Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Networked Systems
ETH Zürich — seit 2015

In Search of Network Shifts

Bachelor Thesis

Author: Fredrik Nestaas

Advisors: Alexander Dietmüller, Romain Jacob

Supervisor: Prof. Dr. Laurent Vanbever

February 2021 to May 2021

Abstract

Machine learning models are becoming increasingly prominent in a range of scientific fields, and with growing popularity also comes an increased need for guarantees and robustness. In particular, many machine learning models are prone to so-called *distribution shifts*, where the underlying probability distribution of the data changes. In this report, we investigate algorithms to detect distribution shifts on network data. We first give an overview of the topic and formulate the problem as placing *breakpoints* in the relevant signal. Thereafter, we test two exact algorithms, *Dynp* and *Pelt* with different cost functions on generated data. We then proceed to investigating real data sets: the FlockLab Sky data set, and a preprocessed CAIDA data set, using the shift detection algorithms. Applying different cost functions, we find that the algorithms are able to detect changes in piecewise constant signals well. Further, we find that the methods generally perform better when more samples are available, and that discarding the data recorded at night from the FlockLab data set makes the breakpoints detected by the shift detection methods agree more with our hypothesis. Additionally, in the CAIDA data set, we explore a connection between high correlation and shift detection among data set features, and attempt to compare the magnitude of short-term and long-term network distribution shifts. We find that the results change noticeably when disregarding certain features that correlate highly with others in the data set, and establish that in order to make conclusion about long- and short-term shifts, we would have needed longer data recordings than those available in the CAIDA data set.

Our analysis indicates that we can probably only be confident about finding network shifts on piecewise constant signals. The methods we use do not perform well otherwise. Thus, we would encourage future research to seek meaningful transformations of data to piecewise constant signals if one wishes to detect shifts on them using the shift detection methods which we use. Alternatively, one could alter the shift detection methods by using different cost functions that perform better on non-piecewise constant signals. Further, the results from the FlockLab data indicate that one should possibly consider data recorded under seemingly similar conditions when detecting network shifts. Thus, another interesting topic for future research could be to determine which factors most heavily influence how similar the conditions during data collection are. In our case, the time of recording seems important.

Contents

Chapter 1

Introduction

Machine learning models are becoming increasingly prominent in a wide range of scientific fields. They are used to solve tasks that computer science has not been able to solve through human design and even achieve super-human performance in e.g. image recognition tasks or anomaly detection. However, with growing popularity also comes an increased need for guarantees and robustness. Many of the machine learning algorithms used today rely on the so-called *i.i.d.* assumption, that is, that the data utilized is independent and identically distributed, both during training and during application. It has been shown that a number of machine learning models are highly prone to distribution shifts. E.g. in [5], state of the art classifiers display poor performance under relatively small perturbations to the sampling probability distributions.

1.1 Distribution Shifts

Changes in a probability distribution are called *distribution shifts*. Mathematically, there are two types of shifts. *Label shifts* are scenarios where the distribution over the *outputs given the same inputs* change, i.e. the probability distribution $p(y|x)$, where y is the output and x is the input, changes. The other type of shifts is called *covariate shifts*, and describes scenarios where *only the input distribution* changes, i.e. $p(x)$ varies with time [4].

Arguably, label shifts are particularly critical to machine learning models, as they attempt to infer the output label y given an input x. In a sense, particularly in generative modeling, these models learn $p(y|x)$ and estimate y based on the distribution on which they trained. Hence, if on a new distribution the same input leads to a different output, the model is no longer valid, and needs to be trained anew on the changed distribution.

There already exists a wide range of research on distribution shifts in different contexts. E.g. in [5], the authors test different methods for detecting a single shift on image data under different perturbations of the probability distribution. However, it is also possible to detect multiple shifts, as distributions may change arbitrarily often.

1.2 Network Shifts

While multiple analysis tools and algorithms exist for time series shift detection, it remains unknown which methods are best suited to analyze network data. Its behavior may be bursty, separating network data from other types of time series, and different dimensions of the analyzed signals may correlate strongly. Furthermore, data points are often missing from real life measurements of network data.

In this report, we aim to explore different distribution shift detection methods applied to network data, discussing their possibilities and limitations. In particular, we assume that the investigated distributions may change multiple times and try to detect these changes using the algorithms *Dynp* and *Pelt*, which are described further in chapter 3. We investigate two data sets, namely a pre-processed data set from the Center for Applied Internet Data Analysis (CAIDA) collected from May 2013 until April 2016 [1], and one collected by Jacob et. al. monitoring the daily wireless link quality of the FlockLab testbed between July 2019 and March 2020 [3].

1.3 Contributions

Concretely, we analyze the FlockLab and CAIDA data sets using the python library *ruptures*, which implements a range of shift detection methods to find *breakpoints* in a signal. These breakpoints separate intervals within which there are no distribution shifts. We use the algorithms *Dynp* and *Pelt* with their available cost functions; the L_1, L_2 and RBF costs. The goals are to analyze how well they are able to detect shifts on different types of signals, and under which conditions, as well as to find some more information about the data itself, whenever possible.

On the FlockLab data, we have some idea of where to expect shifts, making it feasible to test the shift detection methods on the data set. We also make some new discoveries about the FlockLab data using the shift detection methods, confirming a hypothesis set fourth in [2]. In addition to testing the methods on this data, we discuss different approaches to dealing with missing data. Proceeding to the analysis of the CAIDA data, about which we know less, we try to make some statements about long- and short-term network shifts using these methods. Additionally, we investigate the effect of pre-processing the data in different ways, i.e. using low-pass filters and by disregarding multiple highly correlated signal dimensions.

Chapter 2

Background and Related Work

Previous papers have extensively studied distribution shifts both on image data and in time series [8]. Typically, the algorithms detect shifts by quantifying the dissimilarity of different observed data, and their distributions, through some cost function.

2.1 Breakpoints

In order to characterize network shifts, we consider regions which are similar in some sense. To separate these regions, we seek so-called *breakpoints* or *change points* in the signal, such that the samples between two breakpoints are somehow mutually more similar than they are to samples that do not lie between these breakpoints.[1] We consider two different types of shift-detection problems; in one of them, we do not know the number of breakpoints beforehand, and in the other, we do.

For an unknown number of breakpoints, we both have to decide where to put breakpoints, and implicitly, how many breakpoints to assign. Formally, given a signal

$$\{X_i\}_{i\in\{1,\dots,T\}} = \{X_1, X_2, \dots, X_T\} \tag{2.1}$$

we seek a set B of indices b_i:

$$B = \{b_1, b_2, \dots, b_k\} \subset \{1, \dots, T\} \tag{2.2}$$

called *breakpoints* of the signal, such that for some cost function C, the expression

$$\sum_{i=1}^{k-1} C(\{X_{b_i}, \dots, X_{b_{i+1}-1}\}) + p \cdot k \tag{2.3}$$

is minimized. Here, p is a penalty for adding breakpoints, which prevents arbitrarily many breakpoints from being detected.

In the other setting, when we know the number of breakpoints to find, we fix k, and solve

$$B = \underset{B=\{b_1,\dots,b_k\}}{\arg\min} \sum_{i=1}^{k-1} C(\{X_{b_i}, \dots, X_{b_{i+1}-1}\}) \tag{2.4}$$

There are several approaches to solve both of these problems, and [8] gives an extensive overview. We use the python library *ruptures* for the analysis, and a detailed description of the methods can be found in chapter 3.

[1] We will be using the terms *breakpoint* and *change point* interchangeably in this report.

2.2 Data Sets

We analyze two data sets in the report. In the following, we describe each of them, highlight the features that are most interesting to us, and describe their differences.

2.2.1 FlockLab

The FlockLab data set consists of measurements of the wireless link quality in the FlockLab Sky testbed from July 2019 to March 2020. It was collected by Jacob et. al., and is described in detail in [3]. The data collection team measured the wireless link quality by performing tests consisting of 100 transmissions between nodes, and monitoring the number of transmissions that were successfully received [2]. In July, August, September and the very beginning of October 2019, they performed one test every other hour, totalling 12 tests per day, and thereafter, they measured the wireless link quality every sixth hour, resulting in four tests per day.

From these measurements, we create a data set to analyze by considering those transmissions where at least half of the transmitted packets are received, and taking the mean of the number of successful packets. The number of data points per day then corresponds to the number of tests per day. The analysis of the FlockLab data set can be found in chapter 4. It is particularly interesting for testing shift detection algorithms, since we have some idea of what to expect from the data, as is described in chapter 4. In summary, it seems that the wireless link quality is highly correlated with when we expect people to work, indicating that other online applications interfere with the FlockLab testbed. Thus, it allows us to test different shift detection methods and verify whether they agree with the expected results.

2.2.2 CAIDA

The CAIDA (*Center for Applied Internet Data Analysis*) data set consists of processed measurements of different features of internet connections in Chicago between May 2013 and April 2016. It contains measurements of the

- packet count, which is the number of packets in a time window
- TCP count, which is the number of TCP packets in a time window
- UDP count, which is the number of UDP packets in a time window
- IPv4 count, which is the number of packets using native IPv4
- IPv6 count, which is the number of packets using native IPv6
- IPv6 tunnel count, which is the number of packets sending IPv6 using an IPv4 tunnel
- cumulative load, which is the sum of all packets' size, in bytes
- 50, 75, 90 and 95 - percentile loads, which are the packet sizes at the relevant percentile, in bytes
- number of incomplete TCP flows
- average number of packets per flow
- sum of all packet size in one flow, averaged over the flows

- mean packet size in one flow, averaged over the flows

of two directions, A and B, for the relevant links. Each measurement spans approximately one hour, and the measurements are taken on different days, typically spaced by one or three months. For each day, both directions are measured continuously, and in the processed data set, which we use, there are samples at a frequency of 100 samples per second. These samples are extracted from the raw data.

In contrast to the FlockLab data, we do not know what to expect from the CAIDA data set. However, we can analyze it using the shift detection methods described in chapter 3 and applied in chapter 4 to hopefully learn more about the data. The findings are described in chapter 5.

Chapter 3

Shift Detection Methods

As mentioned in chapter 2, we are solving a minimization problem by segmenting a signal into different intervals by so-called *breakpoints* or *change points*, characterizing *network shifts*. There exists a multitude of algorithms to detect shifts in data sets, many of which are described in [8].

In summary, there are two classes of shift detection problems, where we solve for an optimal set B of breakpoints b_i:

- Known number of breakpoints - If we know the number of breakpoints beforehand, we only need to place these breakpoints optimally in order to minimize the cost of the signal X:

$$B = \underset{B=\{b_1,\ldots,b_k\}}{\arg\min} \sum_{i=1}^{k-1} C(\{X_{b_i}, \ldots, X_{b_{i+1}-1}\})$$

  for a fixed number k. C is a so-called *cost function*, which somehow measures the similarity of points in an interval. See Appendix A for a precise definition of the cost functions we use.

- Unknown number of breakpoints - If we do not know the number of breakpoints beforehand, we minimize the *penalized cost* of the signal, such that $B = \{b_1, \ldots, b_k\}$ minimizes the expression

$$\sum_{i=1}^{k-1} C(\{X_{b_i}, \ldots, X_{b_{i+1}-1}\}) + p \cdot k$$

  for an unknown number k. Here, p is the penalty for adding more breakpoints.

Note that if we solve the problem for an unknown number of breakpoints and a cost function C, and the resulting set contains k_0 breakpoints, then solving the problem for a known number of k_0 breakpoints will yield an equally optimal solution. If the minimum is unique, it will yield the same solution.[1]

3.1 Algorithms

For the analysis, we mainly use the python library *ruptures*[2], which implements a large portion of the framework described in [8]. In particular, we extensively use the algorithms

[1] This is easy to see, since the set of minima of $f(x)$ is the same as the set of minima of $f(x) + a$ for any constant $a \in \mathbb{R}$. Setting $a = p \cdot k_0$ yields the desired result, as the cost functions are equal in this case.

[2] `https://centre-borelli.github.io/ruptures-docs/`

- *Dynp* - An algorithm which solves for a given number of breakpoints with respect to the defined cost function, using dynamic programming.
- *Pelt* - An algorithm which solves for an unknown number of breakpoints in linear time.[3]

together with the cost functions

- C_{L_1} (or L_1 *cost*) - which penalizes points deviating from the median by the L_1 norm of the discrepancy.
- C_{L_2} (or L_2 *cost*) - which penalizes points deviating from the empirical mean by the L_2 norm of the discrepancy.
- C_{RBF} (or *RBF cost*) - which searches for changes in the distribution function of the underlying data by use of kernel mean embeddings. The kernel embedding of a probability distribution using the *RBF* kernel is injective, meaning that two probability distributions are equal if and only if they map to the same kernel mean embedding. For a proof of this, see [7].

All of these cost functions are rigorously defined in Appendix A. We choose to use these algorithms because they solve the problems optimally, in contrast to other options in *ruptures* that only approximately solve the problems, and the mentioned cost functions are precisely those available to *Pelt* and *Dynp*.

3.1.1 Parameter Selection

There are a couple of important parameters for *Dynp* and *Pelt*, which heavily influence the detected breakpoints. At initialization, one can set the parameters *model*, *min_size* and *jump* for both algorithms, which respectively define the utilized cost function, the minimum allowed number of samples between breakpoints, and a number that has to divide each breakpoint.[4] The latter two are by default set to 2 and 5, respectively, and we do not modify them in this section. In chapter 4, however, these parameters will be important.

What we do vary in these tests are the *model*, i.e. the cost function applied with each algorithm. Furthermore, when fitting breakpoints to a specific signal, we set the penalty *pen*, in the case of *Pelt*, and the number of detected breakpoints *n_bkps*, in the case of *Dynp*. As mentioned above, since *Dynp* and *Pelt* are optimal algorithms, for a given number of breakpoints, they will find the same set of breakpoints if the optimal solution is unique, and the other parameters are also the same. On the one hand, *Pelt* is an asymptotically faster algorithm than *Dynp*, making it interesting to use in terms of efficiency. On the other hand, selecting the *pen* parameter correctly may not be straight-forward, as its value can impact the number of detected breakpoints severely. See section 3.5.1 and figure 4.3.

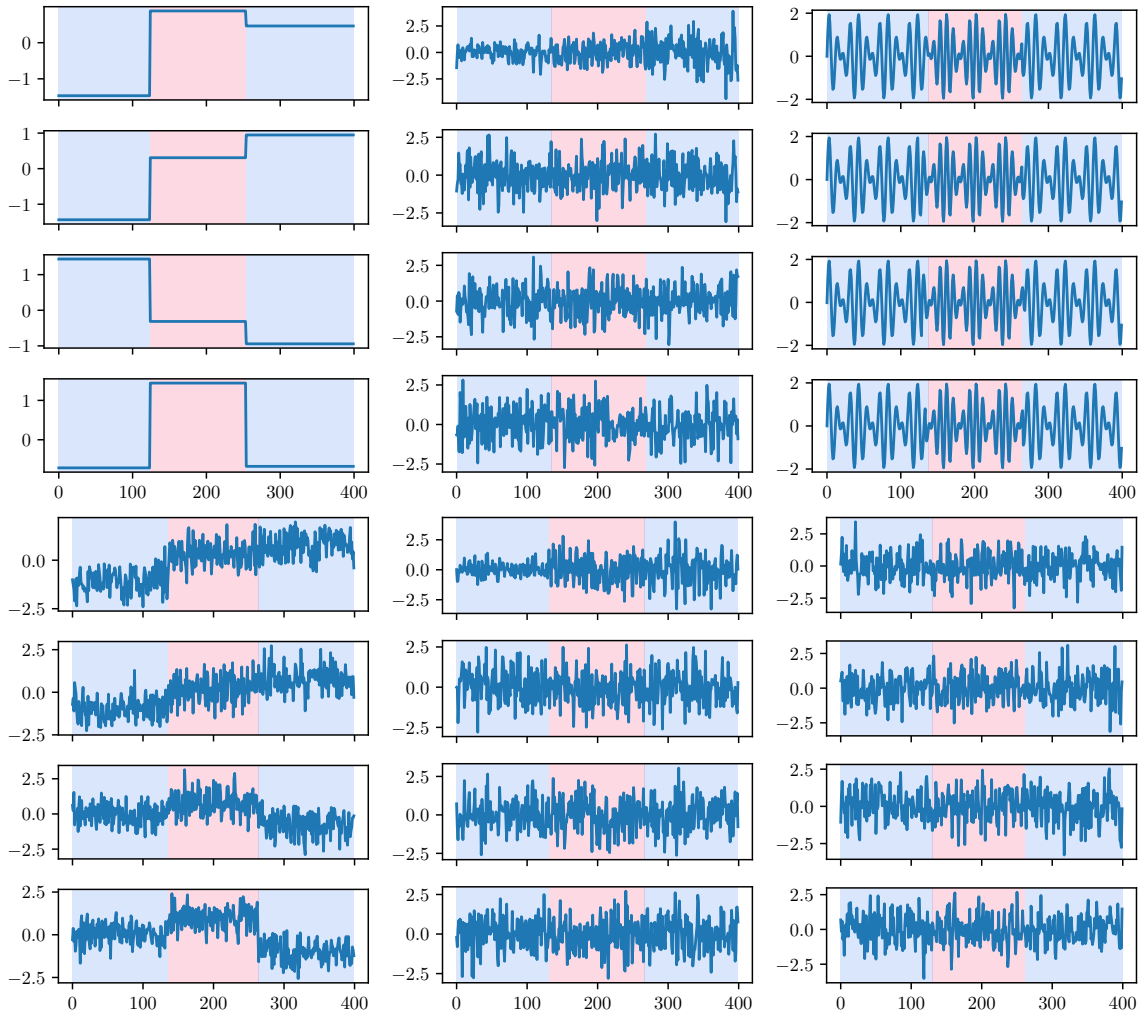
3.2 Data Generation

We test the methods on different types generated data of with varying levels of noise. Concretely, we generate

- Piecewise constant (*pwc*) signals - which have constant values between breakpoints.

[3] *Pelt* stands for *Pruned exact linear time*. Detalils on the implementation can be found in [8]

[4] For clarity, if $jump = 5$, then each breakpoint b_i is divisible by 5. Presumably, this parameter is set for performance reasons, as it greatly limits the search space for the set of breakpoints B.



(a) Noiseless and noisy piecewise constant signals.

(b) Noiseless and noisy piecewise linear signals

(c) Noiseless and noisy piecewise wavy signals.

Figure 3.1: Examples of (normalized) generated signals. The transition between shaded regions indicate the true breakpoints.

- Piecewise linear (*pwl*) signals - which are piecewise linear combinations of Gaussian inputs to a linear regression model.
- Piecewise wavy (*pww*) signals - which are the sum of two sines switching their frequencies at each breakpoint.

These signals are already implemented in *ruptures*.[5] Figure 3.1 shows examples of such signals, containing 400 points and 2 breakpoints.

[5]See `https://centre-borelli.github.io/ruptures-docs/user-guide/` for a more complete description.

3.3 Evaluation

For comparing a set of detected breakpoints to the true set of breakpoints, *ruptures* offers the following metrics:

- Hausdorff-index, which measures the largest number of data points between any detected breakpoint and its closest true breakpoint.
- Randindex, which is a metric between 0 and 1 computed from which breakpoints belong to the same interval. It is computed as

$$\frac{\sum_{i<j} \mathbb{1}(\hat{A}_{i,j} = A_{i,j})}{T(T-1)}$$

  where $\hat{A}_{i,j}$ and $A_{i,j}$ are the indicators for the true and estimated breakpoints of whether the samples X_i and X_j belong to the same regime [8].
- Precision, which measures what fraction of the detected breakpoints are actual breakpoints (within a settable margin).
- Recall, which measures what fraction of actual breakpoints are detected (within a settable margin).

