelt-normal. To be certain the same combinations are the top candidates, different multiplying factors are tested, and however the factors are changed, the same two combinations provide the best results. However, the results for this specific experiment are very good in the sense that it is clear that the combination of the AR-method and generated change points actually works, and Ruptures are able to detect the change points which means it is something to keep using for further experiments.

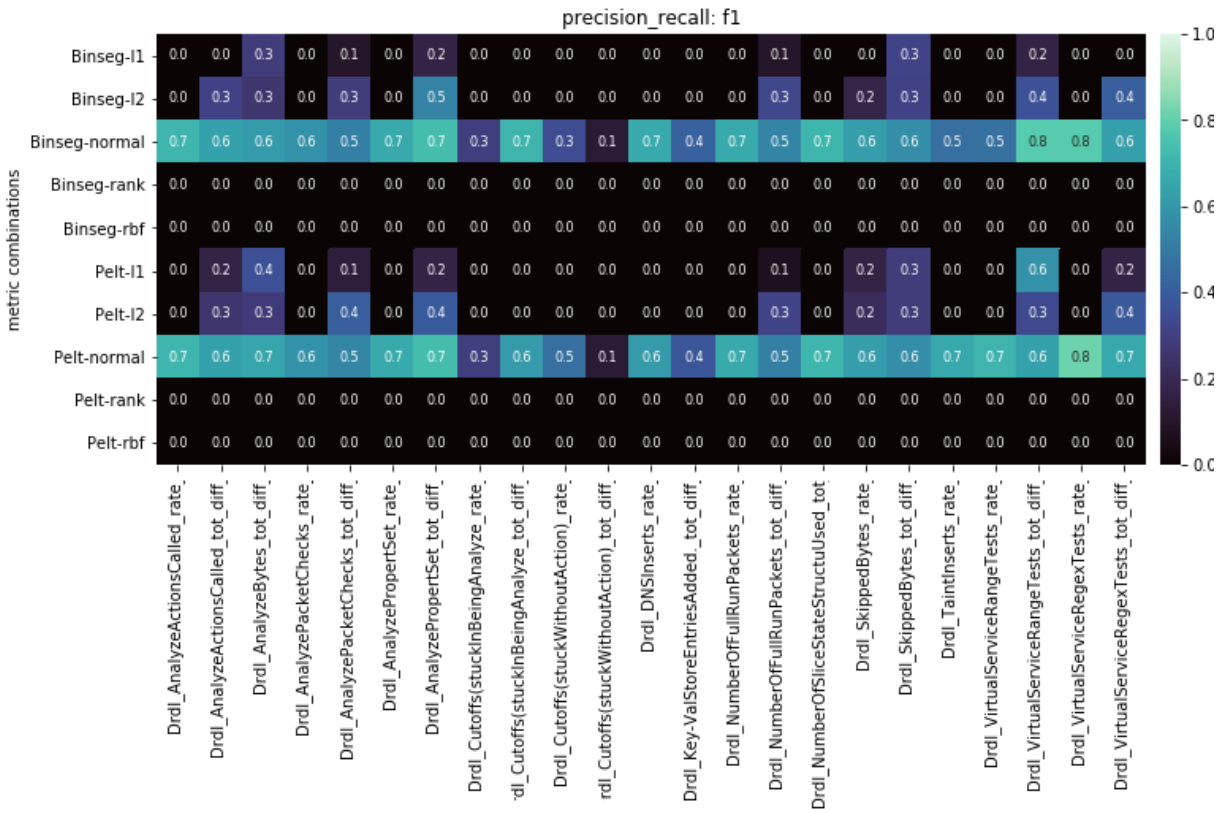

Figure 4.2.3: Illustration of a heatmap where features have generated change points and has been manipulated by the ”AR” method as well as load normalization.

A new data log is acquired including the most interesting features and more ”cleaned up” system IDs, meaning they have less ”weird” values. Also, the time span is increased from one month to three months. Using this new data log and the newly acquired knowledge, more testing is done. First and foremost, some new functions are created to plot and visualize not only the created features, but also the underlying features. As an example, when plotting a certain feature, the first subplot only has the applied diff

method, the next subplot is without ”AR” but has load normalization, and the last one also includes the ”AR” manipulation. This allows for a deeper understanding as to why certain features looks as they do. An example of what a plot like this can look like is shown in Figure 4.2.4.

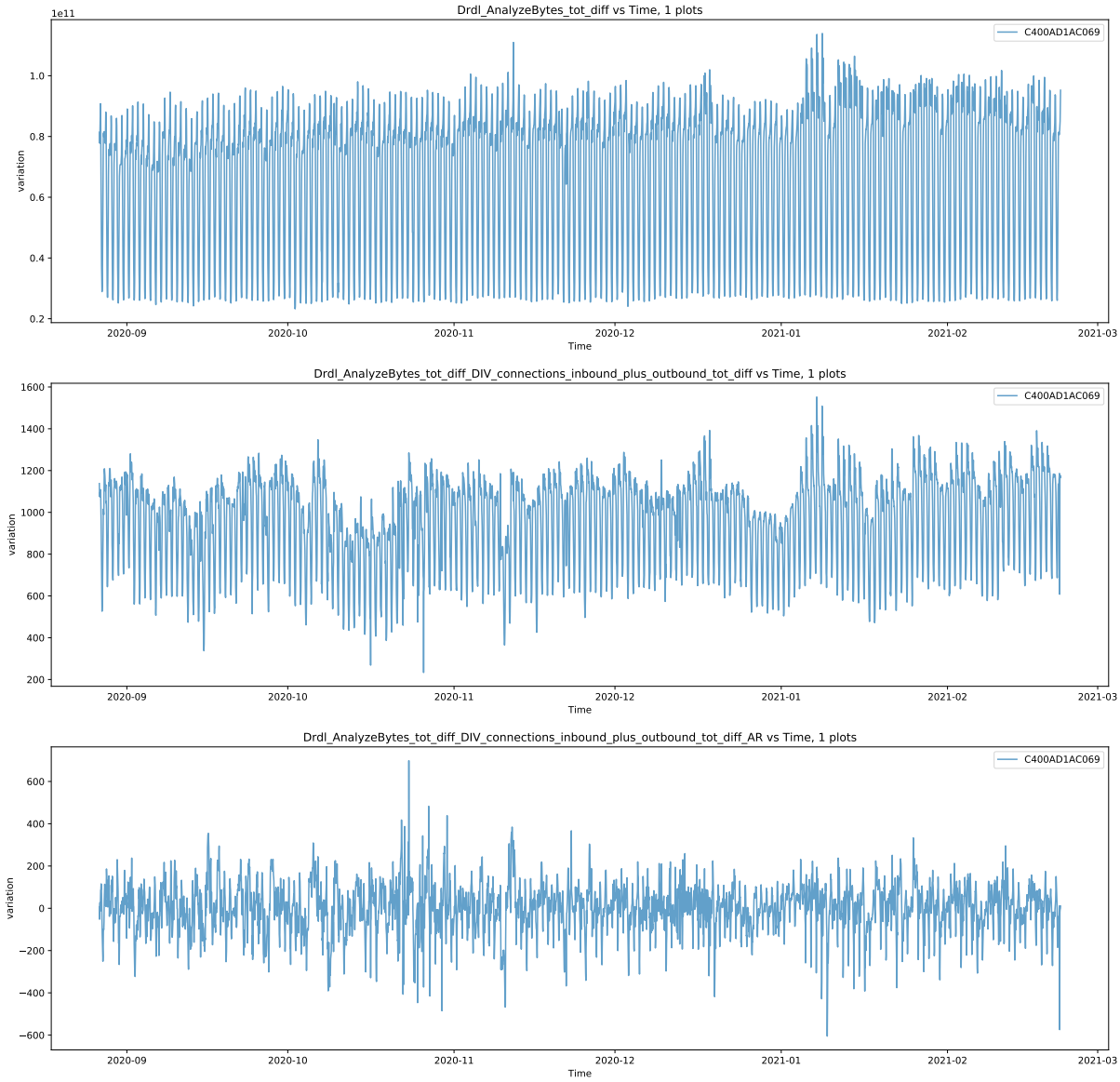

Figure 4.2.4: Illustration of how subplots of any specific feature are plotted.

4.2.2 Filtering Strategies and Parameter Settings

While the factors have been changed during the previous tests, the number of change points have not. Increasing the number of generated change points is part of the next step, so the tests will include an additional two generated change points. Further, more of the search methods and cost functions needs to be tested and evaluated. The strategy for testing the rest of the methods and function combinations is to first investigate the different parameter settings. Next, the new combinations are tested individually with

Ruptures on one very ”clean” system ID, and then plotted to be able to see whether or not the specific combination produces good results/works or not. If a combination seems bad, it is not further used. Next, we look at the features individually to see if some of them are bad, if a feature seems bad for more than one system ID, it is removed as well. The next Ruptures evaluations will be executed with the combinations and features remaining after this filtering on the ”clean” of system ID, and will be plotted into a heatmap for each evaluation metric. The heatmaps will be observed and if there are combinations of features that performs very bad, they will be removed.

After some further contemplation, it is decided that the AR-method will no longer be used since the additional testing indicates that the day/night-cycle does not affect the Ruptures-change point detection that much, and the created methods is considered not good enough. The generated change point function is also changed at this point. Instead of multiplying the specified factor with the values after a specified position, the equation instead multiplies the factor (subtracted by one) with the mean value of the data set, and then adds that result to each values after the generated change point in the data set. This is done because the changes in the mean is of importance, instead of exaggerating the already existing variation.

Investigations of the parameter settings follows, and the outcome is that setting the correct penalty is quite complicated. If the number of change points are unknown, there are different penalty functions depending on which cost function are used. For example, for the cost function ”l1” the penalty equation is

$$pen = sigma * sigma * numpy.log(T) * d$$

where T is the number of samples in a signal, sigma is the noise standard deviation and d is the number of dimensions. With this new information, more testing is done with the same features and metrics as before and as a result, very much more appealing heatmaps are generated. It is concluded that the correct penalty functions for each of the cost functions is crucial for a correct change point detection. However, the Pelt search method can not handle the ”n_bkps” setting and the Dynamic programming search method can not handle the ”pen” parameter, meaning that for some specific testing, they may have to be excluded.

4.2.3 Experiments

Method and function Combination Filtering Experiment

As mentioned earlier in this section, a system ID with very little changes are chosen. This specific system ID has one signature change, but it does not show in the plots as a change point. This is a good thing for the testing at hand which is run with four generated change points where the factors are varied from high to low. The different parameters are set as desired: min_size is 24, penalty (n_bkps) is 4 and margin 30. The produced heatmap, Figure 4.2.5 displays very good results, which is expected considering all the changes such as adding a number of extra metric combinations,

more filtering of features, better feature manipulation, correct penalty settings, better parameter settings, new data set and a very ”clean” system ID. It becomes very clear that some (or all) of these changes improved the change point detection drastically and it is also very clear that the ”clinear” cost function can be excluded in further testing. The reason for the bad results from Pelt is that pelt does not have the n_bkps parameter, so the penalty function is used instead. However, the fact that the results are so bad for that specific penalty function is concerning and will have to be investigated.