The different metrics have their own pros and cons. For example, the Hausdorff-index makes a strong statement if it is small, as this means that all of the detected breakpoints lie within some margin of the true breakpoints, but if it is large, we do not know whether only one breakpoint was incorrectly detected, or if there are multiple bad estimates. In our experiments, the Randindex was also tendentially high, making it difficult to make strong statements from it. As for precision and recall, if we can be sure that we have selected a good margin, they do make a full statement about the accuracy of the breakpoints when considered together, but it may not be clear how to choose the margin properly. Additionally, precision and recall are designed for binary decisions; whether there is a breakpoint in a given interval or not. However, we found that if there are multiple detected breakpoints within the margin of a true breakpoint, the computed precision and recall suffer.

3.4 Pre-processing the Data

Before applying search methods to the relevant data, we normalize it in order to facilitate comparison across dimensions. It is important that the signals analyzed have the same scale in order to obtain consistent results with the applied search methods. This also goes for individual dimensions of a single signal. For example, if we apply the same penalty to two signals which are merely scaled versions of each other, we may obtain different breakpoints, in the case of an unknown number of breakpoints. We would have to scale the penalty as well to obtain the same result; see Appendix B for a proof. However, as is also shown in Appendix B, only the scale of the signals matter; additive shifts to a signal do not change the detected breakpoints. Bearing these results in mind, we will only investigate two approaches to scaling the data:

- No scaling - for the FlockLab data set, we only analyze one dimension, so we can select proper parameters for *Pelt* and *Dynp* without scaling the signal.

- Standard scaling - for the generated and CAIDA data, we bring each dimension to zero mean and unit variance by applying the transformation

$$X_{scaled} = \frac{X - \mu_X}{\sigma_X}$$

  where X is the signal to be scaled, with mean μ_X and standard deviation σ_X. This is done to avoid over-weighting certain dimensions, as the signals then have the same variance.

Further, we apply a low-pass filter to the CAIDA data in order to reduce the effect of noise, and therefore also test this method on the generated data. For the FlockLab data set, we also sometimes have to deal with missing data, and use the following imputation methods:

- *Interpolation*: Linear interpolation of missing data points according to their timestamps.
- *KDE-filling*: Estimate the distribution of data and filling the missing data points by sampling from the estimated distribution.

We defer showing these methods to chapter 4. The reason why we choose two different filling methods is that if we want to search for phenomena that occur in a signal, we need to be careful not to detect artifacts of the used filling method. Finding the same results for two different filling methods makes it more likely that the results stem from the data itself, and not the imputation.

Lastly, in order to reduce the dimensionality and to avoid essentially duplicate data, we analyze which dimensions of the CAIDA data set correlate particularly strongly, and represented these using only one of the dimensions. The effect of this is shown in section 5.2.

3.5 Testing the Methods on Generated Data

In the following analysis, we use generated signals with 800 points and 20 breakpoints using *Pelt* and *Dynp* with different cost functions. To evaluate the results, we use the Hausdorff-index, Randindex, precision and recall, as described in section 3.3.

3.5.1 Results for *Pelt*

Figures 3.2 and 3.3 show the metrics for changepoint detection on noiseless signals and low-pass filtered noiseless signals, respectively. The results for the noisy signals can be found in the Appendix C.1. In the following, we only discuss the noiseless case, since the results with and without noise are similar.

We see that *Pelt* only has high precision and recall on piecewise constant signals, where they are both equal to 1 for all penalties when using the L_1 cost, and for a number of penalties when using the L_2 and RBF cost functions. For the other signals, the performance is quite poor for all models. This is unsurprising for the L_1 and L_2 costs, since they respectively compute deviations from the median and empirical mean, and one can see in figure 3.1 that the *pwl* and *pww* signals do vary quite a bit more than the *pwc* signals. Further, using the RBF cost does not resolve the problem, evidently.

We also see that the Hausdorff- and Randindex can be somewhat misleading in this setting. For example in figure 3.2, the results for any of the cost functions are arguably bad for the *pww* and *pwl* signals, but this is only visible if we consider the precision and recall together. Upon closer inspection, one might notice that the Hausdorff-index is close to the average length of a breakpoint

interval, which is 40, since we use 800 data points and 20 breakpoints. However, the Randindex does not seem to deliver much valuable information in this case, as it is mostly above 0.95.

These observations indicate that we should try to find (nearly) piecewise constant signals to be able to hope for good performance using *Pelt*. E.g. in the *pww* case, this could have been done by a Fourier transform of the signal, since this would result in large components for certain frequencies, and 0 for other frequencies, and thus approximately be piecewise constant.[6]

Further, for the piecewise constant signal, there seem to be different sweet spots for the selected penalty, depending on the model we use. For too low penalties, *Pelt* tends to find too many change points and has low precision, while for high penalties, the algorithm finds too few breakpoints and the recall suffers. We see that *Pelt* with the L_1 cost achieves perfect precision and recall using all the selected penalties, *Pelt* using L_2 gets perfect scores for $pen = 5$, and *Pelt* with RBF achieves perfect score for $pen = 1$. The reason for the difference is presumably that the L_2 cost function is unbounded for a given point, while the RBF cost function has to be smaller than 1 for each point,[7] making it possible for the L_2 cost to produce higher costs that must be penalized accordingly. E.g. if we apply the same penalty with the L_2 and RBF costs, then we might expect the same interval to have a higher cost using the L_2 cost, making it better to split the interval at the cost of one extra breakpoint in the L_2 case, but not in the RBF case.

Figure 3.3 and the Appendix C.1 indicate the effect of applying a low-pass filter to the noiseless and noisy signals, respectively. Evidently, the quality of the detected breakpoints suffers in these tests for all cost functions. A possible reason for this is the chosen window size of the signals, which is set to 20 in this case. If we accept any breakpoint as true if it occurs relatively close to a true breakpoint, but is somewhat off due to a low-pass filter, it might make sense to increase the margin to obtain representative results when using precision and recall. For instance, in a noiseless *pwc* signal, applying a low-pass filter will result in "ramps" at the breakpoints, so we may accept any detected breakpoint lying on the ramp as a true breakpoint, since the ramp is part of the transition between the true intervals. In our experiments, the margin is ± 10, which is half of the applied window size, and possibly too small.

3.5.2 Results for *Dynp*

As mentioned, if *Dynp* and *Pelt* find the same number of breakpoints, they will in fact find the same breakpoints, given that the optimal solution is unique. Therefore, one could expect that the results for *Dynp* are similar to those obtained for *Pelt*.

However, the algorithms are fundamentally different in the sense that we select the number of breakpoints to find in the case of *Dynp*. For 20 breakpoints, we see that in both the noiseless and in the noisy setting in figure 3.4 and the Appendix C.2, the algorithm finds the correct breakpoints for the *pwc* signal for all cost functions. Similarly to the case of *Pelt*, encouraging more breakpoints increases the recall and reduces the precision, which may be because detecting too many breakpoints will mean that we tend to detect more of the actual breakpoints, and that a smaller fraction of the detected breakpoints are actual breakpoints. This phenomenon is not relevant for the *pwc* signals, since we find the correct breakpoints when predicting the correct number of breakpoints, but we see that the recall does not decrease for any signal with an increasing number of breakpoints in figure 3.4, although it is never equal to 1.

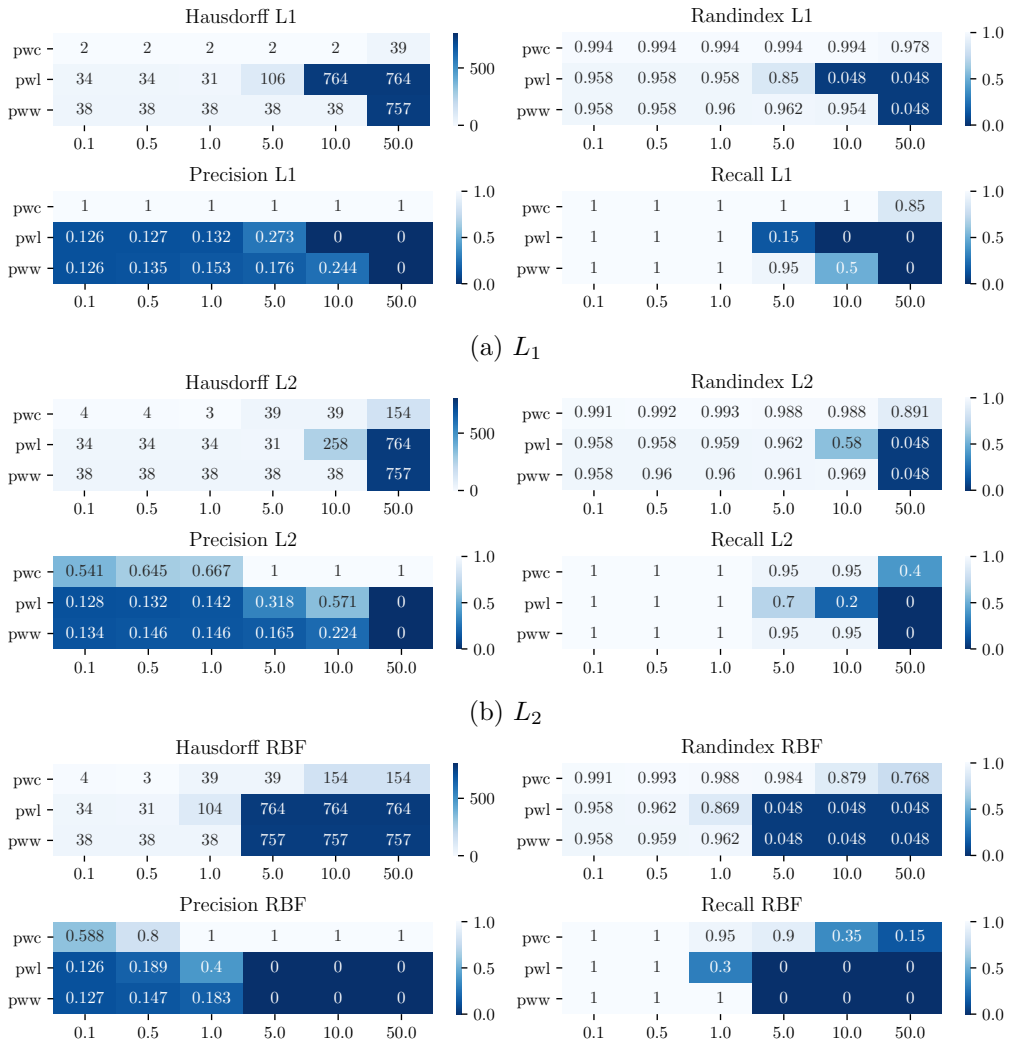
We also see that the metrics for *Dynp* rarely reach the theoretical minimum value. *Pelt* often

[6] Ideally we would know the possible frequencies between which the signal can switch, since we could then correlate the signal with these frequencies to receive a perfectly piecewise constant signal.

[7] Since we are dealing with the square of $exp(-f(x,y)) - \hat{\mu}_{exp(-f)}$ for non-negative $f(x,y)$, the exponential and mean are in the interval $(0,1]$, and the square of their difference is then in the interval $[0,1)$.

would get a precision or recall of 0 for high penalties. In the case of *Dynp*, however, this does not seem to be the case. A possible reason for this, is that for too high penalties, *Pelt* will predict no breakpoints, and thus achieve the minimum scores, whereas *Dynp* always assigns the given number of breakpoints in an optimal fashion.

Figure 3.5 shows that, as with *Pelt*, the precision and recall of *Dynp* decreases for low-pass filtered signals. The relevant discussion in 3.5.1 also applies here.

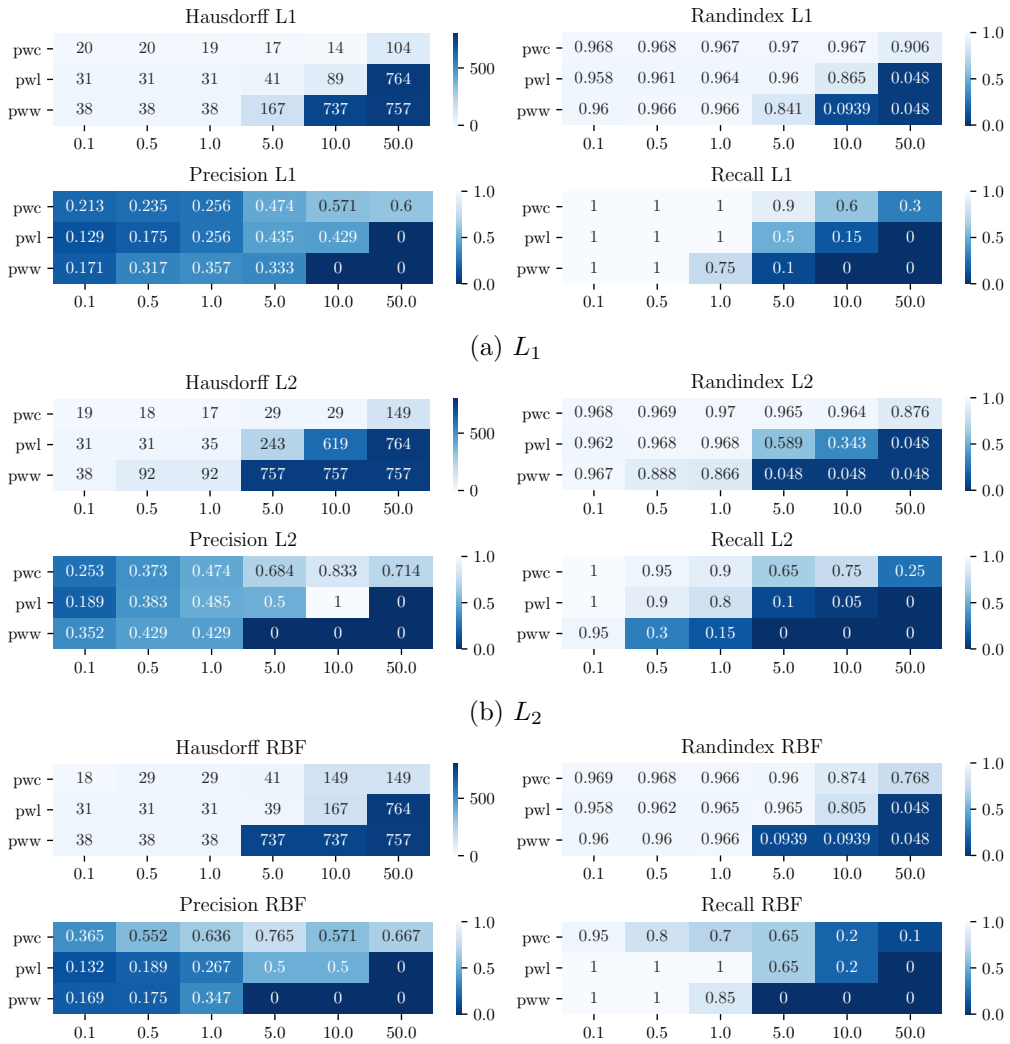


(a) L_1

(b) L_2

(c) *RBF*

Figure 3.2: Results for noiseless signals using *Pelt* with different cost functions. The horizontal axis indicates the utilized penalty.

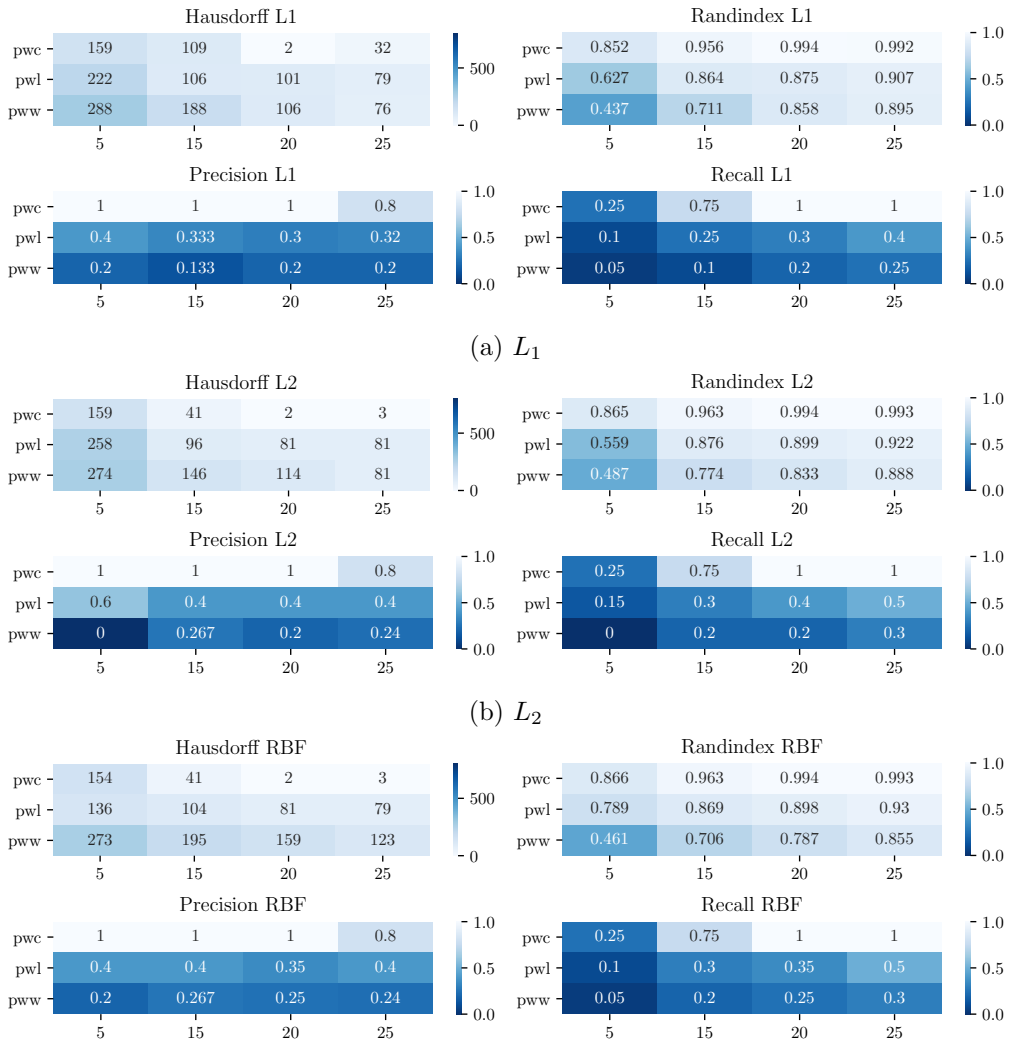


(a) L_1

(b) L_2

(c) *RBF*

Figure 3.3: Results for low-pass filtered noiseless signals using *Pelt* with different cost functions. The horizontal axis indicates the utilized penalty.



(a) L_1

(b) L_2

(c) *RBF*

Figure 3.4: *Dynp* results for noiseless signals with different cost functions. The horizontal axis indicates the number of detected breakpoints.