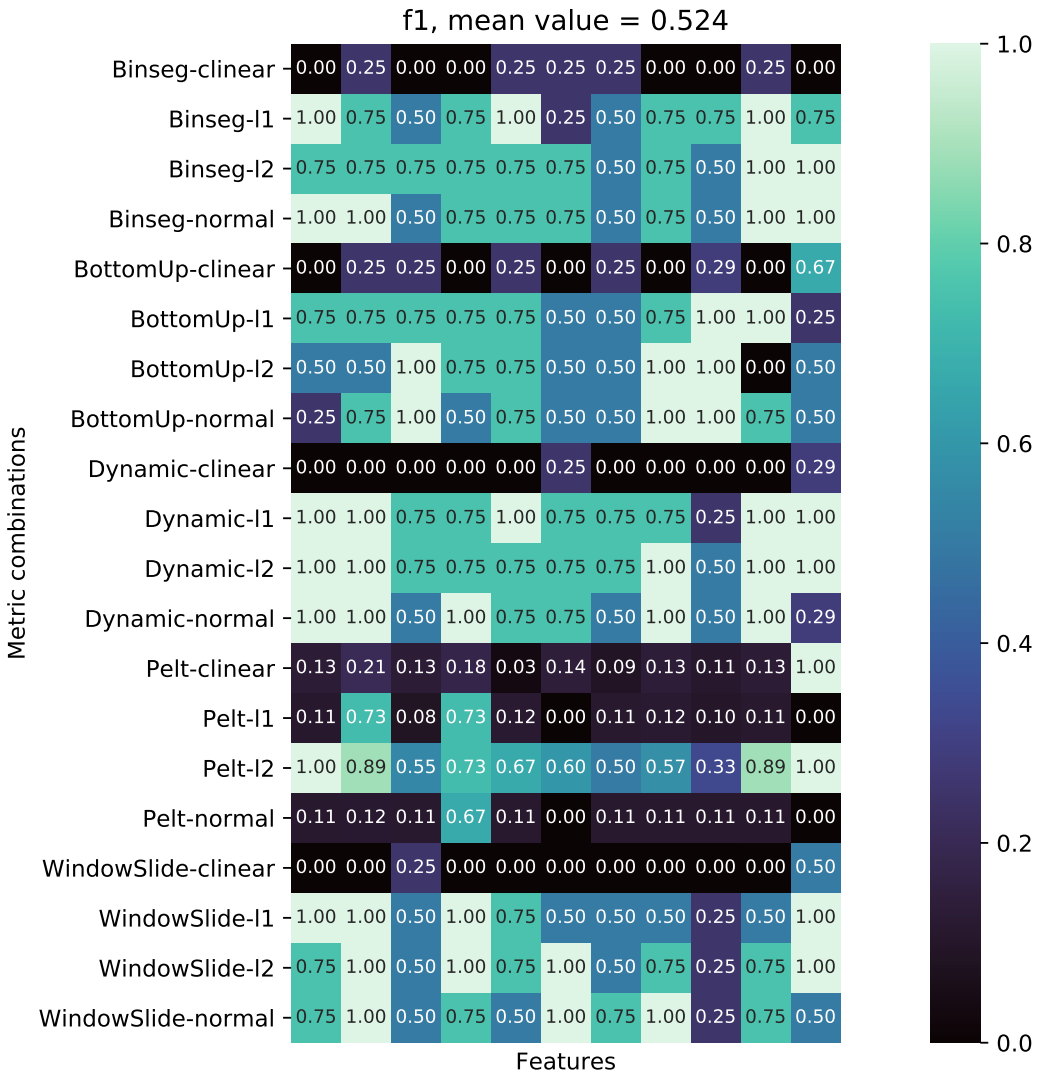

Figure 4.2.5: Illustration of a heatmap with four generated change points for all features, the n_bkps parameter is set to 4 and the test in run on one very ”clean” system ID. The features examined in this figure can be located in appendix A list 1.

Experiment on Features with Obvious Visual Change Points

Now, since the experimenting was done on a system ID that does not contain that much noise of different kinds, this result is very satisfactory but not a reliable source of reference for the later experiments on a bigger data set with many system IDs.

However, it shows which combinations are the best for this type of data which still is very useful. Next, some other tests are created for a very similar purpose, to see and get an understanding of which combinations are the best. By looking at the plots of all the features, ten of them were chosen for a certain system ID where the data clearly has change points in them. The change points does not have to be a signature update or firmware update, just a clear change in the graph. The positions for those changes are noted and used as the actual change points when running Ruptures, and the number of change points for each feature and system ID is defined. This test is to see how well Ruptures find the clear change points when there is quite a lot of noise. The resulting heatmap can be observed in Figure 4.2.6. The penalty equation is still not changed which is the reason for Pelt providing bad results. However, the same test is run again and can be observed in Figure 4.2.7, but this time with changed penalty equations for each cost function. The results are much more appealing and the penalty equations seem to be relatively good. Comparing Figure 4.2.6 and 4.2.7 is educative, since the overall results from Figure 4.2.6 are higher which means that Ruptures have a easier time finding change points if the number of change points are defined. This was expected but it is nevertheless good to confirm the expectations.

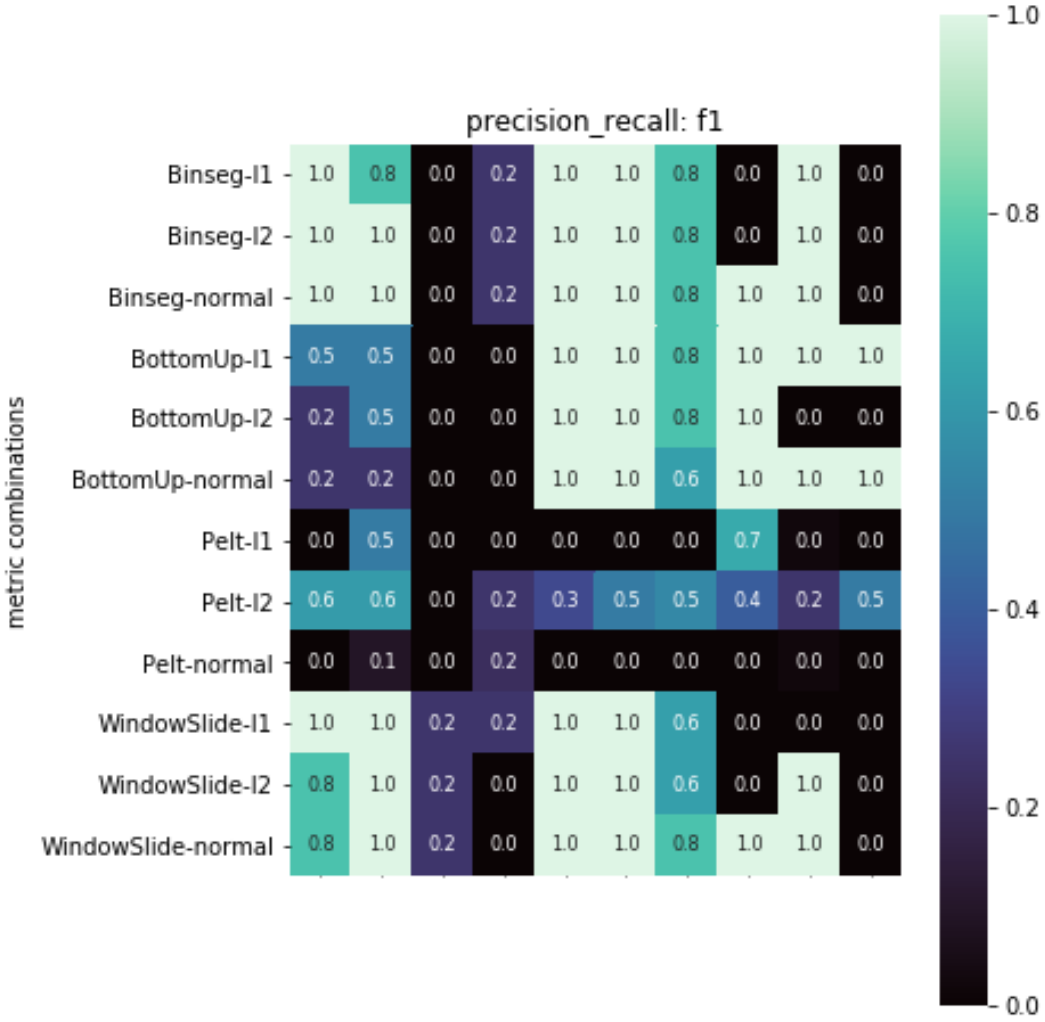

Figure 4.2.6: Illustration of a heatmap with a known set of change points (non generated) for each different feature/metric/system-ID combination and the n_bkps is set thereafter. The features examined in this figure can be located in appendix A list 2.

General Experiment on All Current System IDs

Further experiments are run with the aim of providing reliable and relatively comparable results to the final test run later on. The features chosen are tested on all system IDs for the data set with four generated change points for every feature. The first test is run with the N_bkps parameter set to 4 and the second test with the penalty equations instead of a defined number of change points. The second test allows for a more generalized result when compared to the actual reality, but the first one is run to see how well Ruptures handles more noise. Not all systems are spotless and Ruptures needs to be able to handle occurring problems such as increased noise. Also, during this test, two different methods of calculating the penalty for each cost function are tested based on the Akaike information criterion (AIC) and the Bayesian information criterion (BIC, the one used from the start of the testing phase) [27]. However, the clear winners are the different penalties based on the Bayesian information criterion

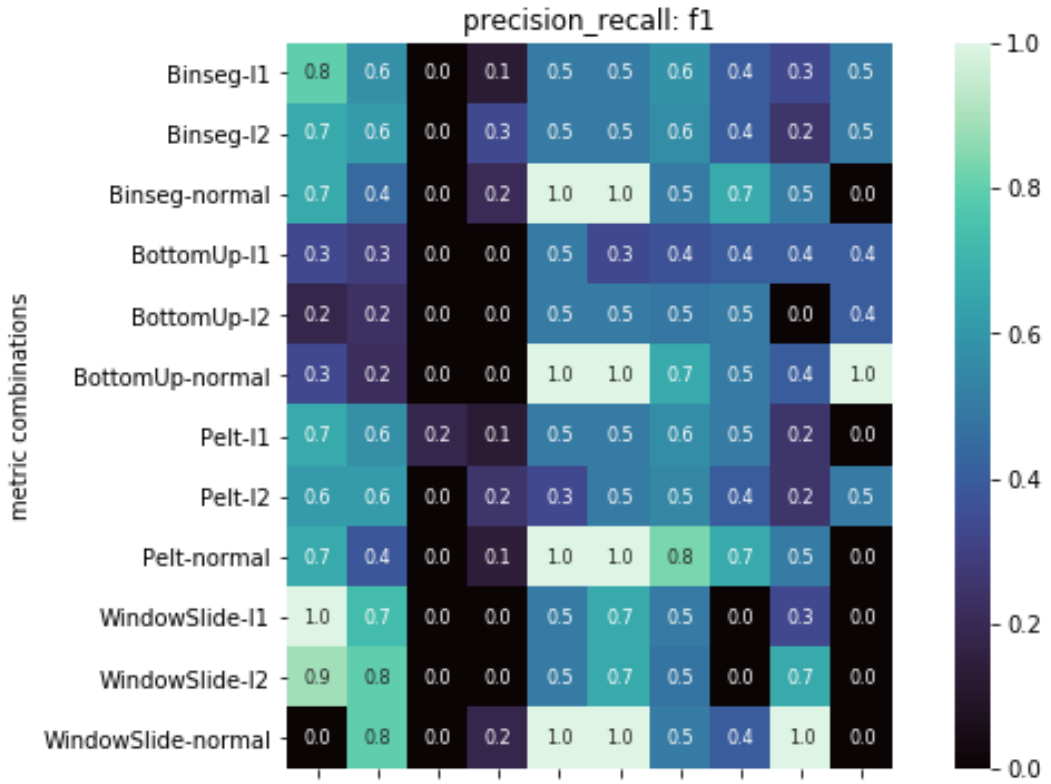

Figure 4.2.7: Illustration of a heatmap with a known set of change points (non generated) for each different feature/metric/system-ID combination and the penalties are defined by specific penalty functions. The features examined in this figure can be located in appendix A list 2.

(BIC) which means the future tests will be run with that. The tests concluded as per usual in some heatmaps and some of the results can be observed in figures 4.2.8 and 4.2.9. The two tests have a very mixed bag of results. Just as the two previous tests, comparing Figure 4.2.8 and 4.2.9 it is still clear that Ruptures detects change points better if you define how many there are with the n_bkps parameter than the general solution, and after these two experiments it is also obvious that more noise decreases the accuracy drastically. Also, no more filtering of metric combinations or features are required since all of them seem to hold up relatively good.

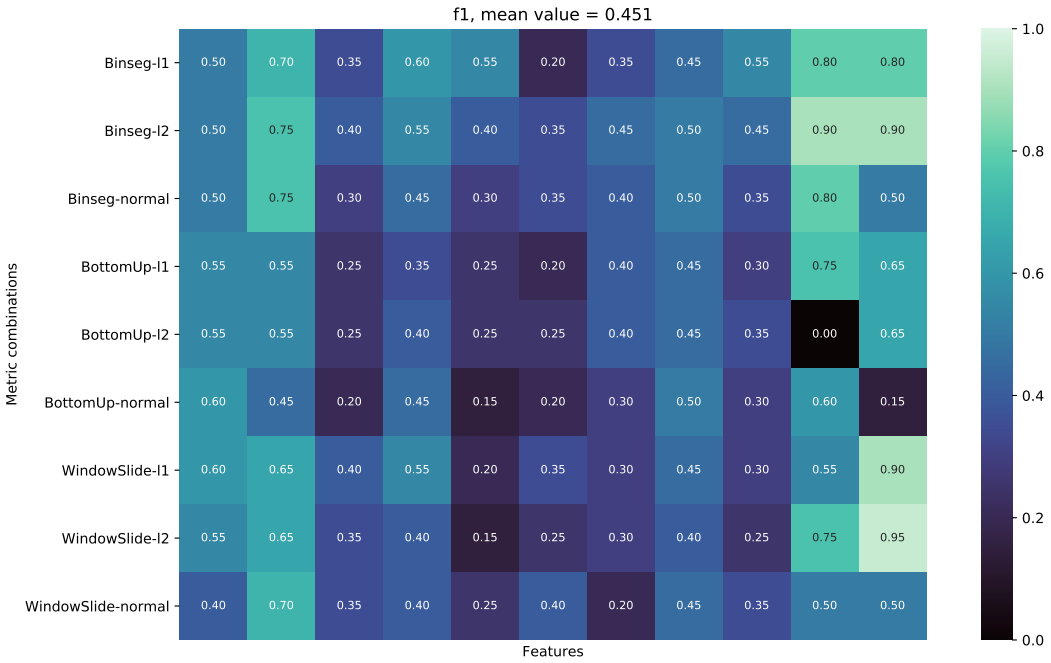

Figure 4.2.8: Illustration of a heatmap with results from Ruptures run over all current system IDs, where the features has generated change points and the n_bkps parameter is specified. The features examined in this figure can be located in appendix A list 1.

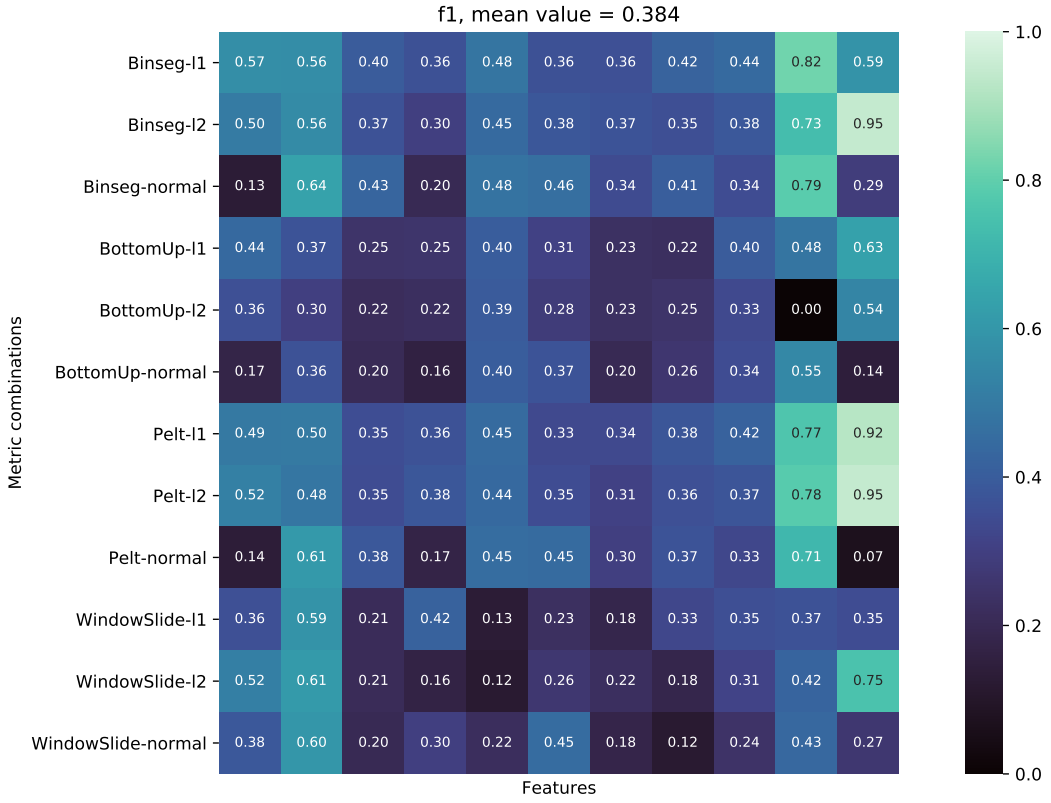

Figure 4.2.9: Illustration of a heatmap with results from Ruptures run over all current system IDs, where the features has generated change points and the penalty is specified by penalty equations. The features examined in this figure can be located in appendix A list 1.

Chapter 5

Evaluation and Analysis

This chapter includes the final tests, filtering of features and method/function combinations, the optimized parameter settings as well as analysis of the results.

5.1 Log Change Detection Experiments with Generated Change Points

Initially, the synthetic data testing provided the expected results, meaning the output of the few experiments made, with some variations in parameters, confirmed the existing beliefs and thoughts. For example, the more the noise increases, the harder it is to find change points, and with more dimensions, the easier it gets. Furthermore, the feature manipulation functions properly produced the correct assumed graphs. The manipulation functions includes the load normalization function, the ”diff” method to remove the monotonically increasing values of certain features, the normalization function to normalize all values in subplots to vary between 0 and 1, the moving average function and ”AR” function to remove the day/night-cycle of the variation, the function for generating change points with chosen factors, the different plotting functions, the heatmap-creation function, and finally the function to create the Ruptures analysis with different parameters and metric combinations. All these functions has led to this last experiment and regardless of which of the functions are used in the end, they have all provided well needed information.

The final experiment, before the large scale test, is run on the last data set with a very large amount of system IDs. However, first the data set is filtered to have 100 system IDs (picked randomly), and then 10 random system IDs are chosen of those to reduce the execution time. Furthermore, the best configuration combinations from all the previous tests are used, which have been narrowed down throughout the testing phase to the following:
Search methods:

- Binary segmentation

- Window sliding segmentation
- Bottom-up segmentation
- Linearly penalized segmentation (Pelt) for when the number of change points are unknown
- Dynamic programming (Dynp) for when the number of change points are known

Cost functions:

- Least absolute deviation (CostL1)
- Least squared deviation (CostL2)
- Gaussian process change (CostNormal)

Evaluation metrics:

- Precision and Recall
- Hausdorff metric
- Rand index

Four change points are generated for all features of interest, and the parameter settings are min_size = 36, margin = 24, the penalty is defined with the different penalty functions for the different cost functions, meaning the tests are executed as if the number of change points are unknown. The factors of the change points are (1.4, 0.6, 1.4, 0.6), meaning after the generated change points, the values either increases or decreases by 40 percent of the total means' value. The penalty functions are:

```
#for costL1:
pen = numpy.log(len(signal)) * numpy.sum(abs(signal - numpy.median(signal))
    ) / len(signal)

#for costL2:
pen = numpy.log(len(signal)) * numpy.var(signal)

#for CostNormal:
pen = numpy.log(len(signal)) * numpy.linalg.slogdet(numpy.array([[signal.
    var()]]))[1]
```

Furthermore, the features that are be experimented on are few, which is both to reduce the execution time of the experiment and because the chosen ones are of more interest than the rest. Also, this experiment is run on not only the load normalized features with the generated change points added *before* the normalization, but also the same set of features but with the generated change points *after* the load normalization, and lastly with no load normalization at all. This means that for Figures 5.1.1, 5.1.2 and 5.1.3, the features looks like (for example) ""feature"_diff_GenCPs_DIV_connections_inbound_plus_outbound_tot_diff", ""feature"_diff_DIV_connections_inbound_plus_outbound_tot_diff_GenCPs" and ""feature"_diff_GenCPs" respectively.

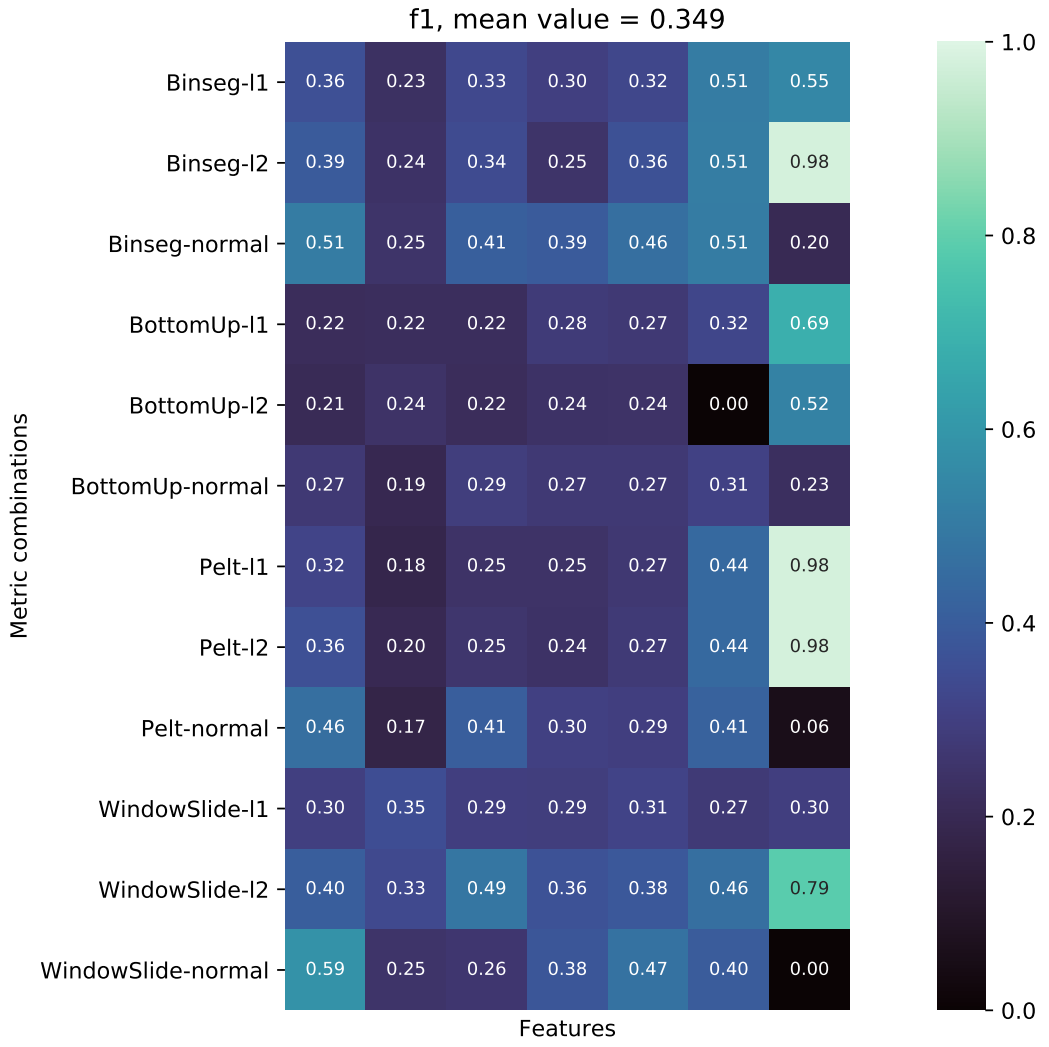

Figure 5.1.1: Illustration of a heatmap with results from Ruptures run over 10 random system IDs, and features with generated change points and load normalization. The features examined in this figure can be located in appendix a list 3.