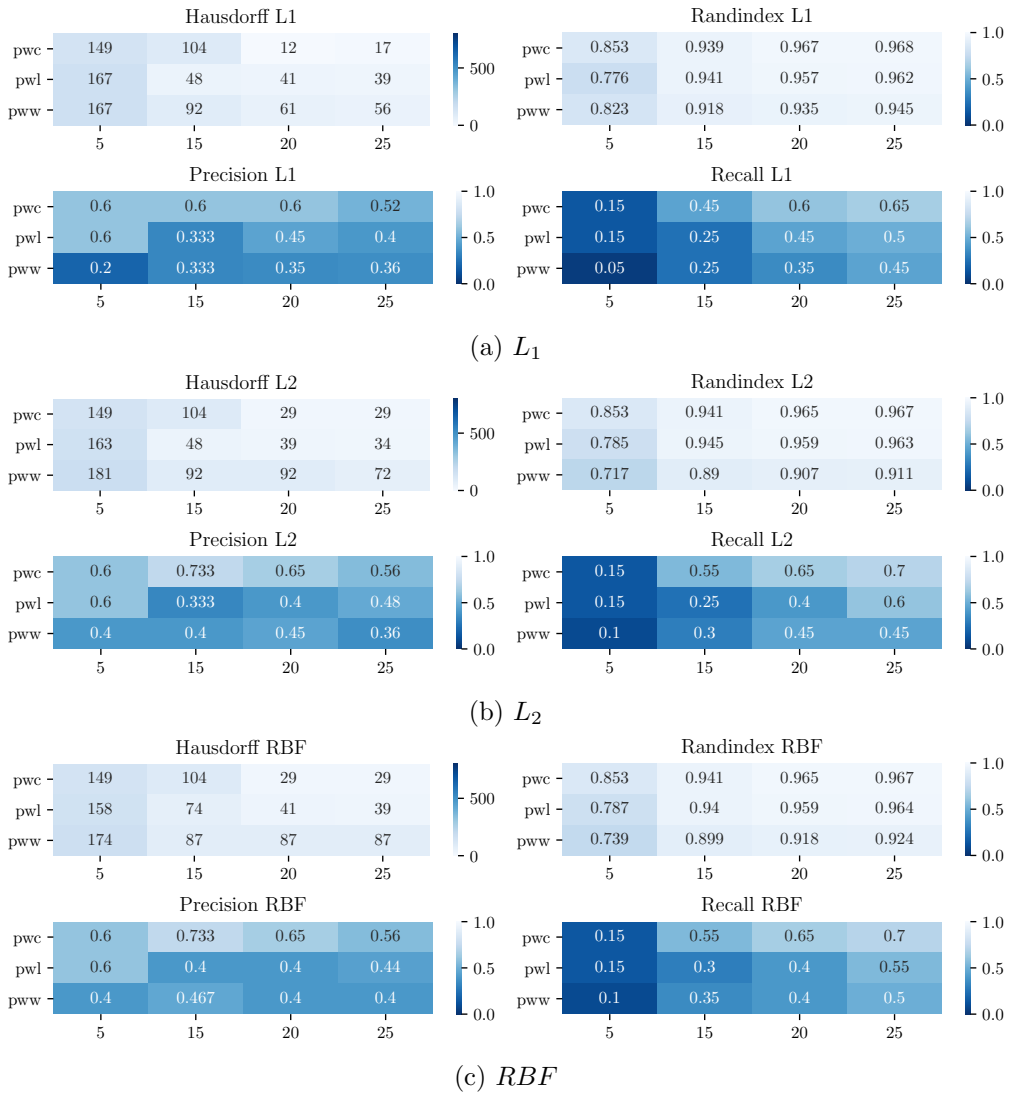


Figure 3.5: *Dynp* results for low-pass filtered noiseless signals for different cost functions. The horizontal axis indicates the number of detected breakpoints.

Chapter 4

FlockLab

The FlockLab Sky data set, which we simply refer to as the FlockLab data set, was collected by Jacob et. al. and is available in [3]. It consists of measurements of the wireless link quality of the FlockLab testbed over eight months, from July 2019 to March 2020, where the wireless link quality was initially tested every two hours, and after three months, every six hours. More details regarding the testing can be found in [3].

For our analysis, we use the data where at least half of the transmissions in a test are successful, and generate data points by taking the mean of the corresponding number of received packets for each test. This results in 12 data points per day for the first three months of the data, and four data points per day thereafter, when there is no missing data. Indeed, data is occasionally missing, and for some of our methods, we need to fill the missing data points. See 4.2 for more details.

4.1 Seasonality

One can see with the naked eye that the data set exhibits some degree of periodicity, as is shown in figure 4.2a and is visible from the autocorrelation plots in figure 4.1.[1] In the autocorrelation plots, we consider all the data recorded using four samples per day, and plot the autocorrelation for the first 112 lags. Jacob et. al. suggested that the FlockLab testbed is likely disturbed by others using the internet nearby, causing interference with the transmissions [2].

It seems from figure 4.2a that the periodicity is weekly, and that the transmissions taking place on the weekends normally have a higher success rate than those taking place on weekdays. Keeping in mind that there are four samples per day, the autocorrelation plot in figure 4.1 confirms this; there are peaks at multiples of four, indicating some degree of daily seasonality, and large peaks at multiples of 28, indicating weekly periodicity.[2] Indeed, this agrees well with the hypothesis from [2], since there would be more internet activity when people work on weekdays, causing less interference during the weekend than during the weekdays. With the apparent seasonality in mind, we have some idea of what to expect from the FlockLab data; breakpoints separating the weekends from the weekdays, as the wireless link quality seems to vary accordingly.

The data set lends itself particularly well to investigating how well we can detect a presumably known number of breakpoints, as we have some evidence indicating a weekly periodicity. However, we may also test algorithms solving for an unknown number of breakpoints and investigate for which

[1] To generate the autocorrelation plots, we need to impute missing data points, as the autocorrelation function we use to generate the plot assumes equidistant samples. See section 4.2.

[2] And even one peak on either side of these peaks, confirming that the wireless link quality is indeed different over *two* days, presumably Saturday and Sunday.

parameters the search methods agree with our hypothesis. Ideally, we may expect to find that the search methods distinguish the weekends well from the weekdays for a wide range of parameters and cost functions. Considering the manageable size of our data, it is feasible to apply an exact algorithm like *Pelt* for an unknown number of breakpoints, or to solve the problem for a known number of breakpoints with the *Dynp* algorithm. We use both approaches and describe the results in the coming sections.

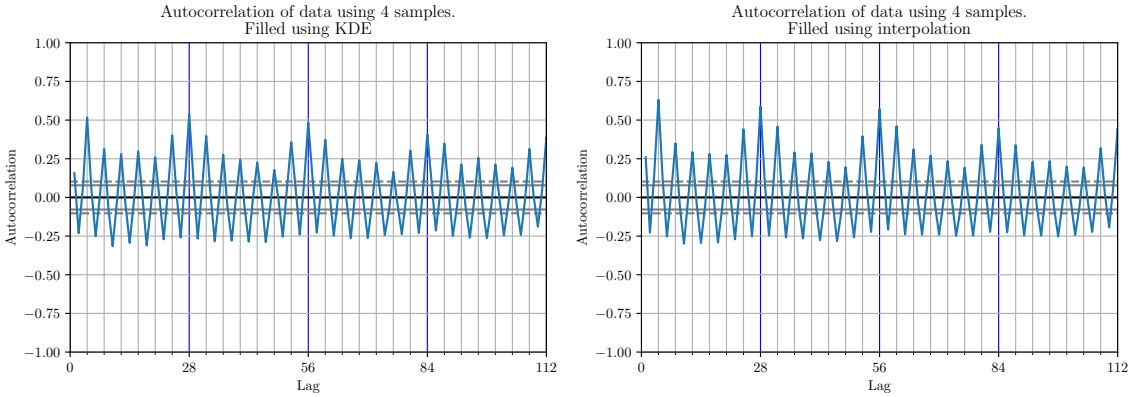


Figure 4.1: Autocorrelation plot filling the data using kernel density estimation (left) and interpolation (right) (see section 4.2). We see that in both cases, there is significant autocorrelation at lags that are multiples of four, indicating daily periodicity, as we used four samples per day for the wireless link quality. Further, the large autocorrelations at multiples of 28 indicate a weekly periodicity.

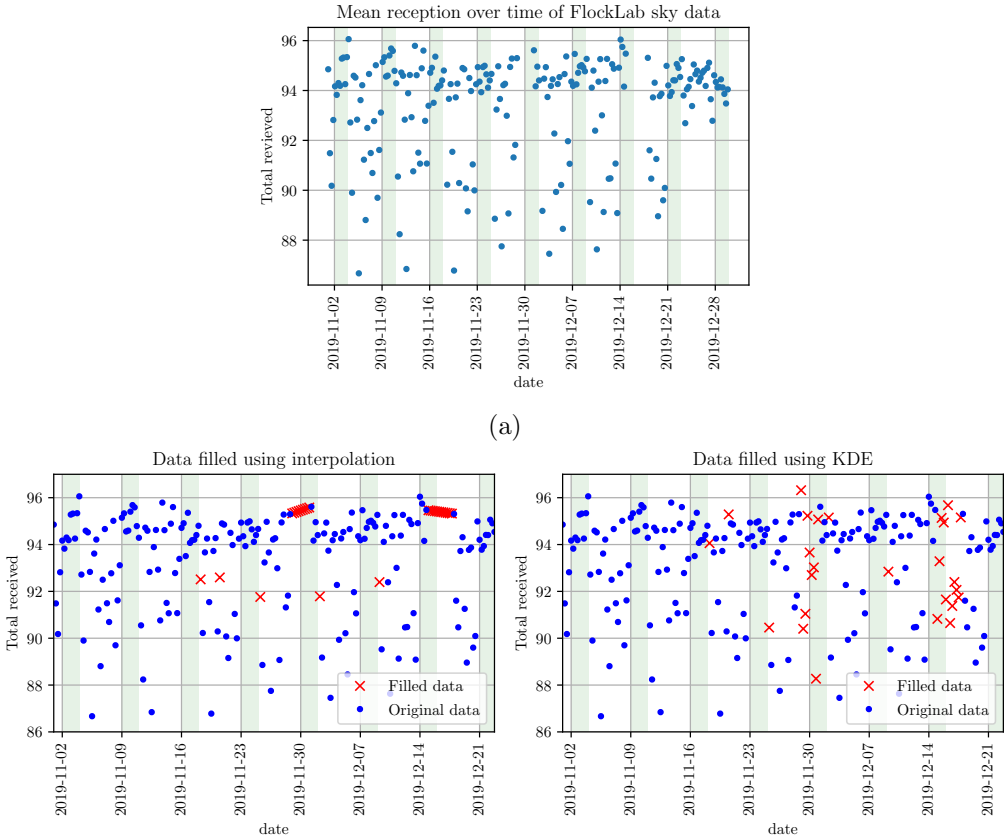


Figure 4.2: (a): FlockLab Sky data (b) filled with interpolation and (c) kernel density estimation to ensure the same number of samples per day. The shaded regions in the plots indicate the location of weekends, and the blue and red points indicate original and filled data, respectively.

4.2 Missing Data

About 20 percent of the data is missing in the FlockLab data set, and some of the functions we use for our analysis do not cope well with missing samples, as they assume that the samples they are given are equidistant. An example is the autocorrelation plot. Furthermore, in Appendix D.4 there is an example where *ruptures* assumes equidistant samples, i.e. we need to fill the missing data to use it for change point detection. Doing so allows us to detect breakpoints only on the boarder between days. Hence, in order to apply these methods, we need techniques to impute missing data.

Generating samples and treating them as true data comes with a range of pitfalls. For example, if we have missing data and simply repeat the last seen sample in order to fill it, we will induce autocorrelation in the data. If we interpolate, the data in a given interval may seem incorrectly noiseless, and so on. Therefore, in order to make it more likely that the phenomena we observe are not artifacts of the filling methods, we employ both KDE-filling and interpolation to impute the missing data points. These methods are both described in 3.4. If a phenomenon occurs for several filling methods, it makes it more likely that it is a property of the underlying true data, and not of the imputation strategies.

E.g. in the case of the autocorrelation function in figure 4.1, we only make conclusions about seasonality because it seems present using both filling methods. In our view, filling the data "randomly" using KDE and by "connecting the dots" using interpolation are two rather different approaches. Yet, in both cases, there is seemingly heavy autocorrelation in the FlockLab data set, which strengthens our confidence in the conclusion.

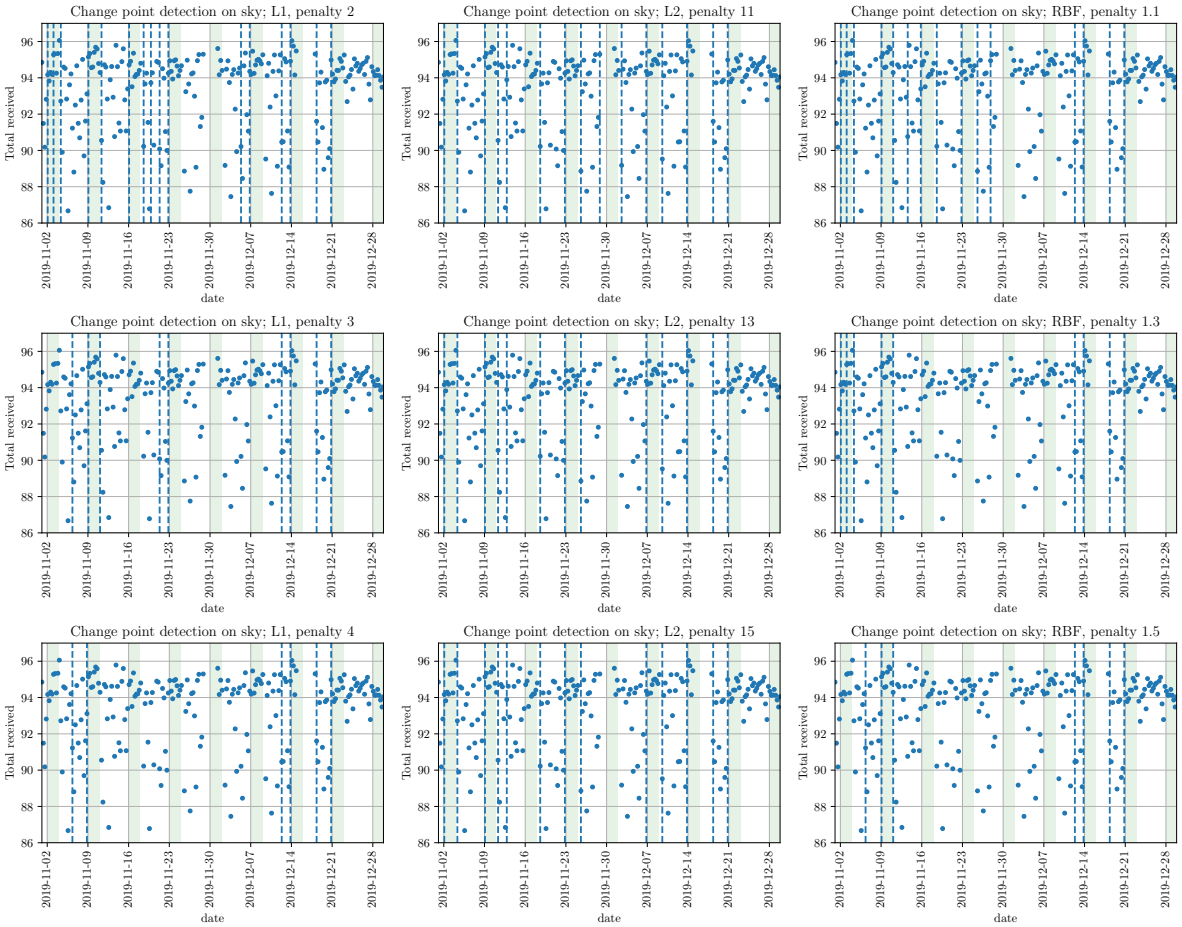
4.3 Applying *Pelt* to the Full Data

Considering the discussion in section 4.1, one might hypothesize that applying *Pelt* to the FlockLab data will result in a clear separation of the weekdays and weekends for a wide range of selected penalties, using any cost function. In the following, we will say that an algorithm *performs well* if it clearly separates the weekdays from the weekends, i.e. matches our hypothesis well. We justify this formulation by considering the autocorrelation plot in figure 4.1 and the discussion in section 4.1. Furthermore, a search method is *robust* if the detected breakpoints do not vary significantly with small changes in the selected penalty parameter.

Figure 4.3 shows representative results of applying *Pelt* to the FlockLab data from November and December 2019. In this figure, we restrict the minimum number of samples between two breakpoints to be at least four, i.e. that each interval has the size of at least one day, as the results without these restrictions are challenging to interpret. For the results without this restriction, see Appendix D.1. We see that indeed, most of the detected breakpoints lie close to the weekends, strengthening the hypothesis that the reception is different on the weekends compared to the weekdays. This is especially true for the L_2 cost, as can be seen in the relevant figures (middle column). However, in particular for the L_1 and RBF costs, the detected breakpoints are sensitive to changes in the selected penalty. Furthermore, occasionally the algorithms find breakpoints in the middle of the week, even when there is no obvious, visible change.[3]

A possible reason for the penalty sensitivity and disagreement with our hypothesis, could be that there are not enough measurements available per day in the November and December data. Indeed, if we consider measurements from August and September 2019, the performance of the

[3]Note that e.g. between December 14th and December 21st, breakpoints are detected in the middle of the week, but this might be attributed to missing data. To the algorithm, if data is missing, the samples on either side of the chunk of missing data seem to be consecutive. See figure 4.3.

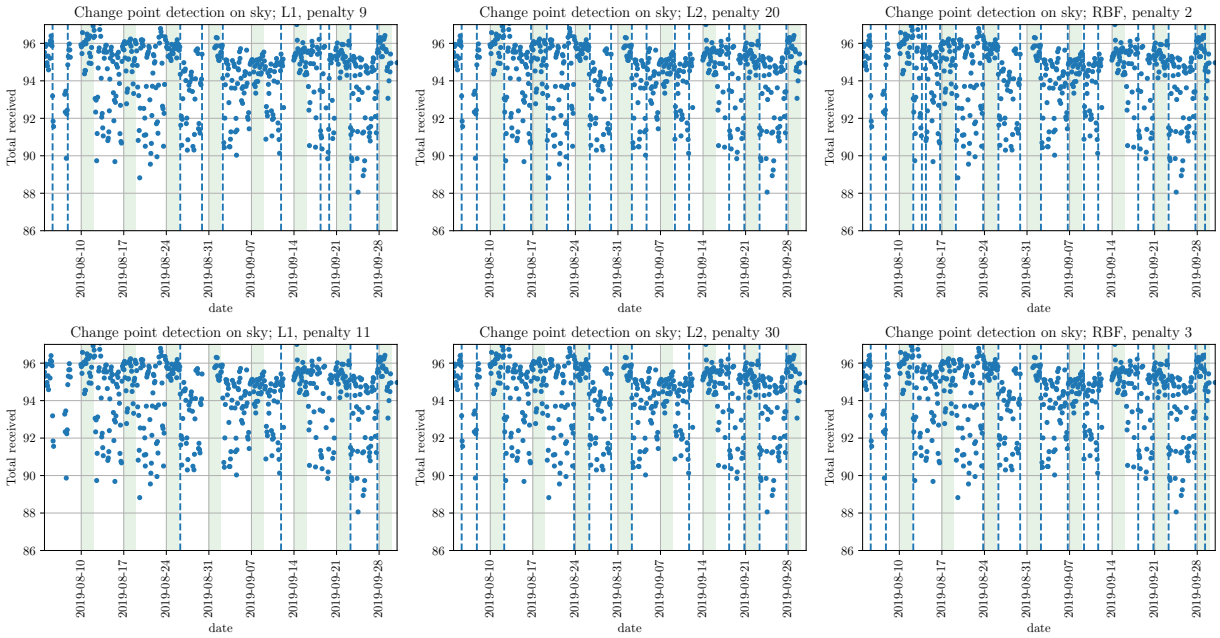


(a) *Pelt* change point detection using L_1 for different penalties. We see that the search method is not too robust, and that the detected breakpoints to not agree too well with our hypothesis.

(b) *Pelt* change point detection using L_2 for different penalties. We see that the search method is more robust than when using the L_1 cost. The breakpoints seem to agree more with our hypothesis.

(c) *Pelt* change point detection using RBF for different penalties. We see that the search method is not robust. The breakpoints also do not agree too well with our hypothesis.

Figure 4.3: Results of applying *Pelt* to the November and December data using different cost functions with different penalties, requiring at least four samples between each breakpoint. The results without this restriction are available in Appendix D.1



(a) *Pelt* change point detection using L_1 for different penalties. We see that the algorithm still does not separate the weekend from the weekdays all too well, and that the algorithm is still not very robust.

(b) *Pelt* change point detection using L_2 for different penalties. We see that the breakpoints still match the edges of the weekend well, and that the robustness is quite good.

(c) *Pelt* change point detection using RBF for different penalties. We see that the breakpoints match the weekends better using more samples per day, and that the robustness has improved compared to the November and December case.