The differences between these three tests are significant, yet not very big. The first one represented by Figure 5.1.1 had a mean F1-value variation between 0,24 and 0,46 for each combination of metrics, Figure 5.1.2 between 0,26 and 0,53 and lastly Figure 5.1.3 varies between 0,12 and 0,58. Which combination of metrics is the best one is hard to decide, considering the best combination is not the same one in all three cases. For the first case, Windowslide-l2 seems to produce the highest mean f1 value, for the second and third case it was Binseg-l1. To be sure about which combination is the best for each experiment, a function is created which counts the number of times a certain combination returns the highest f1 value. The result of the function is a table and the three tables for the three experiments are put into Figure 5.1.4, where the tables are in the same order as the three previously explained test runs. The tables, as observed, also contains the amount of times the Hausdorff metric and Rand index metric were best. However, throughout the entire testing phase, the values returned from Hausdorff and Rand index has been hard to understand and derive understanding from. The f1 value

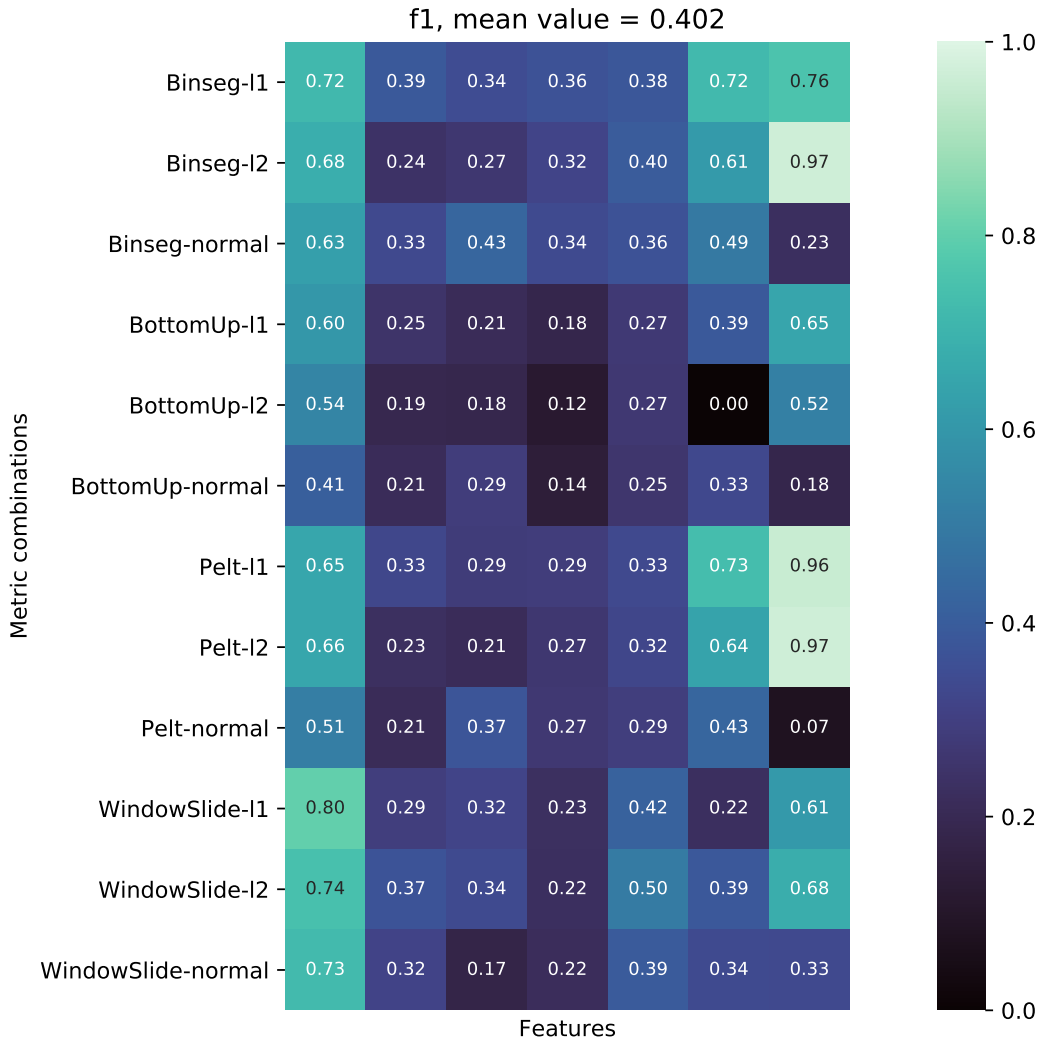

Figure 5.1.2: Illustration of a heatmap with results from Ruptures run over 10 random system IDs, and features with generated change points and load normalization, but the generated change points are added after the load normalization. The features examined in this figure can be located in appendix a list 3.

is the only evaluation value that has been actively considered, but the other two are not ignored but more used for comparison to the f1. Something that also becomes very clear from studying Figure 5.1.4 is that the Pelt search method is not useful at all, considering it never provides the best results for any situation, and neither does BottomUP. This table is very informative and useful for the large scale log detection experiment, where both of the aforementioned search methods can be disregarded if necessary. The three different ways of manipulating the features are, as shown, providing dissimilar results. However, it was done to test whether or not there is a difference between adding the change points before and after load normalization, but mostly to see if the best combination of methods and functions are the same for all three tests. By looking at the heatmaps and the average results of them, it seems like load normalization is not relevant considering the experiment with no normalization provided the highest results. However, load normalization is required to better evaluate the actual data of

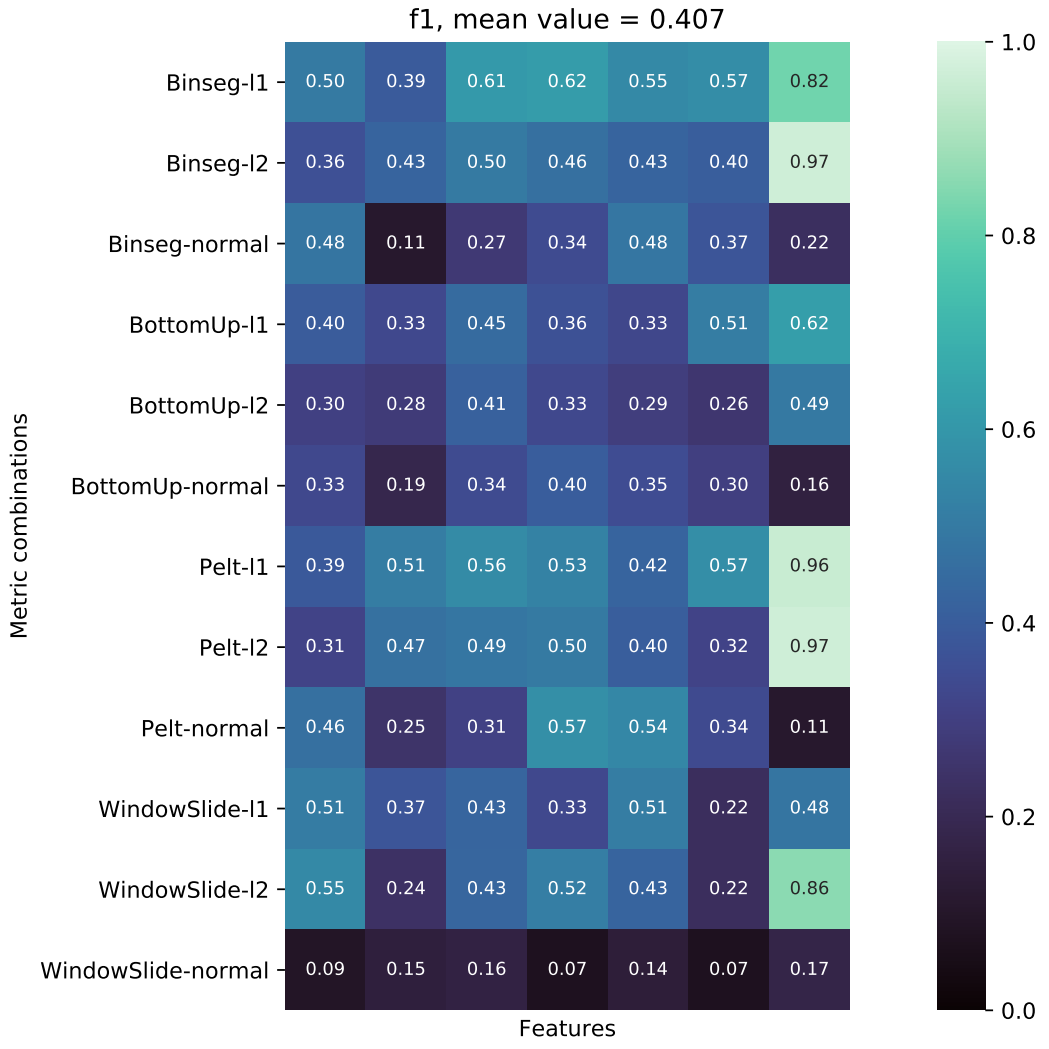

Figure 5.1.3: Illustration of a heatmap with results from Ruptures run over 10 random system IDs, and features with generated change points but no load normalization. The features examined in this figure can be located in appendix a list 3.

features and not data which is subject to change because of the number of connections etc. These results with tables of which combinations are the best has provided some much needed information to be applied to the final large scale test analysis, where additional filtering can be applied. Although, what has not been made clear so far is why the average F1-value results are so low. One of the reasons for this is the fact that there is a lot of noise for certain systems. Often there is one or more actual change points in the system which are not accounted for, i.e. not generated change points. Therefore, many times Ruptures actually finds a correct change point which is not noted as a true positive since it is not a generated change point, which means the average F1-value will be reduced.

Features with generated change points BEFORE load normalization

	search_method	cost_function	number_of_times_best: f1	number_of_times_best: rand_index	number_of_times_best: hausdorff
0	Binseg	l1	8	7	9
1	Binseg	l2	12	6	5
2	Binseg	normal	15	14	7
3	BottomUp	l1	3	1	4
4	BottomUp	l2	0	0	0
5	BottomUp	normal	1	7	3
6	WindowSlide	l1	6	6	7
7	WindowSlide	l2	10	14	4
8	WindowSlide	normal	9	10	6
9	Pelt	l1	0	1	3
10	Pelt	l2	1	0	1
11	Pelt	normal	1	0	17

Features with generated change points AFTER load normalization

	search_method	cost_function	number_of_times_best: f1	number_of_times_best: rand_index	number_of_times_best: hausdorff
0	Binseg	l1	21	16	1
1	Binseg	l2	14	6	3
2	Binseg	normal	7	10	3
3	BottomUp	l1	0	3	7
4	BottomUp	l2	0	1	1
5	BottomUp	normal	1	5	1
6	WindowSlide	l1	9	10	15
7	WindowSlide	l2	8	9	9
8	WindowSlide	normal	4	3	8
9	Pelt	l1	2	2	3
10	Pelt	l2	1	0	2
11	Pelt	normal	1	3	15

Features with no load normalization

	search_method	cost_function	number_of_times_best: f1	number_of_times_best: rand_index	number_of_times_best: hausdorff
0	Binseg	l1	18	19	2
1	Binseg	l2	9	3	2
2	Binseg	normal	9	4	5
3	BottomUp	l1	6	8	2
4	BottomUp	l2	1	0	1
5	BottomUp	normal	1	0	5
6	WindowSlide	l1	5	4	8
7	WindowSlide	l2	5	11	6
8	WindowSlide	normal	1	0	12
9	Pelt	l1	2	4	0
10	Pelt	l2	1	1	3
11	Pelt	normal	3	7	15

Figure 5.1.4: Tables that show the number of times a certain metric combination has provided the best results, from evaluations in Figures 5.1.1, 5.1.2 and 5.1.3 respectively.

5.1.1 Increased Number of System IDs

The large scale test is divided in two steps. The first step runs a Ruptures evaluation experiment on 100 randomly picked system IDs from those containing change points, including both noisy and less noisy ones. Using the lessons learnt from the test in Section 5.1 , only two search methods are used (Binary segmentation and Window Sliding), otherwise the same parameter settings are used. Furthermore, regarding the feature manipulations, the experiment will be done on two different sets of features. Just as in the previous test in Section 5.1 where the experiment was executed with three different variants. However, it is decided that load normalization is of importance which means that the two variants used in this test will be features with generated change points *before* load normalization, and features with generated change points *after* load normalization. This step is important to observe if the generalized experiment in Section 5.1 held up with regards to settings and metrics combinations. The results of the two tests, which can be seen in Figures 5.1.5 and

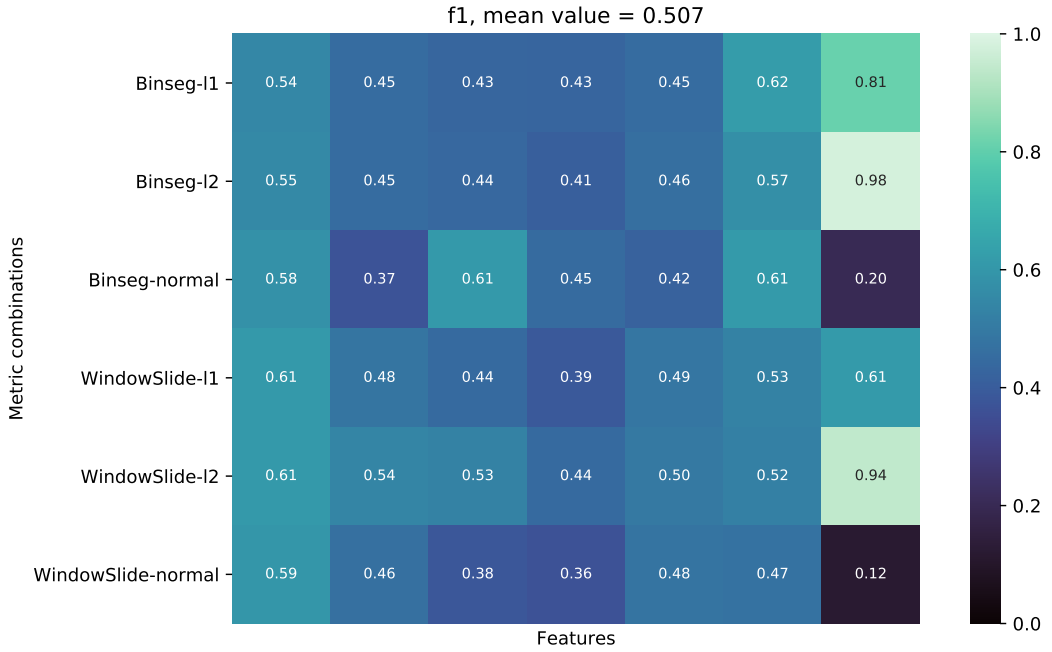

Figure 5.1.5: Illustration of a heatmap with results from Ruptures run over all system IDs, and features with generated change points and load normalization. The features examined in this figure can be located in appendix a list 3.

5.1.6, are very alike, however they still differ a bit. As for the features with generated change points before the load normalization, what actually happens is that a value is added to the existing value at all subsequent timestamps after a change point when no load normalization is applied. This means that if there is any exaggerated variation because of a higher connection count or CPU load usage at a time, it will be exaggerated even more and will most probably be noticed as a change point even if it really is not. However, for many cases, it seems to help in certain situations where the initial graph is clean/straight. Regardless of this, considering the results are so much alike, it has been confirmed that the settings used are correct and will most likely be about the best

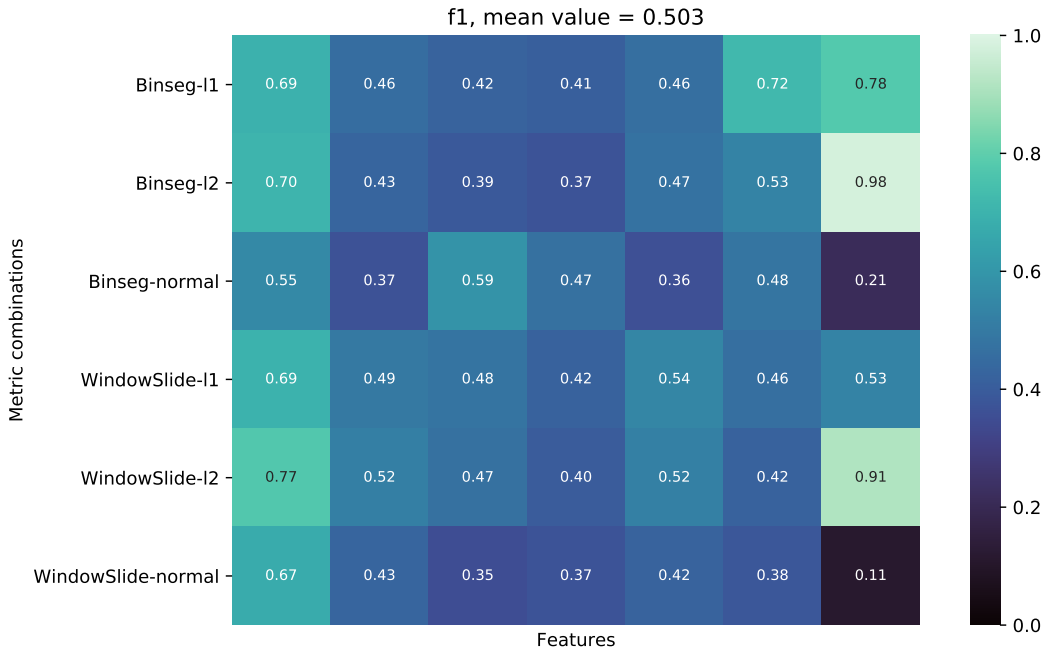

Figure 5.1.6: Illustration of a heatmap with results from Ruptures run over all system IDs, and features with generated change points and load normalization, with the generated change points added after the load normalization. The features examined in this figure can be located in appendix a list 3.

settings to use for the next step. The average f1 values from Figures 5.1.5 and 5.1.6 are ranging from 0.4 to 0.58, which is a good indicator that if there are any noticeable change points they will likely be found and be accurate.

5.2 Log Change Detection with Actual Change Points from System/Configuration Updates

The last step is to run with the same parameter settings as the previous tests 5.1.5 and 5.1.6. However, the system IDs used will be decided by a function created to determine whether or not a system ID has either signature changes or firmware changes in them. There are about 1000 system IDs in total, but only 213 of them contains signature or firmware changes. Furthermore, there will be no generated change points in this test, which means the evaluation of this Ruptures test will be the results of Sandvines updates or changes in the systems. All the tests previous to this one were created for the sole purpose of identifying the settings, both parameters and combinations, which achieve the best results possible in this test.

The experiment required about 72 clock-hours of execution time, and the results were very interesting. After a brief study on both the Hausdorff and Rand-index evaluation metrics, it is concluded that these two evaluation metrics are not particularly informative, so they are not presented or discussed. However, there are as explained three values that can be extracted from the Precision and Recall evaluation method. All

three parts are shown in figures 5.2.1, 5.2.2 and 5.2.3. As can be observed in all three graphs, the average values are very low. To clarify further analysis and discussions,

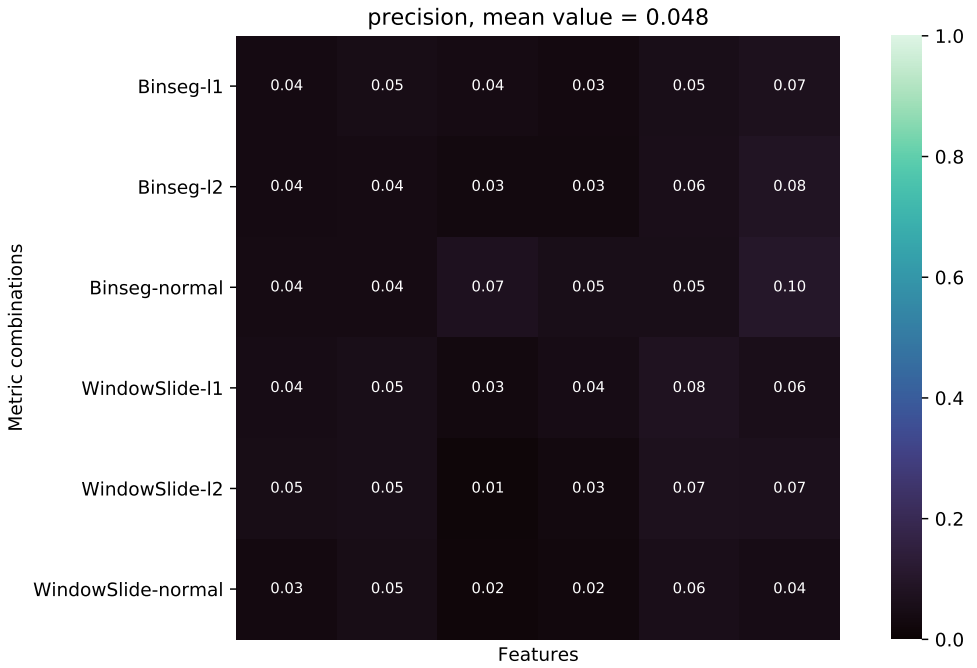

Figure 5.2.1: Illustration of a heatmap with results from the large scale log experiment Ruptures run over all system IDs containing change points. The heatmap displays the precision value of the Precision and Recall evaluation method. The features examined in this figure can be located in appendix a list 4.

we introduce two concepts that will be referred to when discussing change points. ”Actual” change points, i.e. changes due to firmware updates or signature updates will be referred to as s-change points, and changes due to unknown reasons i.e. changing behaviour in features will be referred to as f-change points.