Figure 4.4: Results of applying *Pelt* to the August and September data using different cost functions with different penalties. The minimum size is restricted to eight in these plots, which is less than the 12 samples per day. The choice is a result of missing data.

search method using the RBF cost does improve, but using the L_1 cost, *Pelt* still does not separate the weekends particularly well. This is shown in figure 4.4, where we restricted the minimum size to be eight,[4] and the results without the restriction can be found in the Appendix D.1. Note that when we extend the length of the signal, we tendentially need to increase the penalty. This is because each point contributes with a non-negative cost, meaning that if there are more points in an interval, for a constant penalty, the relative penalty per breakpoint decreases with the length of the interval.

In conclusion, we see that *Pelt* with the L_2 cost seems to do a fairly good job of distinguishing the weekends from the weekdays. The performance becomes more robust when more datapoints are available, in the sense that the detected breakpoints vary less. Given enough data, also *Pelt* used with the RBF cost seems to perform well based on the August and September data. However, the algorithm does struggle to separate the weekdays and weekends well when using the L_1 cost, even when more samples are available.

[4]We tried using 12, the number of samples per day, but it did not work too well, presumably due to missing data

4.4 Applying *Dynp* to the Full Data

Applying *Dynp* to the full FlockLab data leads to quite similar results as when applying *Pelt*. The results for the August and September data are shown on the left hand side in figure 4.5, and for the November and December data, on the left hand side in figure 4.6, using 10 breakpoints in both cases.[5] We show these results, as *Dynp* generally finds breakpoints at similar locations as *Pelt* when using less breakpoints, and the case of 10 breakpoints showcases that the algorithm sometimes identifies the beginning and end of the weekend as breakpoints, and sometimes chooses other breakpoints.[6] As both *Pelt* and *Dynp* are exact algorithms, it is not too surprising that the detected breakpoints are similar. Given the same costs, if both *Pelt* and *Dynp* detect a given number of breakpoints, and the optimal set of breakpoints is unique, they will both find it. An interesting takeaway from this, could be that if we want to enforce a certain number of breakpoints, we can apply *Dynp* without any parameter selection, except choosing the cost function, and receive the same results as we would for *Pelt* using this number of breakpoints. And vice versa, if we find a given number of breakpoints using *Pelt*, these breakpoints could have been found using *Dynp*.

[5] The plots on the right are treated in section 4.5

[6] The number 10 is arbitrary. The results for a given number of breakpoints is similar in all cases with up to 16 breakpoints. For the results using 1, 2, ..., 16 breakpoints on the August data, see Appendix D.3.

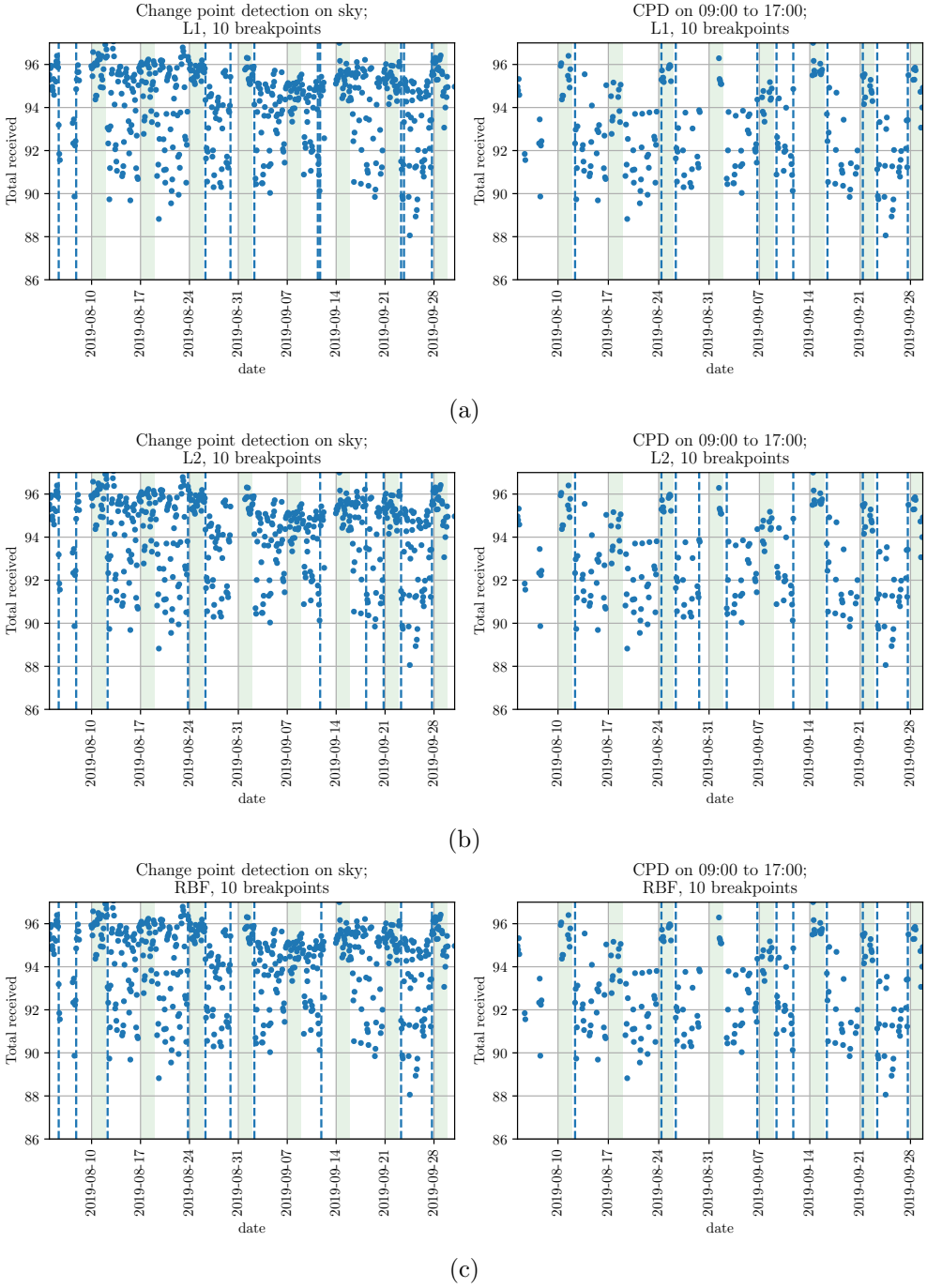
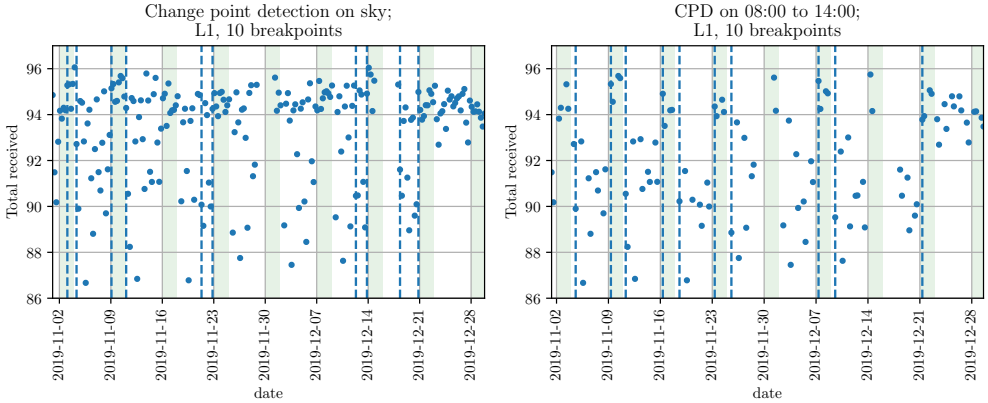
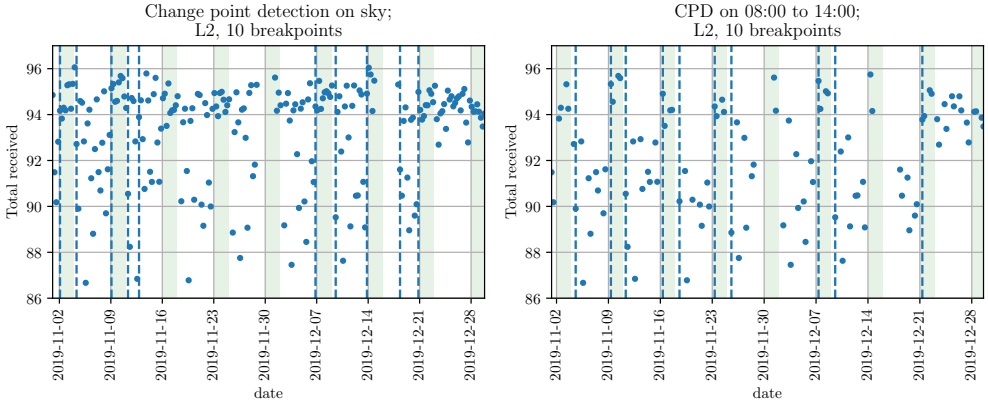


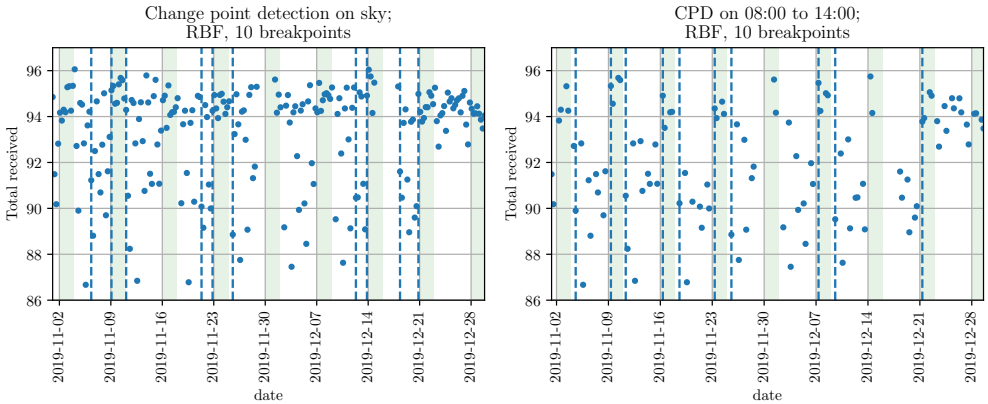
Figure 4.5: *Dynp* applied to the full data (left) and to the most busy hours of the day (right).



(a)



(b)



(c)

Figure 4.6: *Dynp* applied to the full data (left) and to the most busy hours of the day (right).

4.5 Daily Seasonality

Figure 4.2 shows that in the FlockLab data, the wireless link quality displays some degree of daily oscillations, as one may also incur from the autocorrelation plots in figure 4.1. Considering that the data was collected in regular intervals daily, also during midnight, this might be somewhat unsurprising. Indeed, if the hypothesis that the FlockLab wireless link quality depends strongly on interference from other devices is true, one would expect a better wireless link quality on average at midnight than during the day, at least on weekdays. This is because most people will be working during the day, and thus, there should be more interference then.

Inspired by this realization, we segment the data into different data sets, where each sample in each of the new data sets is taken at the same hour of day. Ideally, we would see that the wireless link quality outside of working hours is higher and varies less than during busy hours, as this would back up our hypothesis that the FlockLab testbed is disturbed by devices used during work. Furthermore, finding such evidence would allow us to decrease the size of the utilized data, since the samples collected outside working hours would be essentially constant and deliver no new information. As such, it would indeed be redundant.

Representative results are shown in figure 4.7. Indeed, the number of received packets varies much less at 8 p.m. and 2 a.m. than at 8 a.m. and 2 p.m. Still, it is evident from the Appendix D.2 that *ruptures* does not perfectly distinguish the weekends as in the other data sets for the November and December data. In the case of the August and September data in figure D.3, the results are better for the samples recorded at 9 a.m., 11 a.m., 1 p.m. and 3 p.m., but the search methods are somewhat unreliable for the other samples.

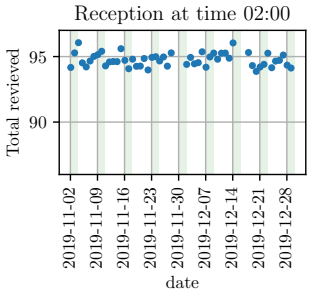
There could be multiple reasons for this. Firstly, if there are fewer samples per day, and the search method chooses one sample as a breakpoint, which in reality is close to the expected breakpoint, the effect of missing by a single sample is significantly bigger than if there are many data points to choose from. Furthermore, given few data points, there is little information from which to calculate the cost of intervals and declare shifts, as is also apparent from the results in section 4.3. For example, if we represent each day by a single data point, a search method has to decide from respectively two and five samples whether a weekend has a different cost than the weekday. This makes the approaches particularly prone to outliers and missing data, and motivates the use of more samples per day.

To remedy this problem we consider only the data where we expect the most seasonality, that is, the data recorded during the working hours from 8 a.m. to 17 p.m.[7] In this setting, the performance of *Pelt* does not improve much. Indeed, the robustness suffers somewhat, compared to when using the full data. This is perhaps due to the decreased number of points per day, resulting from considering only the data recorded during working hours. However, the *Dynp* algorithm without a minimum size restriction detects similar breakpoints on the full data, as on the data measured during active hours. This is shown on right hand side of figures 4.5 and 4.6. In fact, the performance generally even improves compared to when using the full data set. One could reason about this by considering that *Dynp* will find a specified number of breakpoints in order to minimize the total cost of the intervals between the breakpoints. Presumably, removing samples which are essentially the same for all the data will either not change the optimal segmentation that much, or it will increase the discrepancy between the intervals. The latter will be the case if the removed samples tend to bring the average values in the intervals closer together. In our case, it seems that this happens, and *Dynp* more strongly tends to set the breakpoints between the intervals given the larger discrepancy. Figures 4.5 and 4.6 show that change point detection on only the samples

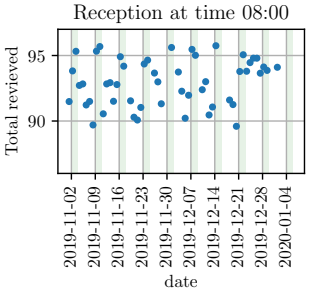
[7] The plots are labeled with different time stamps according to the time of wireless link quality tests.

recorded during working hours results in breakpoints that agree more strongly with our hypothesis than when we consider the full data. The samples recorded outside of working hours may disturb the change point detection, since they give an arguably irrelevantly good impression of the wireless link quality during weekdays, as these samples do not depend strongly on the day of the week.

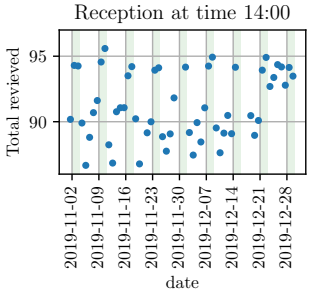
Personally, I believe it is not unlikely that such daily seasonality could be common in networks used by humans within only one timezone. Presumably, less people use wireless devices at night than during the day or evening, and thus, a seasonality similar to that found in the FlockLab data may often be present. Hence, for a range of applications, it could be worth checking the influence of data recorded during different hours of the day, as the behavior during some hours may carry more information than the behavior during other times of day. Furthermore, when detecting network shifts, these results indicate that it could be worth checking for changed probability distributions on samples recorded at a similar time of day, instead of checking the full day. The distribution of the number of received packets consistently displays daily shifts, as the wireless link quality is apparently better at night. So arguably, these shifts are not as interesting as those that change the wireless link quality under presumably similar conditions, i.e. during the same hours on different days.



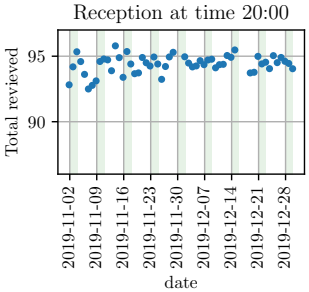
(a) At 2 a.m. the wireless link quality variation is small.



(b) At 8 a.m. the wireless link quality varies more.



(c) At 2 p.m. the wireless link quality varies more.



(d) At 8 p.m. the wireless link quality varies less.

Figure 4.7: Wireless link quality at different hours of the day

Chapter 5

CAIDA

The CAIDA data set contains data from internet traffic in Chicago recorded for one hour at a time approximately every one to three months between May 2013 and April 2016 [1]. In the CAIDA data set, there are two directions A and B, for each link, that both get measured in the same way. The data we use are computed from the original data, and has a sampling frequency of 100 samples per second. Thus, each hour contains approximately 360,000 samples. While for the FlockLab data we have some general idea of what type of results to expect, the CAIDA data set is a different beast, and is arguably also more difficult to interpret as it is higher dimensional. Furthermore, the data seems quite noisy, visible in figure 5.1, which motivates the use of low-pass filters. Nevertheless, we can apply the same methods as described up until now, and compare the results for different choices of parameters and included dimensions. We can also try to infer some information about the data from our results.

In the following, we perform a number of experiments to analyze the CAIDA data. Due to the size of each recording, however, we need to compress the data before applying the shift detection methods. Therefore, we create new data from the original or low-pass filtered data by taking the mean of 100 consecutive samples and representing this as a single data point. Thus, the signals on which we do change point detection contain approximately 3,600 samples instead of 360,000. After this, we bring the signal to zero mean and unit variance, i.e. normalize it using standard scaling, as described in chapter 3.

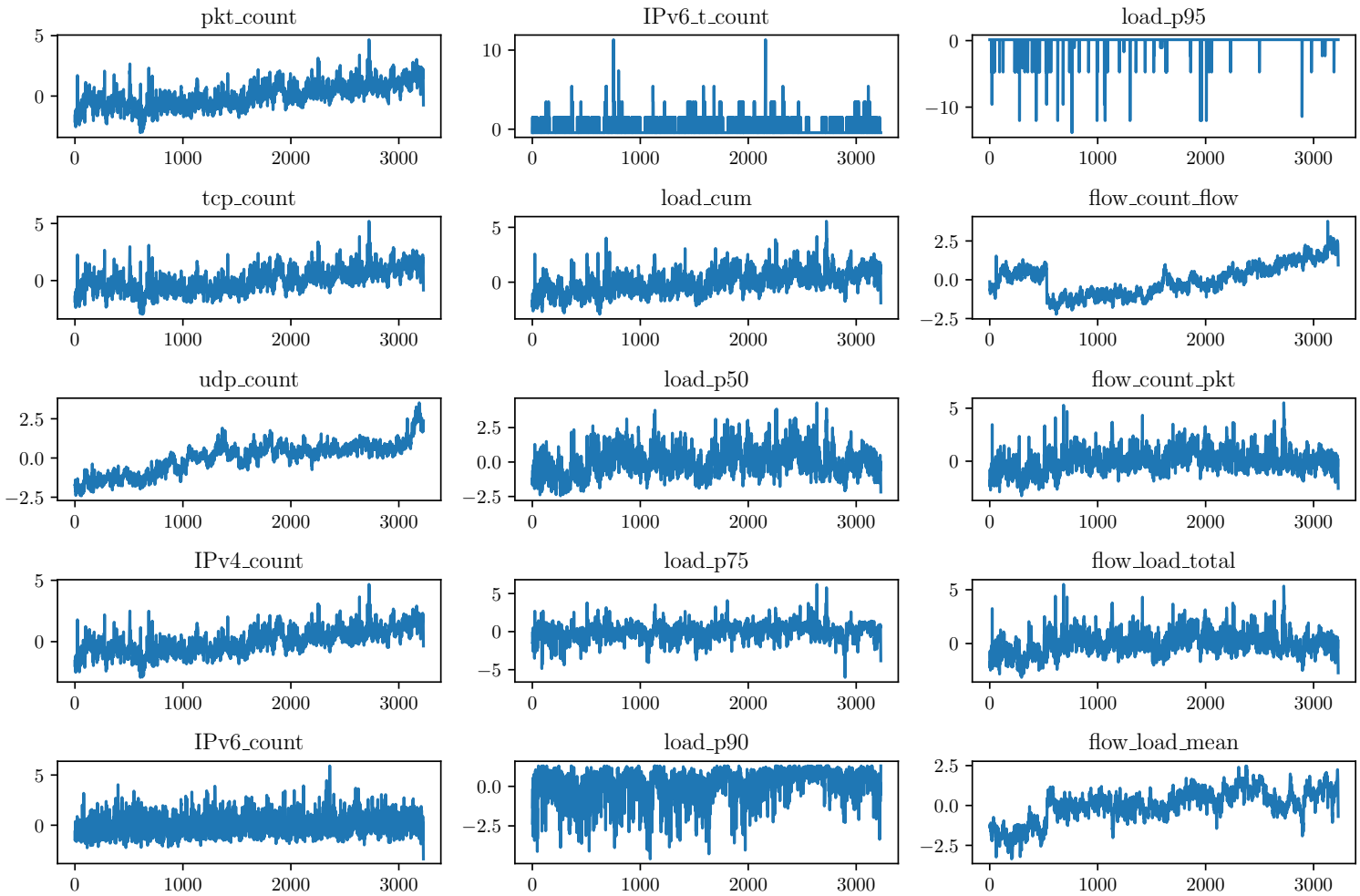


Figure 5.1: Example of a normalized CAIDA signal, taken from February 2nd, 2016 for the A direction. As with the signals use for change point detection, here, each point really represents the average of 100 consecutive samples.