The low values are not surprising. The fact is that the system IDs are generally very noisy. There are a lot of f-change points in the data and many of the s-change points does not seem to affect the variation at all. Because of these facts, recall is the value that is most interesting. Recall determines the value of how many of the s-change points are found, meaning how many of the signature updates or firmware updates are co-occurring with a change. Furthermore, considering the large amount of noise and the non-altering changes, 0.14 (from Figure 5.2.2) may be a relatively good average recall value for this specific experiment. Also, it is quite clear from Figure 5.2.2 that the Windowslide search method does not perform as well as Binary segmentation, meaning if we were to only use Binary segmentation, the average value would be even higher.

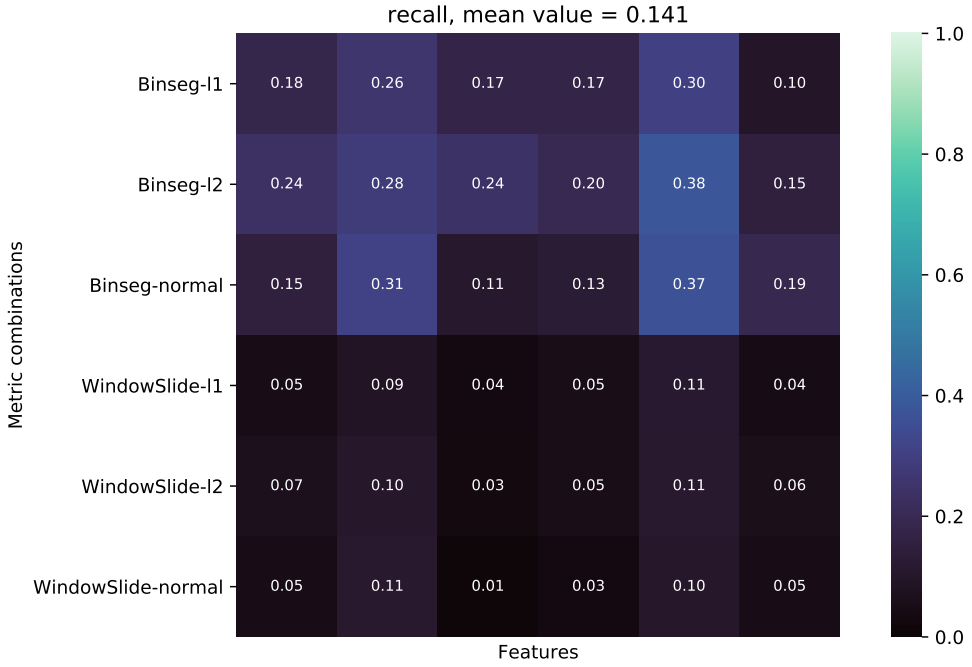

Figure 5.2.2: Illustration of a heatmap with results from the large scale log experiment Ruptures run over all system IDs containing change points. The heatmap displays the recall value of the Precision and Recall evaluation method. The features examined in this figure can be located in appendix a list 4.

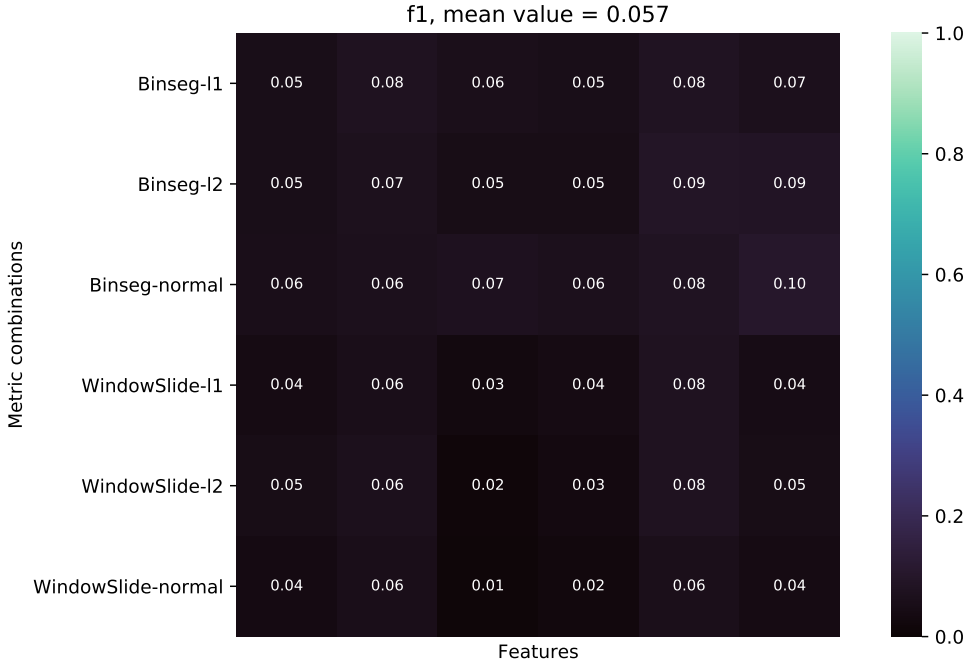

Figure 5.2.3: Illustration of a heatmap with results from the large scale log experiment Ruptures run over all system IDs containing change points. The heatmap displays the f1 value of the Precision and Recall evaluation method. The features examined in this figure can be located in appendix a list 4.

Taking a closer look at the graphs of some of the evaluations, it becomes more clear as to why the evaluation values are low. Figures 5.2.4, 5.2.5, 5.2.6, 5.2.7 and 5.2.8 displays examples of some Ruptures evaluations where the recall values are high, but as can be observed, there are also a couple of f-change points in the data. In these figures, the dashed lines are Ruptures detected change points, and the change in color are the actual change points, i.e. s-change points. The Figures 5.2.9, 5.2.10, 5.2.11, 5.2.12

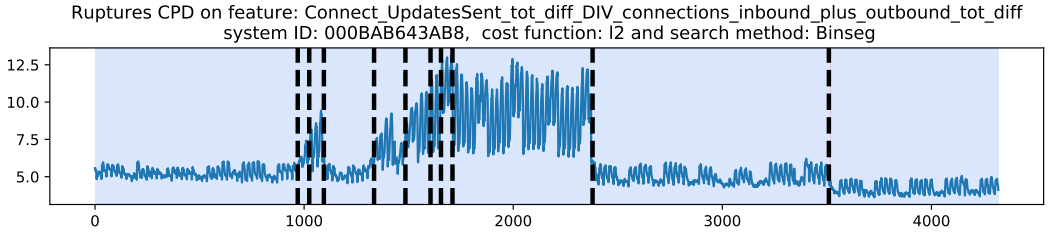

Figure 5.2.4: Ruptures change point detection with 1.0 recall value.

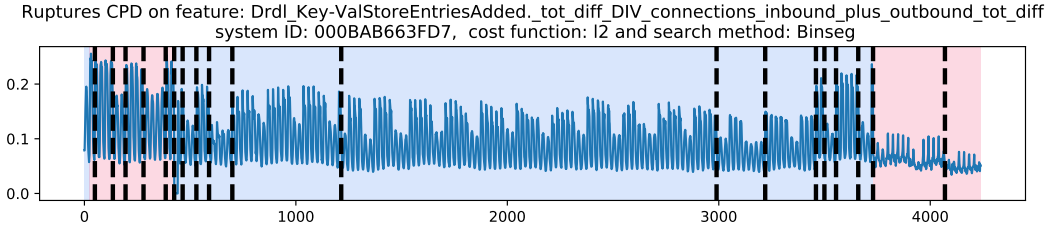

Figure 5.2.5: Ruptures change point detection with 1.0 recall value.

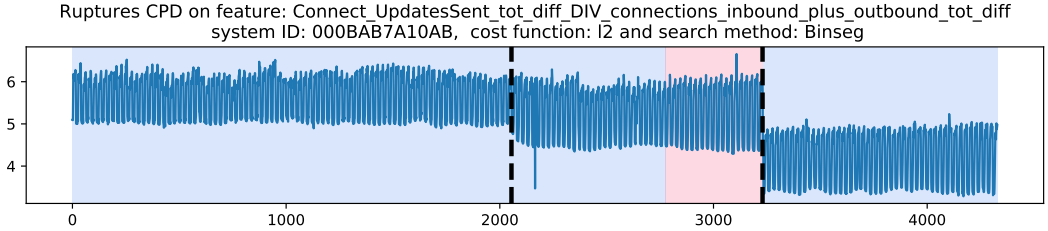

Figure 5.2.6: Ruptures change point detection with 0.5 recall value.

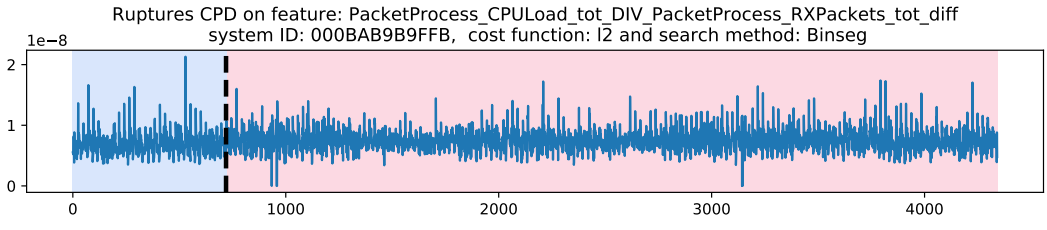

Figure 5.2.7: Ruptures change point detection with 1.0 recall value.

and 5.2.13 however, displays the Ruptures evaluation results of some of the really bad evaluations where no s-change point is found by Ruptures, and there are at the same time some obvious f-change points in the graphs. Both the Recall and Precision values for these graphs are 0.0, meaning the f1 value is 0.0.

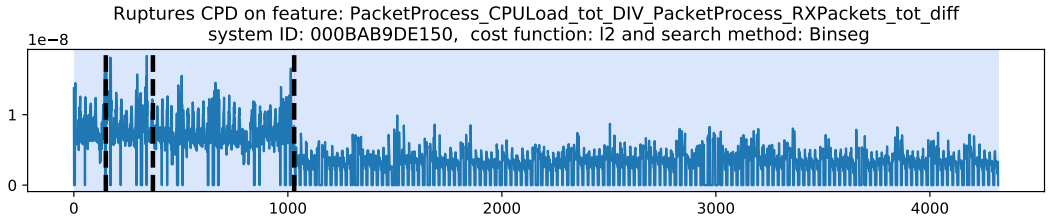

Figure 5.2.8: Ruptures change point detection with 1.0 recall value.

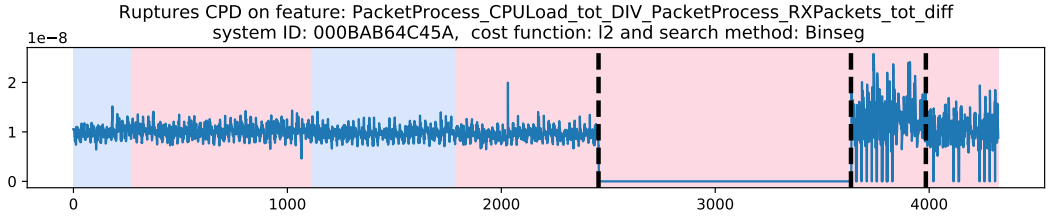

Figure 5.2.9: Ruptures change point detection with 0.0 recall value, 0.0 precision value and 0.0 f1 value.

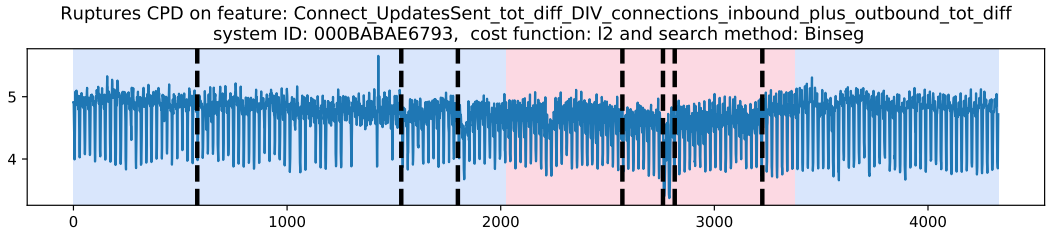

Figure 5.2.10: Ruptures change point detection with 0.0 recall value, 0.0 precision value and 0.0 f1 value.

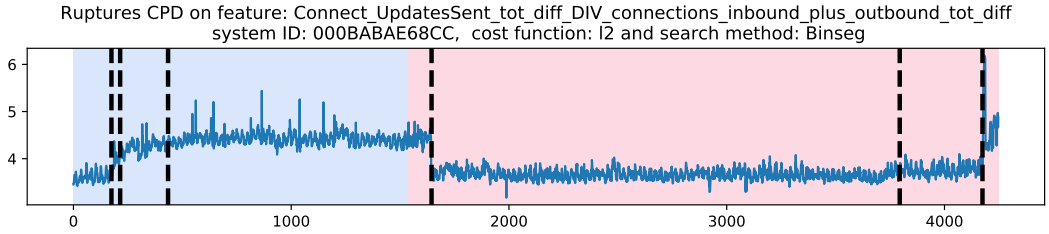

Figure 5.2.11: Ruptures change point detection with 0.0 recall value, 0.0 precision value and 0.0 f1 value.

Observing these graphs led us to believe there are additional external factors that affect the variation because of the large amount of f-change points. For almost every one of the ten figures of Ruptures evaluations provided above, there are at least one f-change point which motivated one additional evaluation run on the same set of features and the same system IDs, but including change points from configuration changes which occurs when the values in a monotonically increasing feature resets to 0. An example of how the configuration change points are found for each system ID is shown in Figure 5.2.14, where the value resets timestamp is noted. These change point are added to the evaluation as s-change points and only the Binary segmentation search method is used,

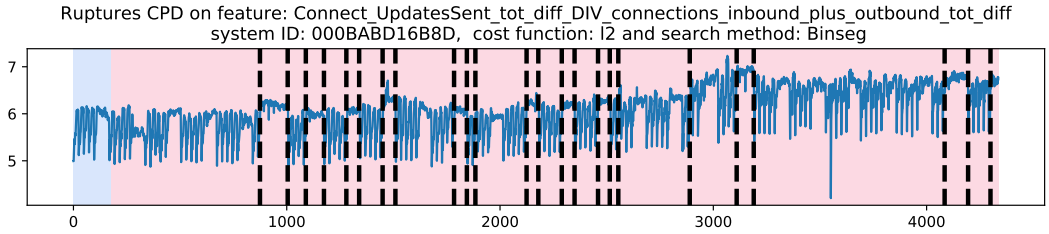

Figure 5.2.12: Ruptures change point detection with 0.0 recall value, 0.0 precision value and 0.0 f1 value.

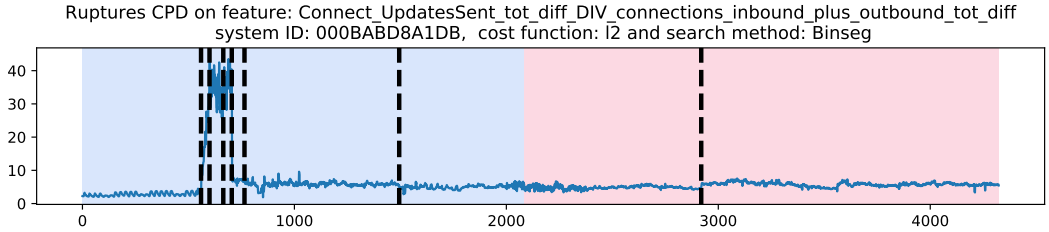

Figure 5.2.13: Ruptures change point detection with 0.0 recall value, 0.0 precision value and 0.0 f1 value.

with l1 and l2 as cost functions. The result can be observed in Figure 5.2.15.

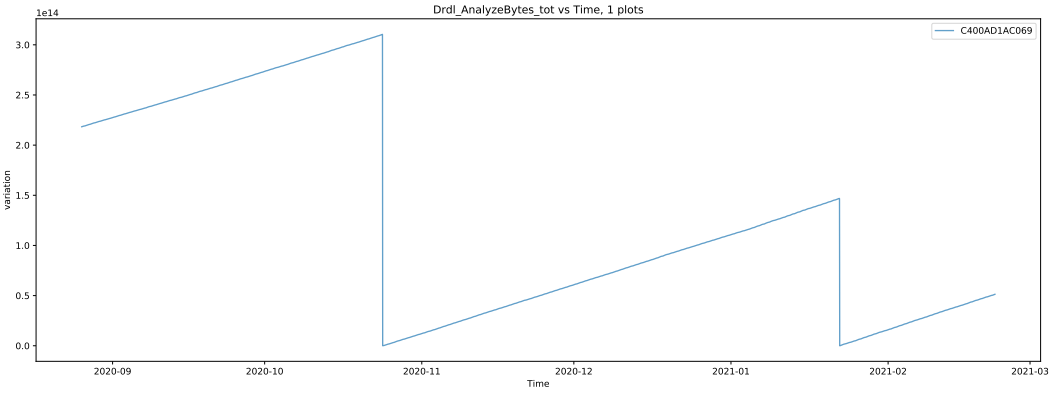

Figure 5.2.14: Illustration of a feature with monotonically increasing values and configuration changes of the system when the values are dropped to 0.

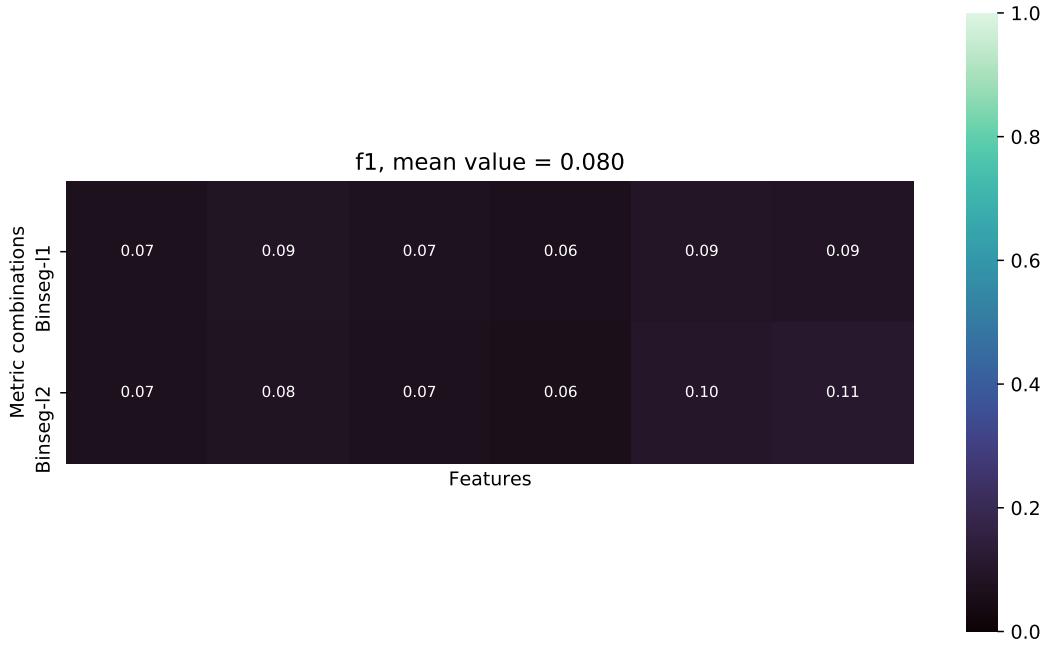

Figure 5.2.15: Illustration of a heatmap with results from the large scale log experiment Ruptures run over all system IDs containing change points including system reboots because of configuration changes. The heatmap displays the f1 value of the Precision and Recall evaluation method. The features examined in this figure can be located in appendix a list 4.

The result of the evaluation with additional change points added (configuration changes), the test seen in Figure 5.2.15 did not significantly improve the results. The average F1-value is still very low and the Precision and the Recall values are just about the same, meaning that the reason for the f-change points are still unknown and needs to be ascertained to derive additional understanding from these evaluations. As a concluding remark for the testing phase, there seem to be too many reasons for variation changes in the data to be limited to only firmware/signature/configuration changes. Also, the firmware/signature/configuration changes does not always affect the variation of the data at all, leading to many false negatives.

Chapter 6

Conclusions, Discussion and Future Work

6.1 Conclusion

In this thesis work we investigated whether or not it is possible to detect changes in the variation of a time series through change point detection methods implemented in Ruptures. The data files used were data logs converted into data frames, with many different features to consider as well as various system IDs representing different equipment. The implemented methods and functions were tested throughout the work with many different settings to filter out the lesser performing combinations. Furthermore, feature manipulation functions were created to counteract anomalies and unwanted changes in the data. Synthetic data were generated to facilitate the experimentation, and all the experiments created were executed on a large variety of sequential data to achieve as accurate and general results as possible. When a satisfactory setting was achieved, both the choice of method and function combinations and parameter settings, a large scale log detection experiment were executed on a data frame with a large amount of system IDs and no synthetic data to evaluate whether or not the initial assumption, that signature/firmware version changes or configuration changes causes significant changes to the variation of the data, was a correct assumption. However, the results concluded that the initial assumption was not incorrect, but not all of the firmware/signature/configuration changes actually affect the variation. It is observable in the various graphs that often at one of these changes, the variation is not affected at all. It is also very clear that at many locations, the variation changes for reasons unknown. An assumption to be made is that there are many other factors affecting the variation that are not considered in this thesis work, which could potentially be part of future work.

6.2 Discussion

The results of the various experiments created in this thesis work were almost always creating insights helpful for further experimentation. Every experiment was created for a purpose, and each result caused us to either change some parameter setting, filter out additional features or change the set of method/function combinations. Moreover, what can be derived from the synthetic data experiments is that good settings are actually achieved considering the tests results on low-noise system IDs were very good and high-noise system IDs were as good as can be expected with high average Recall values, meaning that the change points expected to be found were found (true positives), and a little lower Precision meaning that there are other factors affecting the variation and causing the algorithms to find some false positives.

As for the results from the large scale detection test, both the average Precision value and the average Recall value were low and as a consequence, so was the average f1 value. The reason for these low values is that first of all, all the signature changes and the firmware changes does not significantly alter the variation of the data, resulting in many false negatives. Furthermore, there exists a lot of noise in many of the systems causing big changes in the variation which causes many false positives. To actually be able to understand the evaluations better, it is required to find out what those unknown affecting factors are and include them in the evaluation experiments.

6.2.1 Project evaluation

As for the occurring problems such as the bad value management, the functions created throughout the project were implemented with exception handling where special cases like ”division by zero” or NaN values were managed. Additionally, numerous functions were created to achieve the optimal feature manipulation such as the ”diff” method to remove the monotonically increasing part of the features, the load normalization function, a couple of functions to remove the day/night-cycle variation (which were decided not to be used) and so on and so forth. The feature manipulation functions served their purpose well, all the anomalies and variation changes that we wanted to counteract got counteracted through these functions.