5.1 Change Point Detection on Single Dimensions

For multiple dimensions, we do not know a priori whether they exhibit the same breakpoints or not. Change point detection on each individual dimension may actually result in the same breakpoints for every dimension. In this case, there would be little point in considering all the dimensions together, since the same breakpoints would be detected anyways. In order to get some idea of which dimensions agree on breakpoints, and which ones do not, we perform change point detection on each dimension and use the Hausdorff index, as described in section 3.3, to investigate their similarity. Due to the apparent noisy nature of the data, we also show the results using different low-pass filter window sizes. We find the breakpoints using *Dynp* with the L_2 cost, and show the results in figure 5.2.[1]

We find that the detected breakpoints depend on which dimension is used. Considering figure 5.1, it seems that the dimensions indeed look different and thus the detected breakpoints may then be different as well. We also see that applying a fairly small low-pass filter window of 100 samples, i.e. averaging over one second, does not influence the similarity matrix greatly. However, when increasing the window size to 10,000, i.e. averaging over 100 seconds, the breakpoints lie closer to each other, with the exception of those detected by the *load_p95*-dimension. Presumably, this is because we smooth out local variations and see only the long term trends when increasing the size of the low-pass window. As can be seen in figure 5.1, some of the dimensions, in particular *load_p95* and *IPv6_t_count*, are quite spikey compared to the other signals. This is possibly a reason why *Dynp* detects rather different breakpoints for these dimensions for a low-pass filter window size of 1 and 100 in figure 5.2.

[1] We use *Dynp* with the L_2 cost because we can then ensure the same number of breakpoints in each case, and search methods with the L_2 cost function produced the best results in chapter 4.

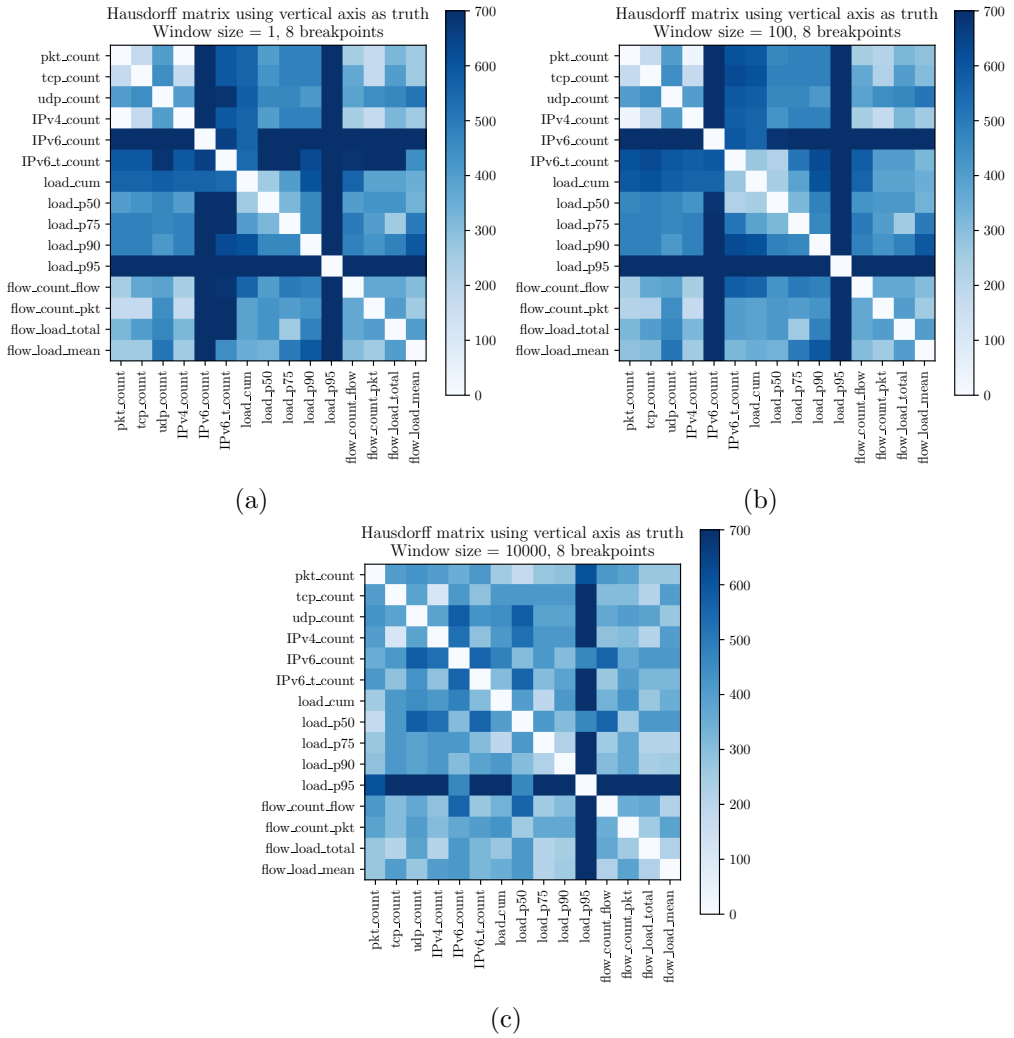


Figure 5.2: Hausdorff-index between breakpoints found by single dimensions for different low-pass window sizes using *Dynp* with the L_2 cost. A low Hausdorff-index indicates that the breakpoints are similar, since they all lie within a small margin of each other.

5.2 Correlating Dimensions of the CAIDA Data

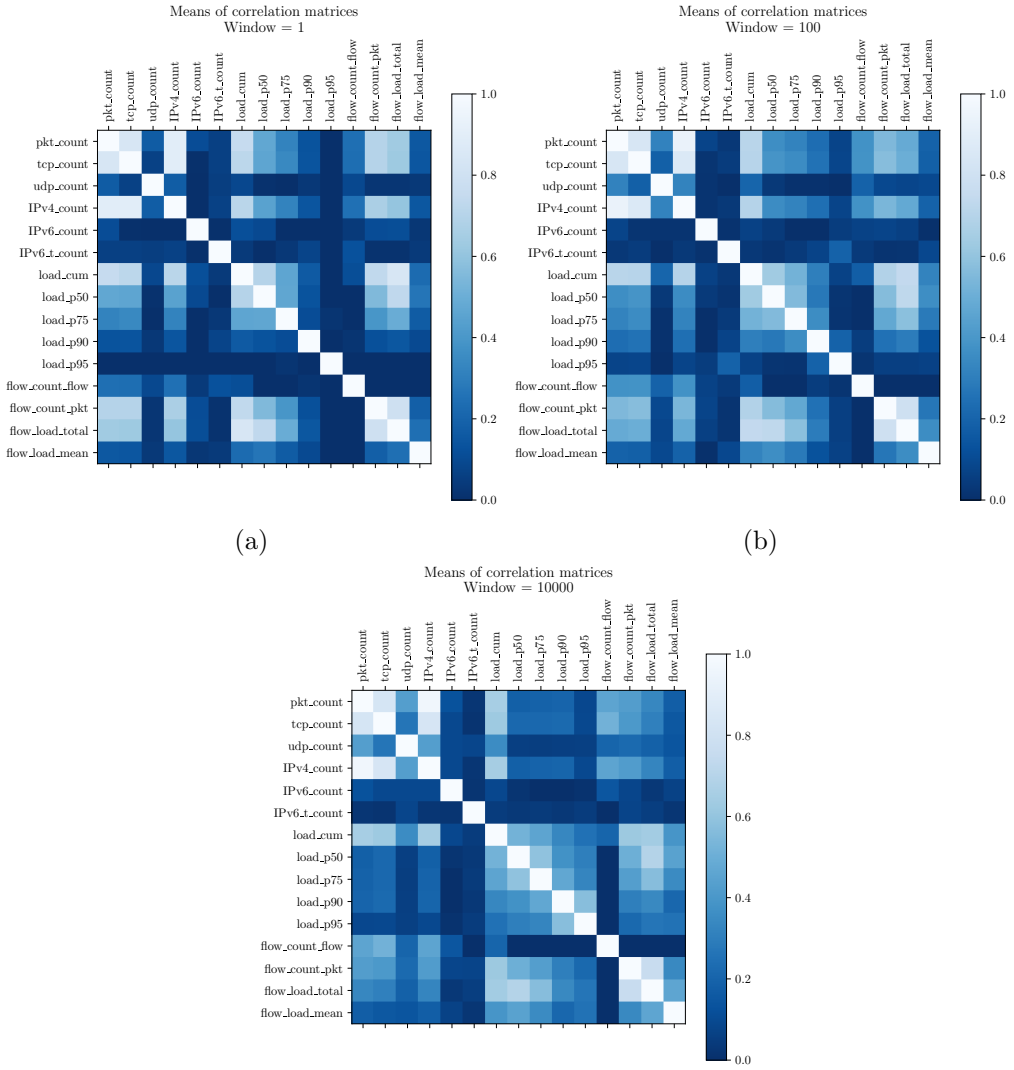
While we have established that some of the different dimensions of the CAIDA data set detect different breakpoints when analyzed separately, it could be that certain dimensions correlate with each other. Strong correlations could be harmful to change point detection, since some trends may be over-represented. In an extreme case, where two dimensions are identical, including both dimensions may be considered as giving nearly twice the weight to a single underlying trend *causing* both dimensions to look the way they do.[2] Thus, if we find that certain dimensions of the data correlate strongly, it may be justified to consider only one of the strongly correlating dimensions for change point detection.

We choose to measure the correlation between the dimensions using the Kendall correlation. This is because it is non-parametric, and we do not have much information about the possible underlying trends. Correlating the different dimensions of the CAIDA data set reveals significant correlation between certain dimensions, as is shown in figure 5.3. Interestingly, the correlation matrices are reminiscent of the similarity plots in figure 5.2. Indeed, we see that the dimensions that correlate very strongly also have a low Hausdorff-Index, indicating that the breakpoints are the same up to a small margin. This further justifies removing single dimensions, since the breakpoints found by the heavily correlated dimensions would be weighted particularly strongly. In particular, the sets of dimensions {*pkt_count*, *IPv4_count*, *tcp_count*} and {*flow_count_pkt*, *flow_load_total*} correlate strongly. Therefore, we perform change point detection twice; once using all the dimensions of the data set, and once representing the former of the aforementioned sets by the *pkt_count*, and the latter by the *flow_count_pkt*.[3] We also apply different low-pass filters for each combination of dimensions.

One might think that applying the low-pass filter should make the breakpoints more similar, as only the long term trends stay when applying a large low-pass window, possibly making the dimensions themselves more similar. Furthermore, we saw in figure 5.2c that the greatest number of samples between most breakpoints decreases when the window size is 10,000, and figure 5.3 even shows that dimensions tend to correlate more strongly when we apply a low-pass filter, than when we do not. Figure 5.4 shows the effect of considering all dimensions contrary to subsets of them, as well as the effect of applying a low-pass filter, and performing change point detection using *Dynp* for seven and eight breakpoints. We see that the breakpoints are indeed different depending on what dimensions we use, indicating that some trends might be over-represented when dimensions are very similar. However, it is not obvious from the figures *how* the breakpoints change. Both for few and for many breakpoints, there seems to be some disagreement, but this disagreement also depends on the size of the used low-pass window. Interestingly, it is not always the case that increasing the size of the low-pass window makes the breakpoints more similar. We are not sure why this happens, considering the discussion above, but do report the results.

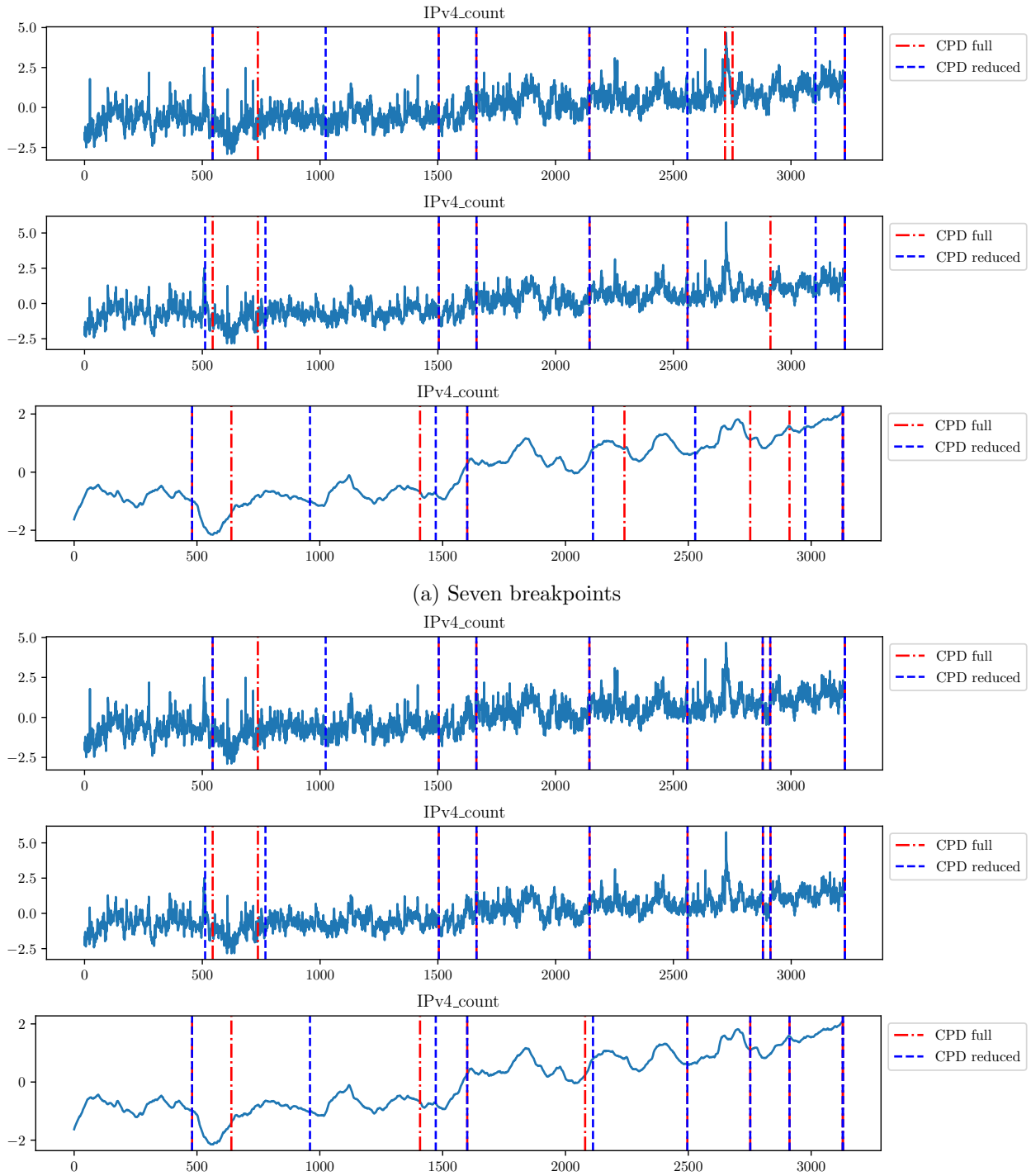
[2] In the case of the L_1 and L_2 costs, it would indeed be twice the weight, as the norms of the deviations from the median and empirical mean add. However, for the *RBF* cost, the analysis is more complicated, since the norm of the signal occurs in an exponential.

[3] This choice is arbitrary, but considering the strong correlation, the detected breakpoints will likely not differ from the results if we had chosen other dimensions to represent each set.



(a) (b) (c)

Figure 5.3: Means of Kendall correlation coefficient computed for the different dimensions of low-pass filtered CAIDA data. While most dimensions correlate at least somewhat, we see that there is a particularly strong correlation among the in the sets of dimensions {*pkt_count*, *IPv4_count*, *tcp_count*} and {*flow_count_pkt*, *flow_load_total*}. Also interesting to note is the lack of correlation between *IPv6_count* and *IPv6_t_count* and most other dimensions.



(a) Seven breakpoints

(b) Eight breakpoints

Figure 5.4: Change point detection using all available dimensions (CPD full) and representing the strongly correlated dimensions through *pkt_count* and *flow_count_pkt*, as described in section 5.2 (CPD reduced). The two subplots show breakpoints detected for a low-pass filter window of 1 (top), 100 (middle) and 10,000 (bottom). We show the detected breakpoints on top of the *pkt_count* since it is available in both scenarios, and represents the largest number of dimensions in CPD reduced.

5.3 Long Term Evolution

The CAIDA data set was collected over nearly three years; from May 2013 to April 2016. One could suspect that the most significant distribution shifts happen over a longer time span than one hour, which is the size of the measurements available to us. An approach to investigating this would be to consider the entire data set as a single signal, as shown in figure 5.6, and then perform change point detection on this signal. Note that in figure 5.6,while the horizontal axis does represent the time within one hour of recorded data, is not representative of the actual time between the data sets. Between every shaded region, there are in fact multiple weeks, if not months, that are not shown.

The resulting signal is reminiscent of a noisy piecewise constant signal, at least for some of the dimensions. Other dimensions, such as the *IPv6_t_count*, *load_p90* and *load_p95* show rather spiky behavior. Presumably, the reason for the piecewise constant reminiscence is that the dimensions would normally change smoothly, but at the boarder between days, any discrepancy between the days seems like an abrupt change, similar to those we see piecewise constant signals. Furthermore, it could be that the hours we have recorded do not perfectly represent the internet state at the time the data were collected. For example, it could be that we, by chance, recorded the data during an hour when particularly many people were using the internet, leading to extreme measurments.

With this in mind, we could expect that change point detection on the signal containing all the available days will separate the days more or less perfectly. Figure 5.5 shows the result when applying *Dynp* to detect 19 breakpoints; one less than the number of days in the relevant data.[4][5] We see that indeed, the match is perfect. However, considering the approximately piecewise constant nature of the concatenated signal, this is not particularly surprising. The results in chapter 3 show that *Dynp* performs best on piecewise constant signals. An interesting question is whether these shifts arise from a long term network shift, or if they are only due to the nature of sampling a single hour of data from a time span of many months. It is not possible to answer this question with the CAIDA data alone, as one would need more regular data to establish whether the typical behavior of the relevant internet traffic has changed.

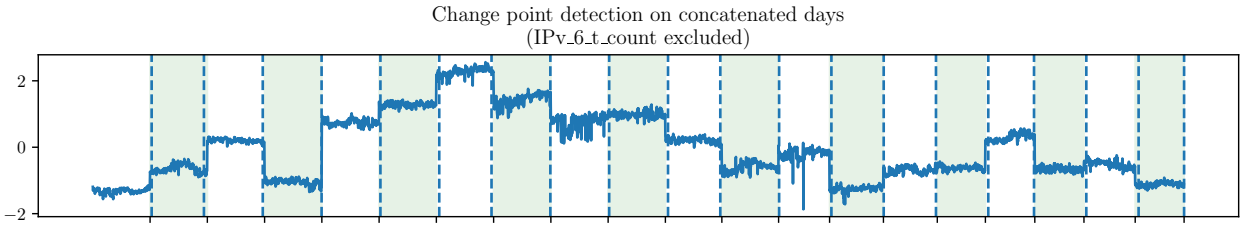


Figure 5.5: Change point detection on concatenated days. The breakpoints are displayed on top of the *flow_load_mean*, which is an arbitrary choice. The different shaded regions indicate different days, and we see that the transitions coincide perfectly with the 19 found breakpoints. It is worth noting that the detected breakpoints were not found only using the shown dimension, but rather using all dimensions (except the *IPv6_t_count*). If some breakpoints seem off in this figure, they are likely detected due to other dimensions.

[4] *ruptures* also always places a breakpoint at the end of the signal, which is not included in the number of breakpoints we specify.

[5] We do not include the *IPv6_t_count* since it is spikey. When we included it, the breakpoints did not agree perfectly with the transitions between days. However, considering the shape of this dimension in figure 5.6, we think it is justified to neglect it, as its behavior seems rather chaotic.

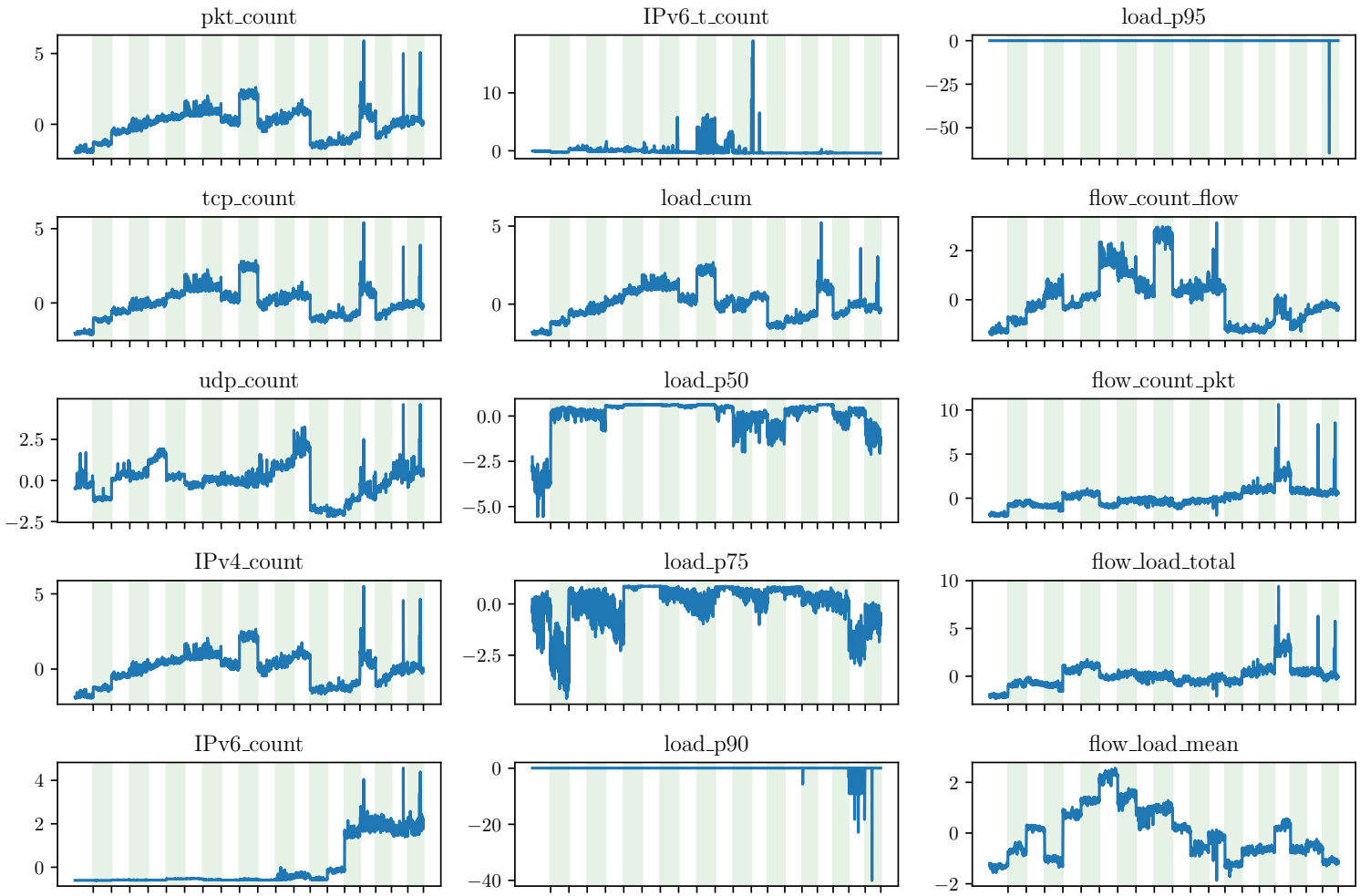


Figure 5.6: CAIDA data in the B direction, concatenated according to when the data was collected. The shaded regions and x-axis ticks indicate the transitions between days.

Chapter 6

Discussion

6.1 Shift Detection Methods

While in our analysis we focused on two algorithms, *Dynp* and *Pelt*, used with only three different cost functions, the L_1, L_2 and RBF costs, many other shift detection methods are available. In particular, in *ruptures*, one can specify custom cost functions to try to capture other phenomena than deviations from the median and mean, or changes in a mean embedding. We saw, for example, that while the algorithms we used generally perform well on piecewise constant signals, they do not on more complex time series, such as the piecewise wavy and linear signals. This might be different for other cost functions.

When removing the hours when we do not expect people to work from the analyzed data in section 4.5, the results obtained for the FlockLab data were particularly good. Considering the right hand side plots in figure 4.5, this might be unsurprising, as the resulting signals seem to resemble noisy piecewise constant signals.

Conversely, considering figure 5.1, which shows an example of a CAIDA signal, one might suspect that the breakpoints found within one day of this data set may not be representative of actual shifts, since the signal is not piecewise constant. It could be interesting for future research to apply different cost functions to the CAIDA data and compare the resulting breakpoints to those obtained using the methods in this report. If the breakpoints differ greatly, it may be an indication that either the cost functions we use or the new cost functions are not well suited to detect network shifts on the CAIDA data.

6.2 FlockLab

6.2.1 The Risk of Confirmation Bias

In the FlockLab analysis, we see that we often have to either set some parameters properly or consider only parts of the data in order to obtain results that agree well with our hypothesis. Examples of the first point are the *pen*, *n_bkps*, *min_size* and *jump* parameters. Choosing these inappropriately sometimes leads to results which do not agree too well with the weekend-seasonality hypothesis. Further, if we only consider the hours during which we *expect* people to work, one could view the obtained results (which match the weekend-hypothesis perfectly) as obtained through a sort of confirmation bias, as we suspect the lower link-quality could be due to disturbances from other online applications, and change parameters if the fit is poor.

On the other hand, it can be natural to place restrictions on the detected breakpoints, e.g. by tuning the *min_size* parameter, if we are not interested in e.g. very densly packed breakpoints. Further, neglecting data that delivers little information is a well established approach in data analysis [6].[1] As we did indeed see that the wireless link quality mostly varies throughout working hours, and is more or less constant outside of working hours, disregarding the data that carry little information is, in my view, justified.

6.2.2 Analysis of Seasonality by Other Methods

There already exist tools for detecting regular seasonality and trends in time series. An example is the STL-analysis, which is implemented in the python library `statsmodels`.[2] Hoping to be able to detect a weekly seasonality, we experimented with passing the full, scaled and unscaled, FlockLab Sky data, with and without imputation, to the function `STL` from `statsmodels.tsa`, using different parameters. However, the results were difficult to interpret, and in the resulting model, the residuals were on the same scale as the detected trends, indicating that the model may not be too insightful in our case. A possible reason for this is that we considered the full data, which also contains the samples taken outisde of working hours. In section 4.5, we find that change point detection works best on data where we consider only the samples recorded during working hours, and suspect that this might be true for the STL-analysis as well, since the trends seem to be clearer on the data that only considers these hours. Future research could try passing such reduced data to this function, either using the FlockLab Sky data or other network data where certain hours of the day display less variance than others. Then, we may hope to find that also this analysis method works better when considering the time at which we record the measurements.

6.3 CAIDA

6.3.1 Inter- and Intra-day Similarity

The CAIDA data consists of measurements spanning one hour of internet traffic per recording between 2013 and 2016. Presumably, the most significant network shifts taking place in this data set occur over a larger time span than the single hours on which we have data. Thus, when analyzing the CAIDA data, we would like to compare how similar points in a given interval are, compared to how similar points in the entire data set are. To this end, we define the *per-point-cost*.

The Per-Point-Cost

Formally, given a signal with time indices in I_{tot} and two consecutive breakpoints $b_1, b_2 \in I_{tot}$, we consider the interval

$$I = [b_1, b_2) \subset I_{tot}$$

and compute

$$c_I = \frac{C(I)}{|I|}, c_{tot} = \frac{C(I_{tot})}{|I_{tot}|}$$

[1] E.g. principal component analysis is *designed* to discard (linear combinations of) the dimensions of a data set containing the least variance.

[2] `https://www.statsmodels.org/stable/examples/notebooks/generated/stl_decomposition.html`

for some cost function C. c_I and c_{tot} measure how much each point in the relevant interval contributes to the cost on average. We then define the *per-point-cost* as

$$\tilde{c_I} = \frac{c_I}{c_{tot}}.$$

Intuitively, the per-point-cost measures how much each point in a detected interval, i.e. data between two consecutive breakpoints, costs on average, measured by the relevant cost function, compared to the average cost of each point in the data. If the per-point-cost is small, this indicates that the points in a detected interval are more similar than the points in the entire data set are on average, measuring similarity with the relevant cost function.

Applying the Per-Point-Cost

In order to compare the intra-day similarity to the inter-day-similarity, one approach could be to apply the per-point-cost with each interval being a separate day. Should the cost be low in this setting, it would indicate that the differences between days are more significant than those within a single day. However, considering the nature of the CAIDA data, this comparison has its limitations. Normally, there are months between the recorded data, which itself only spans one hour; a much shorter time than the gaps between data. As a consequence, we do not know whether the recorded hour is actually representative of the "average" internet behavior for a given month, and further, there is no guarantee on how well we can compare the hours themselves, as some of them may be anomalous. Additionally, while the discrepancy between days may be large, resulting in a large c_{tot}, it could be that taking information from only one hour of data results in a c_I which is too small to represent an actual day, due to relatively low variance.

This becomes particularly visible when concatenating the available days and plotting the result; see figure 5.6. The result is reminiscent of a piecewise constant signal; at the boarder between days, the different dimensions normally show considerable jumps, and then remain nearly constant until the next available measurement. This happens because normally, the change in each dimension is smooth, but any minor difference between two consecutive recordings will seem abrupt.

Possibly, if there were more data available, there could be greater intra-day variation, making it feasible to compare entire days to each other. We may still see large jumps at the boarder between days, but maybe the data representing each day is less likely to be anomalous. We would encourage further research to investigate long-term network shifts by considering precisely data over longer time spans.

Chapter 7

Conclusion

In this report, we have explored different shift detection methods in the context of network data, and applied these methods to the FlockLab and CAIDA data sets.

For the FlockLab data, we had some idea of what types of shifts to expect, and were able to confirm these suspicions using the *Pelt* and *Dynp* algorithms with the L_1, L_2 and RBF cost functions. Realizing that internet traffic may be highly seasonal with respect to the time of day, we were able to reduce the size of the utilized data set, as the data recorded outside of working hours generally displayed little variation. Indeed, the remaining data resembled a noisy piecewise constant signal, and applying the shift detection methods to this signal lead to results that agreed more strongly with our hypothesis. Together with the results in chapter 3, it seems that *Pelt* and *Dynp* are able to detect shifts in noisy piecewise constant signals rather well, when using the L_1, L_2 and RBF cost functions. Thus, we propose two possible approaches for future research. On the one hand, finding meaningful transformations of time series data to piecewise constant signals would allow us to use the shift detection methods that we apply in this paper. As such, it could be worth seeking such transformations for other signals. On the other hand, this might be quite challenging for arbitrary signals. Therefore, another approach we would suggest is to find other cost functions that are able to detect shifts on non-piecewise constant signals. In [8], the authors already mention multiple possible cost functions that future research could investigate.

We also saw for the FlockLab data that there are regularly daily shifts, in the sense that the wireless link quality is better at night than during the day. In our view, it could be sensible to consider data recorded under seemingly similar conditions when detecting network shifts. One may expect that the conditions during the day are different from those during the night, making it reasonable to consider data recorded during the same time of day when searching for network shifts. The time of day is, however, possibly not the only factor which comes into play when concerning similar conditions, and we would suggest future research in investigating which conditions allow for a fair comparison, in order to detect network shifts. In particular, it could be interesting to see how strong of an influence the time of recording has on other data, and whether is is only characteristic to the FlockLab data.

The CAIDA data is more sparsely spaced and is higher dimensional than the FlockLab data. It turns out that some of its dimensions correlate strongly, and when performing change point detection on each dimension individually, the dimensions that correlate strongly tend to find similar breakpoints. From this, we conclude that it may be wise to remove some of the high correlation in network data sets prior to shift detection, since some trends may otherwise be over-represented. Furthermore, while we were able to perform change point detection on the CAIDA data, one could question the quality of the estimates, as the data does not seem to be piecewise constant, and the

methods we used displayed poorer results for more complex signal shapes in chapter 3. We would suggest further research in either finding methods that can detect shifts on arbitrary signal shapes, or to transform the CAIDA data in a meaningful way to something resembling a piecewise constant signal before making inferences on possible causes of these shifts.

Due to the sparsity of the CAIDA data, we would also encourage research on data taken over longer time spans, or more regularly. This could allow for long-term shift detection, which is not possible on the CAIDA data, since we have no guarantee of how representative the recorded hours are of the internet behavior of the time period in which the data were collected.

Bibliography

[1] Center for Applied Internet Data Analysis based at the University of California's San Diego Supercomputer Center. The caida ucsd statistical information for the caida anonymized internet traces, 2019. Trace Statistics for CAIDA Passive OC48 and OC192 Traces, https://www.caida.org/catalog/datasets/trace_stats/#H2480.

[2] Romain Jacob, Reto Da Forno, Roman Trüb, Andreas Biri, and Lothar Thiele. Wireless Link Quality Estimation on FlockLab – and Beyond. In *Proceedings of the 2nd International Workshop on Data Acquisition to Analysis (DATA)*, New York, NY, USA, November 2019. ACM. doi:10.3929/ethz-b-000355846.

[3] Romain Jacob, Reto Da Forno, Roman Trüb, Andreas Biri, and Lothar Thiele. Dataset: Wireless Link Quality Estimation on FlockLab – and Beyond, March 2020. doi:10.5281/zenodo.3731498.

[4] Zachary C. Lipton, Yu-Xiang Wang, and Alex Smola. Detecting and correcting for label shift with black box predictors, 2018. arXiv:1802.03916.

[5] Stephan Rabanser, Stephan Günnemann, and Zachary C. Lipton. Failing loudly: An empirical study of methods for detecting dataset shift. *Proceedings of the 2nd Workshop on Data Acquisition To Analysis, DATA'19*, pages 57 – 60, 2019. doi:https://doi.org/10.5281/zenodo.3408382.

[6] Dr S Velliangiri, S Alagumuthukrishnan, and Iwin Thanakumar Joseph Swamidason. A review of dimensionality reduction techniques for efficient computation. 01 2020. doi:10.1016/j.procs.2020.01.079.

[7] Ingo Steinwart, Philipp Thomann, and Nico Schmid. Learning with hierarchical gaussian kernels, 2016. arXiv:1612.00824.

[8] Charles Truong, Laurent Oudre, and Nicolas Vayatis. Selective review of offline change point detection methods. *Signal Processing*, 167:107299, 2020. URL: https://www.sciencedirect.com/science/article/pii/S0165168419303494, doi:https://doi.org/10.1016/j.sigpro.2019.107299.

Appendix A

Definitions

Definition A.0.1 (Empirical mean)**.** Given a signal $\{X_i\}_{i\in\{1,\cdots,T\}} \subset \mathbb{R}^n$, we define the *empirical mean* of the signal as

$$\hat{\mu}_X := \frac{1}{T}\sum_{i=1}^{T} X_i$$

Definition A.0.2 (Median)**.** Given a signal $\{X_i\}_{i\in\{1,\cdots,T\}} \subset \mathbb{R}^n$, we define the *median* of the signal as $\bar{X}$, such that $|\{X_i : \bar{X} \geq X_i\}| = |\{X_i : \bar{X} \leq X_i\}|$, and $\bar{X} \in \{X_i\}$. If this equation cannot be satisfied, $\bar{X}$ is the mean of the two numbers that most closely satisfy the equation.

Definition A.0.3 (L_1 *cost*)**.** Given a signal $\{X_i\}_{i\in\{1,\cdots,T\}} \subset \mathbb{R}^n$, we define the L_1 *cost* as

$$C_{L_1}(X) := \sum_{t=1}^{T} \left\| X_t - \bar{X} \right\|_1$$

Where $\bar{X}$ is the median of the signal.

Definition A.0.4 (L_2 *cost*)**.** Given a signal $\{X_i\}_{i\in\{1,\cdots,T\}}$, we define the L_2 *cost* as

$$C_{L_2}(X) := \sum_{t=1}^{T} \|X_t - \hat{\mu}\|_2^2$$

Where $\hat{\mu}$ is the empirical mean of the signal.

Definition A.0.5 (*RBF cost*)**.** Given a signal $\{X_i\}_{i\in\{1,\cdots,T\}} \subset \mathbb{R}^n$, we define the *RBF cost* as

$$C_{RBF}(X) := \sum_{i=1}^{T} \|\Phi(X_i) - \bar{\mu}\|_{\mathcal{H}}^2$$

Where $\Phi(x) = k(x,\cdot)$, $k(x,y) = \exp(-\gamma \|x-y\|_2^2)$ is the *RBF* kernel, and $\bar{\mu}$ is the empirical mean of the embedded signal $\{\Phi(X_i)\}_{i\in\{1,\ldots,T\}}$. Further, γ is a bandwidth factor, which can be chosen heuristically as the inverse of the median of all pairwise distances in the signal.