The initial synthetic data Ruptures testing was very instructive and educational and provided the knowledge required to use Ruptures in a structured and easy way. Furthermore, investigating the data logs beforehand and looking at the different features was extremely necessary, not only to understand the layout and the content but also to see and understand the various bad values. For example, in the first data log inspected there was 0s and -1 values in some places where they should not be. This was because when a system is down for some reason, there is no recorded value and instead of a NaN value, it was set to either -1 or 0. This knowledge was crucial to have when creating the methods for feature manipulation, since exceptions has to be handled correctly. Furthermore, all the feature manipulations and plotting was also a very good way of learning about the features and how they acted.

All the tests including generated change point are created to try to achieve the highest possible evaluation values on systems that already have existing change points. Those real change points could affect the evaluation value since they are not accounted for as s-change points, but may cause noise/variation changes. The only change points that are noted as s-change points are the generated ones. Realistically, achieving a f1 value above 0.5 with so much noise would be challenging. As we described and presented on one of the experiments, when a Ruptures evaluation test was made on a system without noise, the average F1-value were around 0.9 which is really good but not at all realistic. However, such results clarify that the best settings are most likely used (or at least good), and the same settings can be used for further evaluations with no generated change points and still produce the best results possible.

There were a lot of ambiguity concerning the generated change points-function settings and how to correctly choose the factor. When generating the synthetic change points they are created with a certain factor which decides how large portion of the mean value of the signal should be added to the continued signal. This setting was, using trial and error, picked to be 0.4 meaning 40 percent of the mean value. It is however not for certain that factor is the most appropriate, but considering a firmware/signature update is assumed to affect the data, and a smaller factor results in a very small change, 0.4 is assumed to be a good factor.

All the different parameter settings changed frequently (at least in the beginning) to optimize the evaluation results and although it is hard to be completely certain whether the best settings are achieved, through trial and error and some logic we can confidently say that good settings were achieved. The penalty settings were the hardest part of all the parameter settings since we had to come up with our own penalty functions through looking at the codes for the specific existing cost functions.

6.3 Future Work

The future work may investigate the different techniques for change point detection by not limiting the amount of tested approaches to the ones existing in Ruptures. There are many other search methods which has not been tested in this thesis work, and there is also many more ways of testing the different parameter settings and penalty functions. Further, there are possibly ways of manipulating the data to remove the different unwanted variations such as the cycle of whether it is day or night. Some methods were tested in this work but they did not seem to help with the change point detection. However, the most important part for future work is to address the unknown f-change points mentioned in this work to be able to evaluate the data further.

Bibliography

[1] Adams, Ryan Prescott and MacKay, David JC. “Bayesian online changepoint detection”. In: *arXiv preprint arXiv:0710.3742* (2007).

[2] Aminikhanghahi, Samaneh and Cook, Diane J. “A survey of methods for time series change point detection”. In: *Knowledge and information systems* 51.2 (2017), pp. 339–367.

[3] Aminikhanghahi, Samaneh, Wang, Tinghui, and Cook, Diane J. “Real-time change point detection with application to smart home time series data”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.5 (2018), pp. 1010–1023.

[4] Anastasiou, Andreas and Fryzlewicz, Piotr. “Detecting multiple generalized change-points by isolating single ones”. In: *arXiv preprint arXiv:1901.10852* (2019).

[5] Arlot, Sylvain, Celisse, Alain, and Harchaoui, Zaid. “A kernel multiple change-point algorithm via model selection”. In: *Journal of machine learning research* 20.162 (2019).

[6] Barnett, Ian and Onnela, Jukka-Pekka. “Change point detection in correlation networks”. In: *Scientific reports* 6.1 (2016), pp. 1–11.

[7] Beaulieu, Claudie, Chen, Jie, and Sarmiento, Jorge L. “Change-point analysis as a tool to detect abrupt climate variations”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 370.1962 (2012), pp. 1228–1249.

[8] *Binary segmentation (Binseg)*. `https://centre-borelli.github.io/ruptures-docs/user-guide/detection/binseg/`. Accessed: 2021-05-05.

[9] *Bottom-up segmentation (BottomUp)*. `https://centre-borelli.github.io/ruptures-docs/user-guide/detection/bottomup/`. Accessed: 2021-05-05.

[10] Boysen, Leif, Kempe, Angela, Liebscher, Volkmar, Munk, Axel, Wittich, Olaf, et al. “Consistencies and rates of convergence of jump-penalized least squares estimators”. In: *The Annals of Statistics* 37.1 (2009), pp. 157–183.

[11] Burg, Gerrit JJ van den and Williams, Christopher KI. “An evaluation of change point detection algorithms”. In: *arXiv preprint arXiv:2003.06222* (2020).

[12] Chen, Hao, Zhang, Nancy, et al. “Graph-based change-point detection”. In: *Annals of Statistics* 43.1 (2015), pp. 139–176.

[13] Cho, Haeran and Fryzlewicz, Piotr. “Multiple-change-point detection for high dimensional time series via sparsified binary segmentation”. In: *Journal of the Royal Statistical Society: Series B: Statistical Methodology* (2015), pp. 475–507.

[14] Fryzlewicz, Piotr et al. “Wild binary segmentation for multiple change-point detection”. In: *Annals of Statistics* 42.6 (2014), pp. 2243–2281.

[15] Garreau, Damien, Arlot, Sylvain, et al. “Consistent change-point detection with kernels”. In: *Electronic Journal of Statistics* 12.2 (2018), pp. 4440–4486.

[16] Harris, Charles R., Millman, K. Jarrod, Walt, St’efan J. van der, Gommers, Ralf, Virtanen, Pauli, Cournapeau, David, Wieser, Eric, Taylor, Julian, Berg, Sebastian, Smith, Nathaniel J., Kern, Robert, Picus, Matti, Hoyer, Stephan, Kerkwijk, Marten H. van, Brett, Matthew, Haldane, Allan, R’ıo, Jaime Fern’andez del, Wiebe, Mark, Peterson, Pearu, G’erard-Marchant, Pierre, Sheppard, Kevin, Reddy, Tyler, Weckesser, Warren, Abbasi, Hameer, Gohlke, Christoph, and Oliphant, Travis E. “Array programming with NumPy”. In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[17] Haynes, Kaylea, Eckley, Idris A, and Fearnhead, Paul. “Computationally efficient changepoint detection for a range of penalties”. In: *Journal of Computational and Graphical Statistics* 26.1 (2017), pp. 134–143.

[18] Killick, Rebecca, Fearnhead, Paul, and Eckley, Idris A. “Optimal detection of changepoints with a linear computational cost”. In: *Journal of the American Statistical Association* 107.500 (2012), pp. 1590–1598.

[19] Lindquist, Martin A, Waugh, Christian, and Wager, Tor D. “Modeling state-related fMRI activity using change-point theory”. In: *NeuroImage* 35.3 (2007), pp. 1125–1141.

[20] Liu, Song, Yamada, Makoto, Collier, Nigel, and Sugiyama, Masashi. “Change-point detection in time-series data by relative density-ratio estimation”. In: *Neural Networks* 43 (2013), pp. 72–83.

[21] Polunchenko, Aleksey S and Tartakovsky, Alexander G. “State-of-the-art in sequential change-point detection”. In: *Methodology and computing in applied probability* 14.3 (2012), pp. 649–684.

[22] Rand, William M. “Objective criteria for the evaluation of clustering methods”. In: *Journal of the American Statistical association* 66.336 (1971), pp. 846–850.

[23] Shewhart, Walter Andrew. *Economic control of quality of manufactured product*. Macmillan and Co Ltd, London, 1931.

[24] Tartakovsky, Alexander G, Polunchenko, Aleksey S, and Sokolov, Grigory. "Efficient computer network anomaly detection by changepoint detection methods". In: *IEEE Journal of Selected Topics in Signal Processing* 7.1 (2012), pp. 4–11.

[25] team, The pandas development. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

[26] Truong, Charles, Oudre, Laurent, and Vayatis, Nicolas. "Selective review of offline change point detection methods". In: *Signal Processing* 167 (2020), p. 107299.

[27] Vrieze, Scott I. "Model selection and psychological theory: a discussion of the differences between the Akaike information criterion (AIC) and the Bayesian information criterion (BIC)." In: *Psychological methods* 17.2 (2012), p. 228.

[28] *Window-based change point detection (Window)*. https://centre-borelli.github.io/ruptures-docs/user-guide/detection/window/. Accessed: 2021-05-05.

Appendix - Contents

Appendix A

Table of features for different experiments

1. Method and function combination filtering experiment
    (a) Connect_UpdatesSent_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff
    (b) Drdl_Key-ValStoreEntriesAdded._tot_diff_DIV_connections_inbound_plus_outbound_tot_diff
    (c) Drdl_AnalyzeBytes_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff
    (d) Drdl_AnalyzeActionsCalled_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff
    (e) PacketProcess_CPULoad_tot_DIV_PacketProcess_RXPackets_tot_diff
    (f) Drdl_NumberOfSliceStateStructuUsed_tot_DIV_Connect_CurrentCount_tot
    (g) Drdl_AnalyzeActionsCalled_rate_DIV_connections_inbound_plus_outbound_tot_diff
    (h) Drdl_AnalyzeBytes_rate_DIV_connections_inbound_plus_outbound_tot_diff
    (i) Drdl_Key-ValStoreEntriesAdded._rate_DIV_connections_inbound_plus_outbound_tot_diff
    (j) Connect_UpdatesSent_rate_DIV_connections_inbound_plus_outbound_tot_diff
    (k) PacketProcess_FreeMemory_tot
2. Experiment on features with obvious visual change points and general experiment on all current system IDs
    (a) PacketProcess_CPULoad_tot
    (b) PacketProcess_RXPackets_tot_diff
    (c) PacketProcess_CPULoad_tot_DIV_PacketProcess_RXPackets_tot_diff
    (d) Drdl_Key-ValStoreEntriesAdded._tot_diff
    (e) Drdl_AnalyzeBytes_tot_diff
    (f) Connect_UpdatesSent_tot_diff
    (g) Connect_UpdatesSent_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff
    (h) Drdl_NumberOfSliceStateStructuUsed_tot
    (i) Drdl_AnalyzeActionsCalled_tot_diff
    (j) Drdl_AnalyzeActionsCalled_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff
3. Log change detection experiments with generated change points
    (a) Connect_UpdatesSent_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff
    (b) Drdl_Key-ValStoreEntriesAdded._tot_diff_DIV_connections_inbound_plus_outbound_tot_diff

(c) Drdl_AnalyzeBytes_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff

(d) Drdl_AnalyzeActionsCalled_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff

(e) PacketProcess_CPULoad_tot_DIV_PacketProcess_RXPackets_tot_diff

(f) Drdl_NumberOfSliceStateStructuUsed_tot_DIV_Connect_CurrentCount_tot

(g) PacketProcess_FreeMemory_tot

4. Log change detection with original change points from system updates/configurations

(a) Connect_UpdatesSent_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff

(b) Drdl_Key-ValStoreEntriesAdded._tot_diff_DIV_connections_inbound_plus_outbound_tot_diff

(c) Drdl_AnalyzeBytes_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff

(d) Drdl_AnalyzeActionsCalled_tot_diff_DIV_connections_inbound_plus_outbound_tot_diff

(e) PacketProcess_CPULoad_tot_DIV_PacketProcess_RXPackets_tot_diff

(f) Drdl_NumberOfSliceStateStructuUsed_tot_DIV_Connect_CurrentCount_tot