Appendix B

Analytical Results

B.1 Additive Shift Cost Function Invariance

L_1 and L_2 costs

Lemma B.1.1. *Given two signals $\{X_t\}_{t\in\{1,\dots,T\}}$ and $\{Y_t\}_{t\in\{1,\dots,T\}}$, such that $Y_t := X_t + a$ for some $a \in \mathbb{R}^n$, then for the cost functions C_{L_1}, C_{L_2} and C_{RBF}, X and Y have the same cost.*

Proof. Let $\hat{\mu}_X$ and $\bar{X}$ be the empirical mean and median of the signal X, respectively, and $\hat{\mu}_Y$ and $\bar{Y}$ be the empirical mean and median of the signal Y, respectively. Clearly, $\hat{\mu}_Y = \hat{\mu}_X + a$, $\bar{Y} = \bar{X} + a$, and it follows immediately that

$$C_{L_1}(X) = C_{L_1}(Y) \tag{B.1}$$

and

$$C_{L_2}(X) = C_{L_2}(Y) \tag{B.2}$$

The RBF-cost function is defined as

$$C_{RBF}(X) := \sum_{i=1}^{T} \|\Phi(X_i) - \bar{\mu}\|_{\mathcal{H}}^2 \tag{B.3}$$

Expanding the norm, we obtain (with $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ as the Hilbert-space inner product)

$$\|\Phi(X_i) - \bar{\mu}\|_{\mathcal{H}}^2 = \langle k(X_i,\cdot) - \bar{\mu}, k(X_i,\cdot) - \bar{\mu}\rangle_{\mathcal{H}} = k(X_i,X_i) + \|\bar{\mu}\|_{\mathcal{H}}^2 - 2\langle\Phi(X_i),\bar{\mu}\rangle. \tag{B.4}$$

Writing out $\bar{\mu}$

$$\bar{\mu} = \frac{1}{T}\sum_{i=1}^{T}\Phi(X_i) = \frac{1}{T}\sum_{i=1}^{T} k(X_i,\cdot) \tag{B.5}$$

which leads to

$$\|\mu\|_{\mathcal{H}}^2 = \frac{1}{T^2}\sum_{i=1}^{T}\sum_{j=1}^{T} k(X_i,X_j) \tag{B.6}$$

$$\langle\Phi(X_i),\bar{\mu}\rangle = \frac{1}{T}\sum_{j=1}^{T} k(X_i,X_j). \tag{B.7}$$

Thus, $C_{RBF}(X)$ depends only on expressions of the form $k(X_i, X_j)$. We now see that exchanging X and Y must result in the same cost, since

$$k(X_i, X_j) = \exp(-\gamma \|X_i - X_j\|_2^2) = \exp(-\gamma \|X_i + a - (X_j + a)\|_2^2) = k(Y_i, Y_j) \tag{B.8}$$

□

B.2 Scaled Signals

Lemma B.2.1. *Given the signals $\{X_t\}_{t\in\{1,\ldots,T\}}$ and $\{Y_t\}_{t\in\{1,\ldots,T\}}$ and a penalty p, such that $Y_t := aX_t$ for some $a \neq 0$, if the set of breakpoints $B = \{b_1, \ldots, b_k\}$ minimizes $C_{L_1}(X) + pk$, then B also minimizes $C_{L_1}(Y) + |a|pk$.*

Proof. We see from the definition of C_{L_1} that

$$C_{L_1}(Y) = |a| C_{L_1}(X) \tag{B.9}$$

It follows that

$$\min_{B=\{b_1,\ldots,b_k\}} C_{L_1}(Y) + |a|pk = |a| \min_{B=\{b_1,\ldots,b_k\}} C_{L_1}(X) + pk \tag{B.10}$$

Since $|a| > 0$, a set B which minimizes $C_{L_1}(X) + pk$ also minimizes $C_{L_1}(Y) + |a|pk$.

□

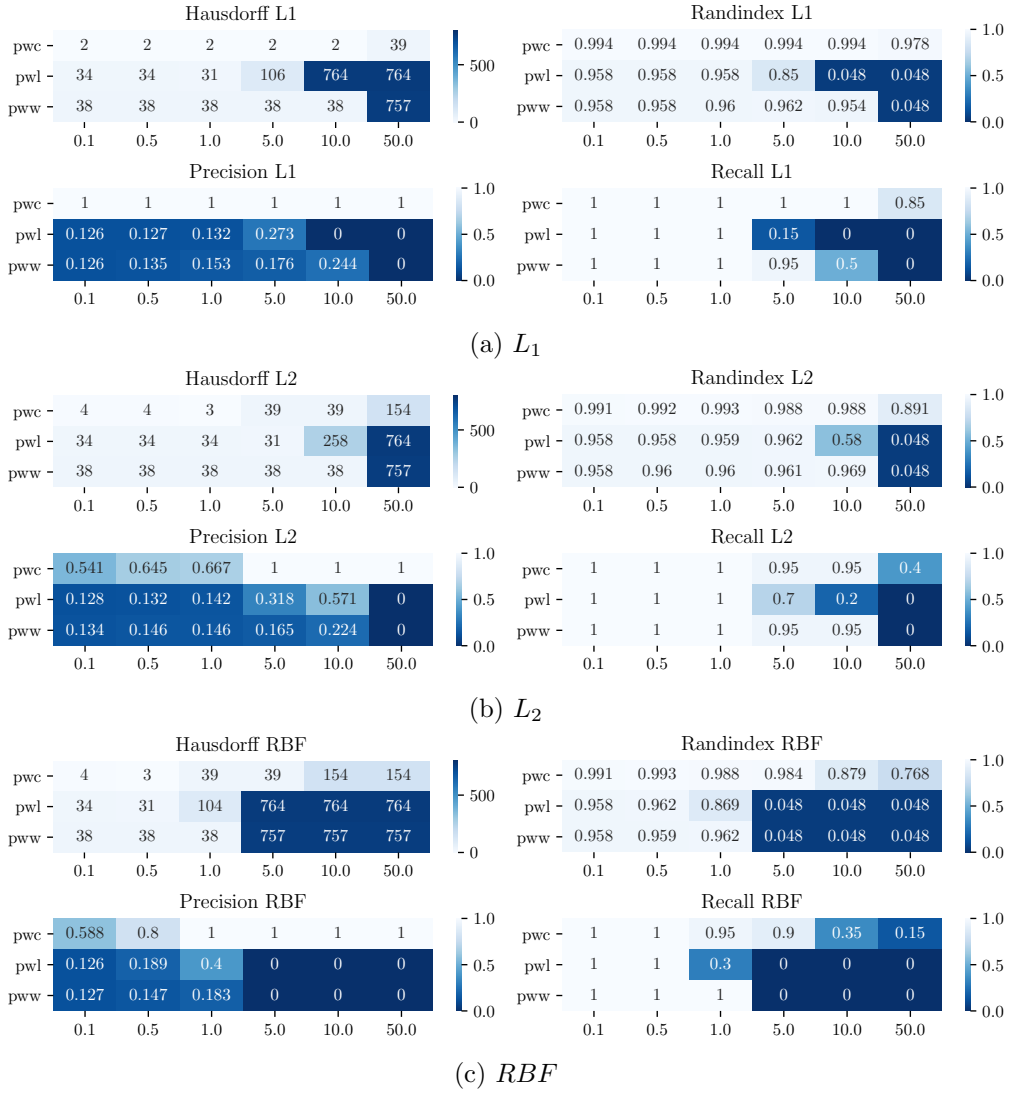
Lemma B.2.2. *Given the signals $\{X_t\}_{t\in\{1,\ldots,T\}}$ and $\{Y_t\}_{t\in\{1,\ldots,T\}}$ and a penalty p, such that $Y_t := aX_t$ for some $a \neq 0$, if the set of breakpoints $B = \{b_1, \ldots, b_k\}$ minimizes $C_{L_2}(X) + pk$, then B also minimizes $C_{L_2}(Y) + a^2pk$.*

Proof. Left as an exercise to the reader. □

Appendix C

Full Results Shift Detection Methods

C.1 Full Results for *Pelt*

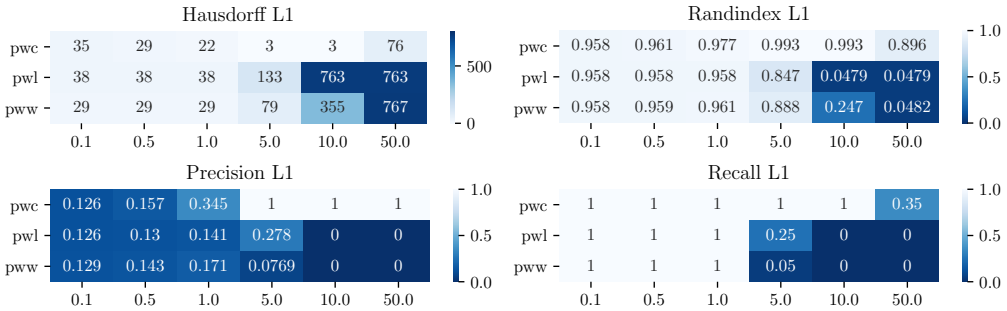


(a) L_1

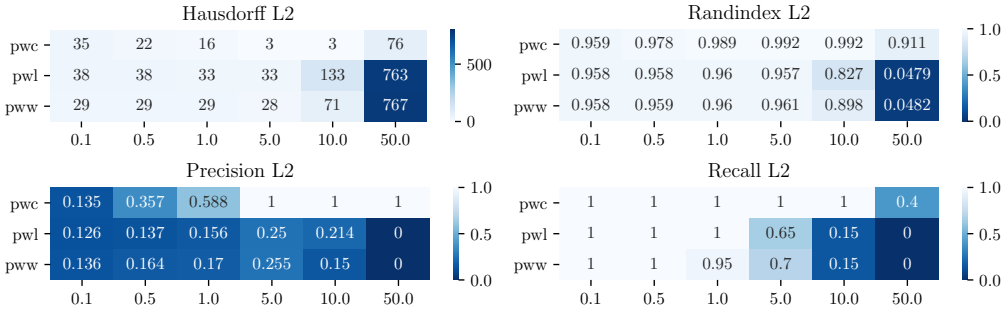
(b) L_2

(c) *RBF*

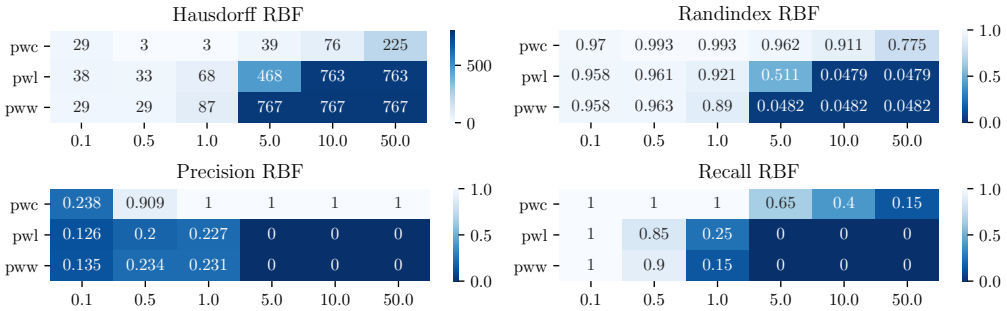
Figure C.1: Results for noiseless signals using *Pelt* with different cost functions.



(a) L_1

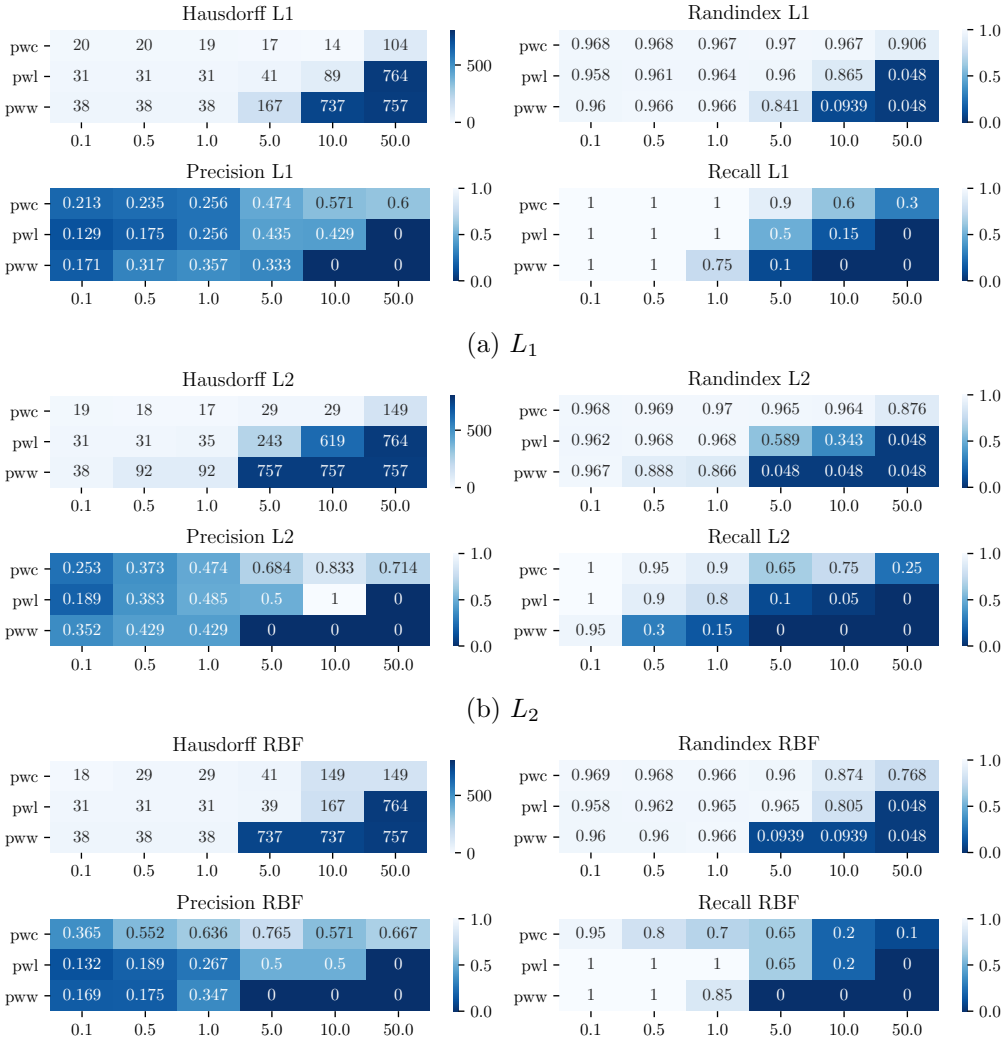


(b) L_2



(c) *RBF*

Figure C.2: Results for the same signals but with additive white Gaussian noise with a standard deviation of 3. The captions indicate the utilized cost function.

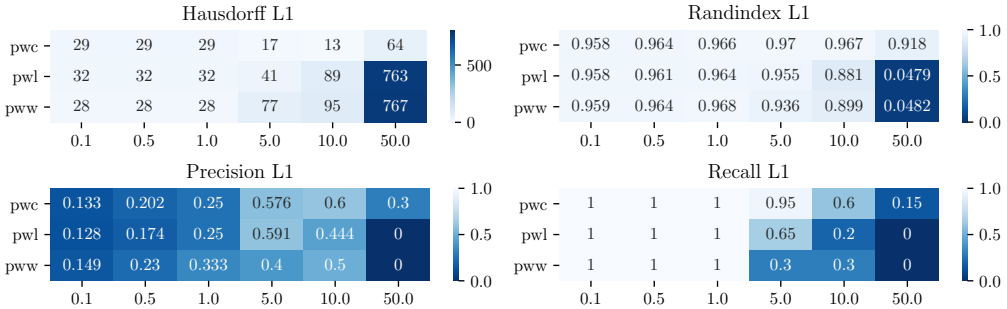


(a) L_1

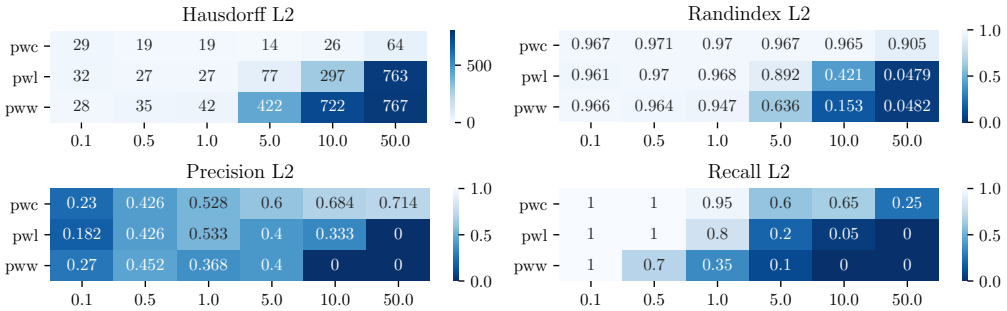
(b) L_2

(c) *RBF*

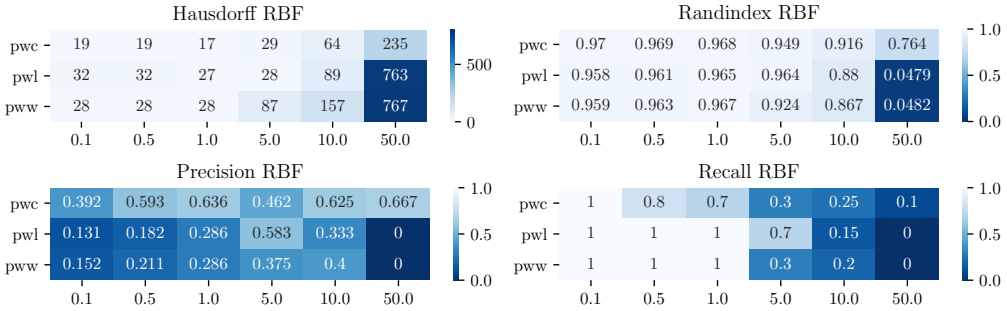
Figure C.3: Results for low-pass filtered noiseless signals for different cost functions.



(a) L_1



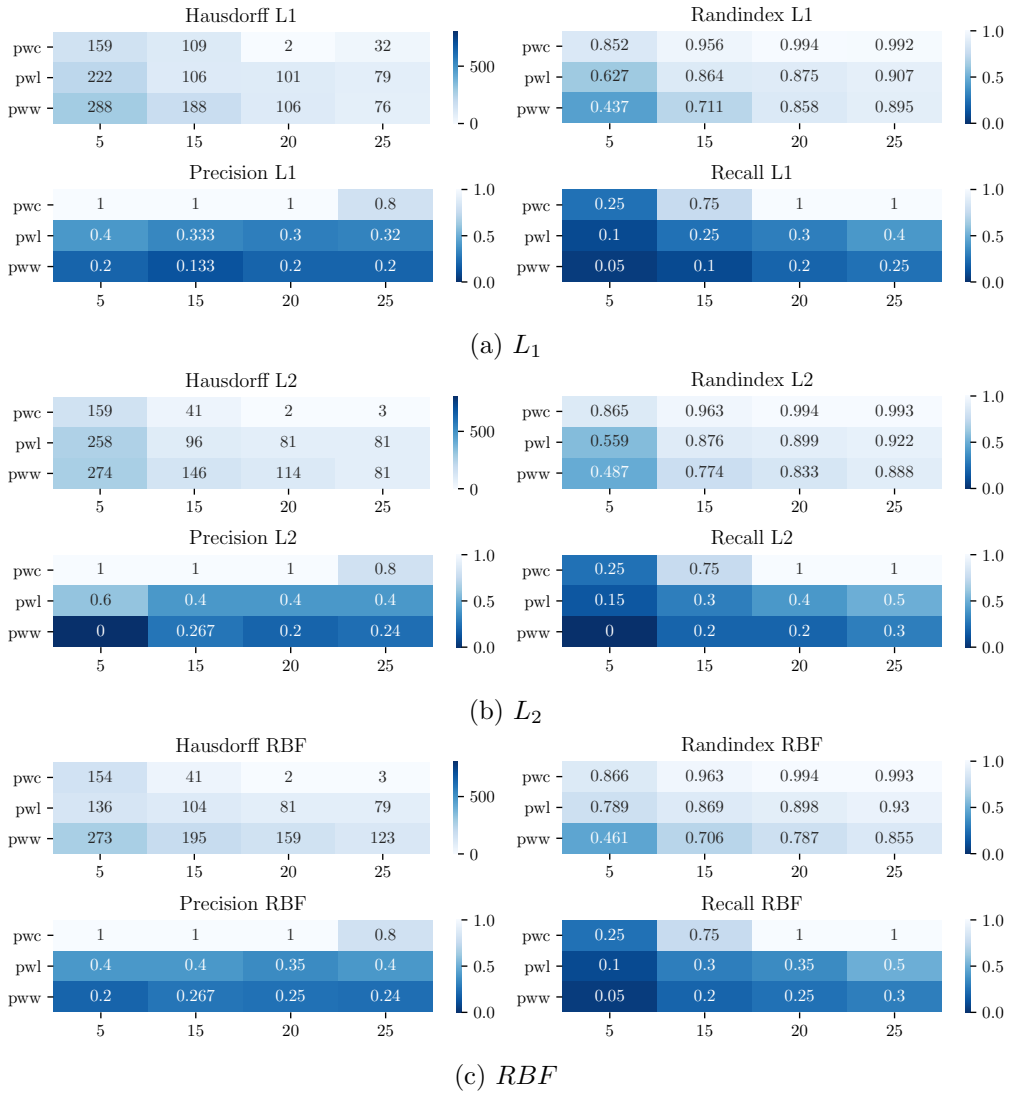
(b) L_2



(c) *RBF*

Figure C.4: Results for low-pass filtered noisy signals using *Pelt*.

C.2 Full Results for *Dynp*

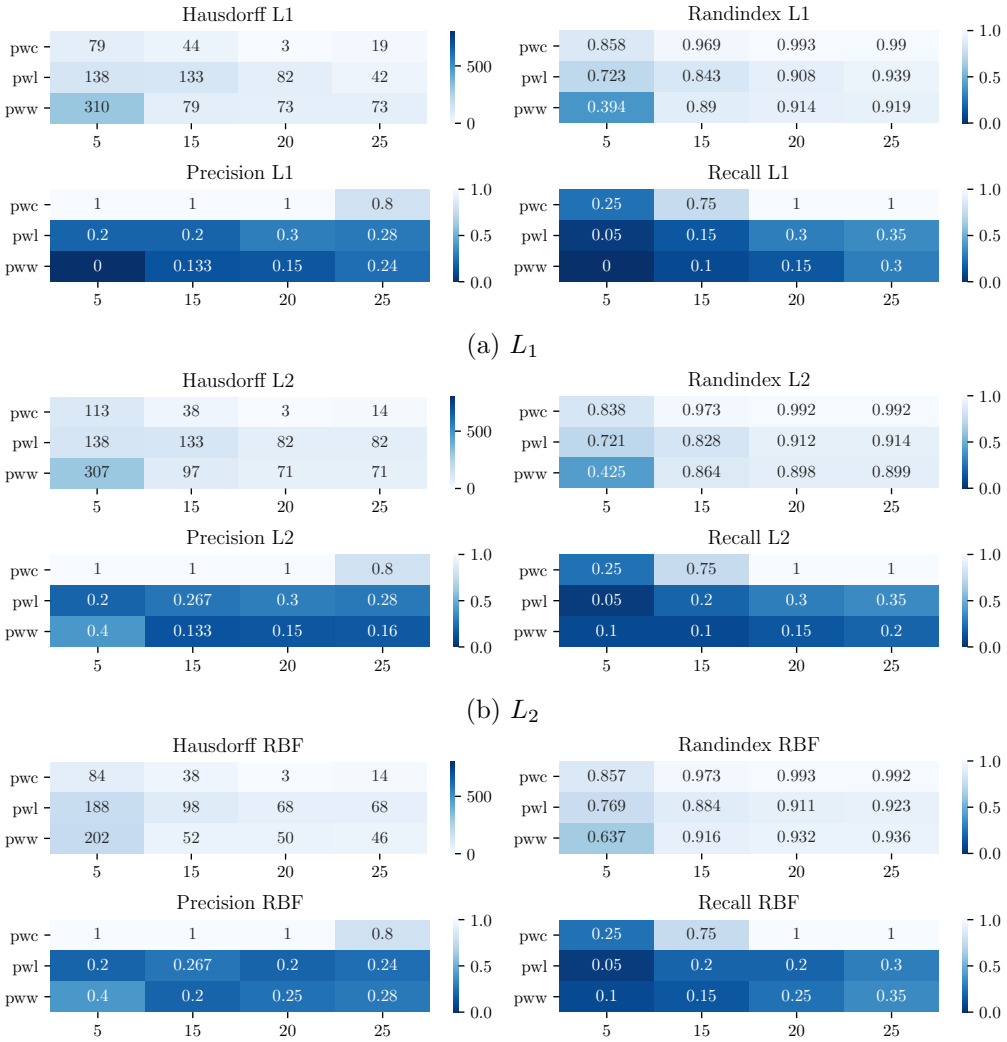


(a) L_1

(b) L_2

(c) *RBF*

Figure C.5: *Dynp* results for noiseless signals using with different cost functions.

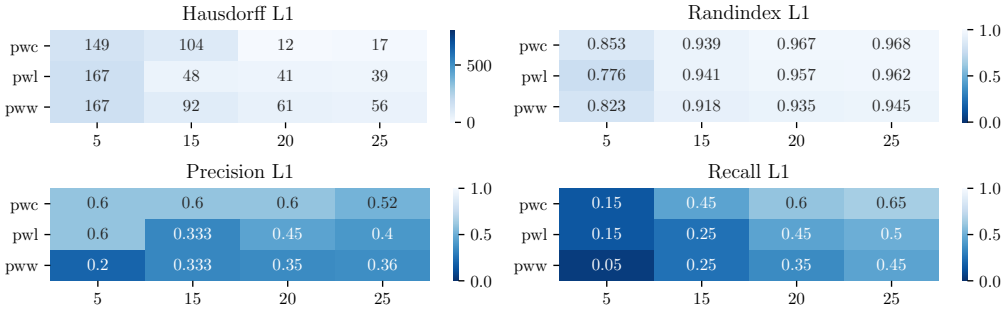


(a) L_1

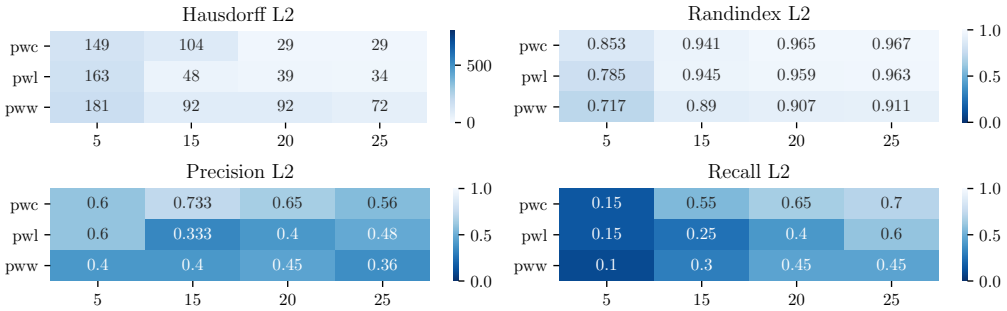
(b) L_2

(c) *RBF*

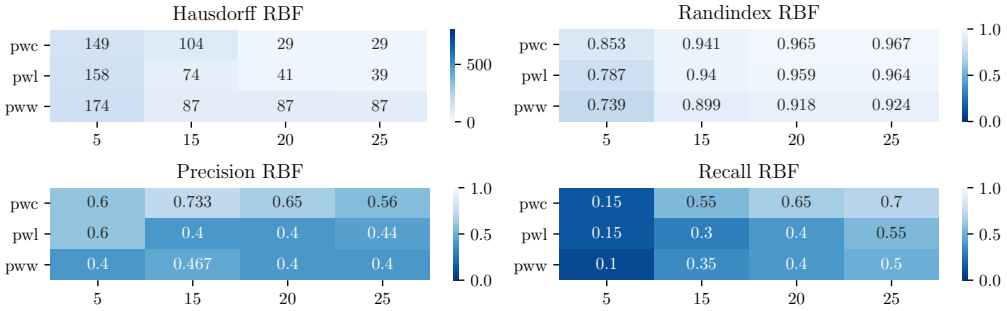
Figure C.6: *Dynp* results for the same signals but with additive white Gaussian noise with a standard deviation of 3. The captions indicate the utilized cost function.



(a) L_1

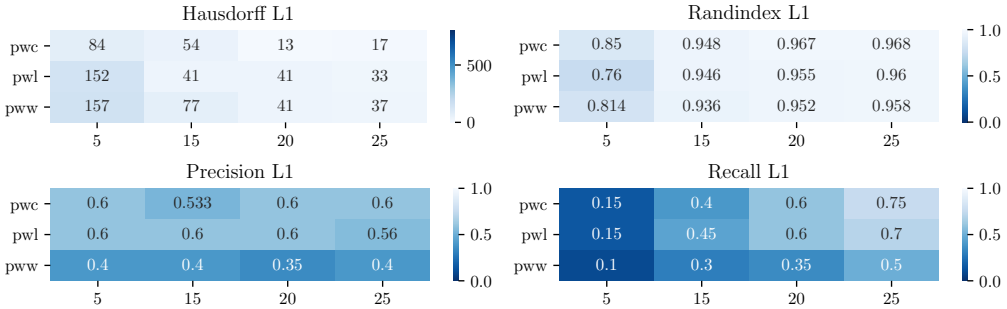


(b) L_2

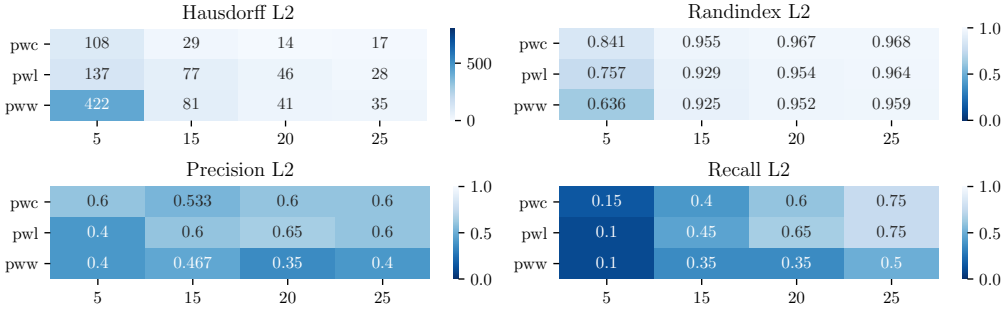


(c) *RBF*

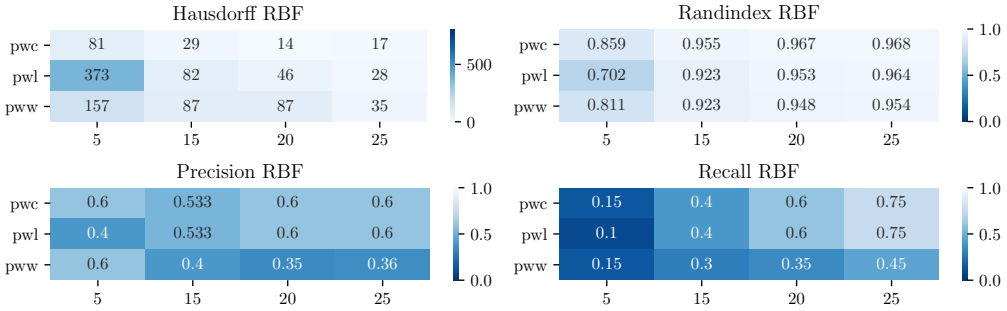
Figure C.7: *Dynp* results for low-pass filtered noiseless signals for different cost functions.



(a) L_1



(b) L_2



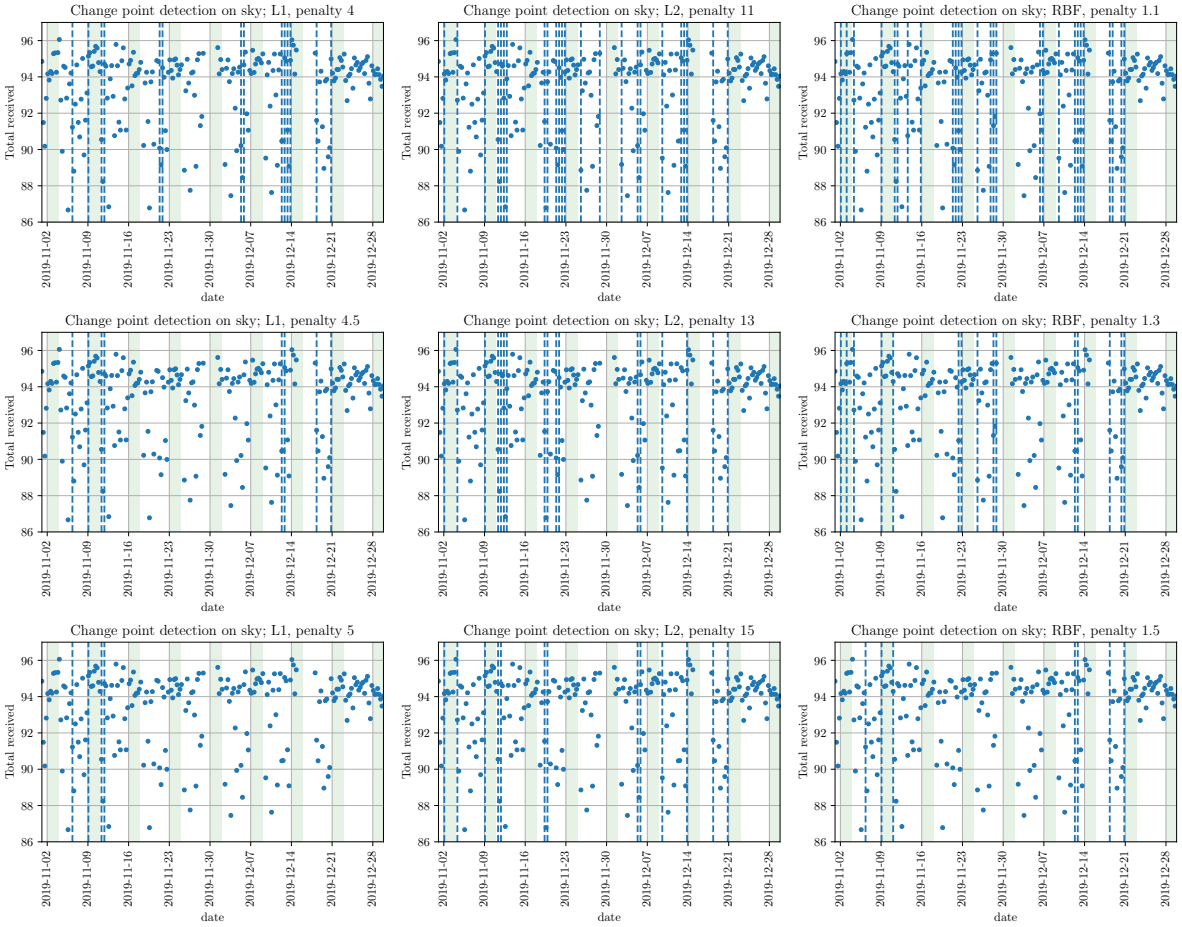
(c) *RBF*

Figure C.8: *Dynp* results for low-pass filtered noisy signals.

Appendix D

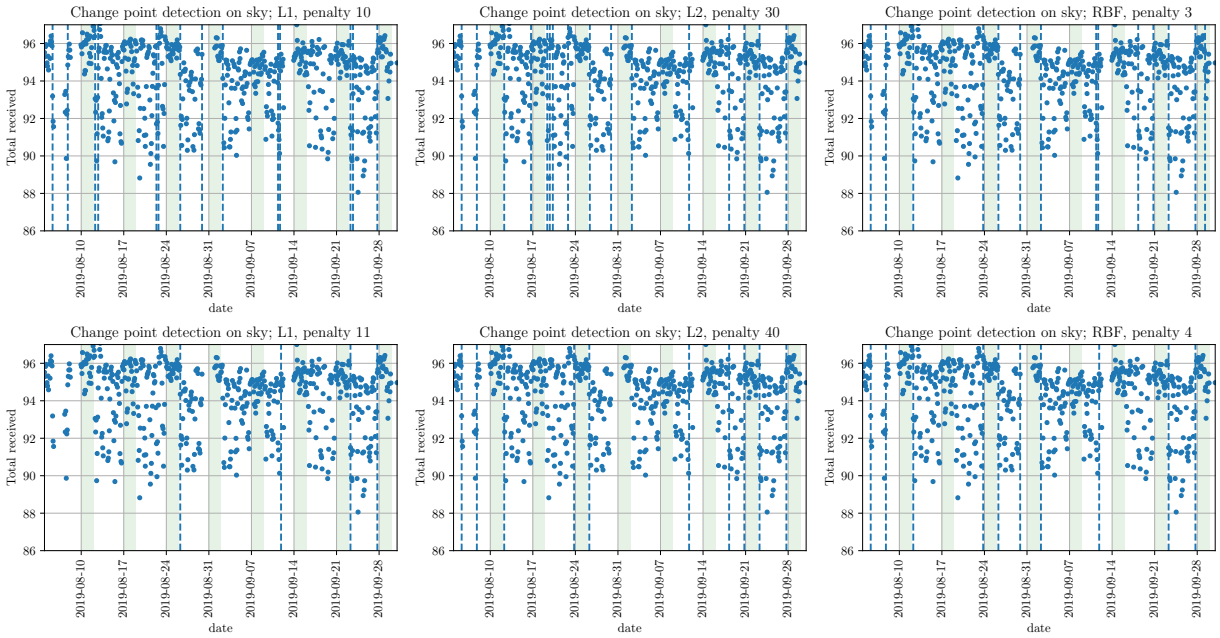
Further FlockLab Figures

D.1 Change Point Detection with no Minimum Size



(a) (b) (c)

Figure D.1: Results of applying *Pelt* to the November and December data using no minimum size restriction. We see that all the applied methods are not robust and that they detect many breakpoints for low penalites.



(a) (b) (c)

Figure D.2: Results of applying *Pelt* to the August and September data using no minimum size restriction. We see that the results are better than in the case of November and December, but for low penalties, often breakpoints are detected very close to each other.

D.2 Change Point Detection on Single Hours

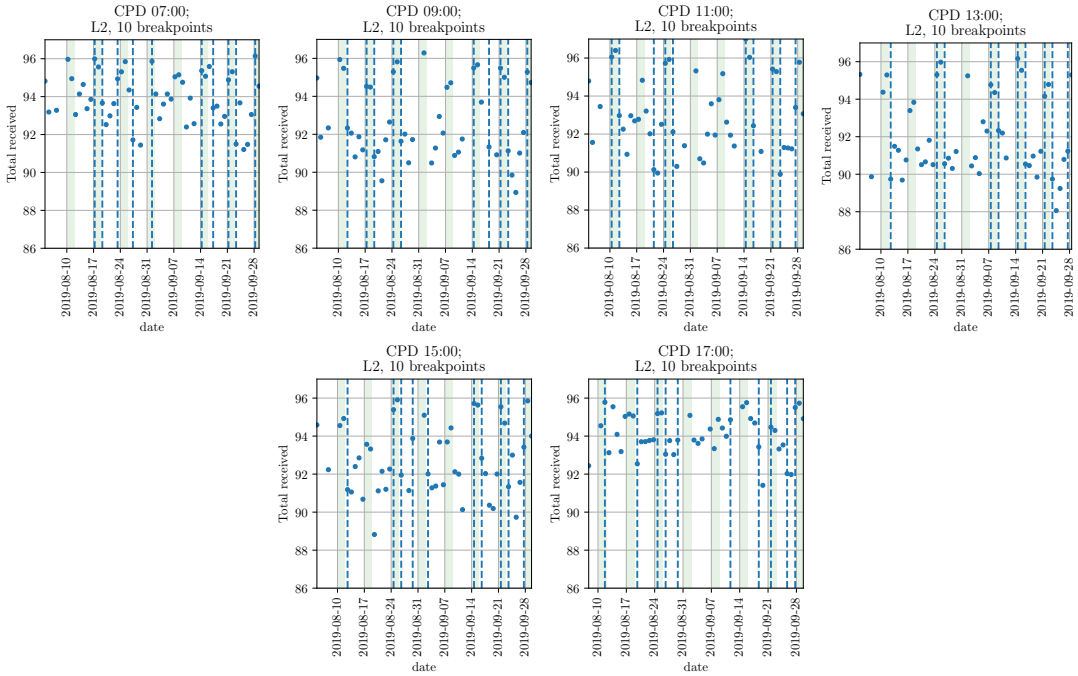


Figure D.3: Change point detection on single hours in August and September using 10 breakpoints, using the L_2 cost. The fit is quite good, but not perfect. This is also true when using 5 and 15 breakpoints (not shown). For other samples, the fit is poor in all cases, and we do not show these plots.

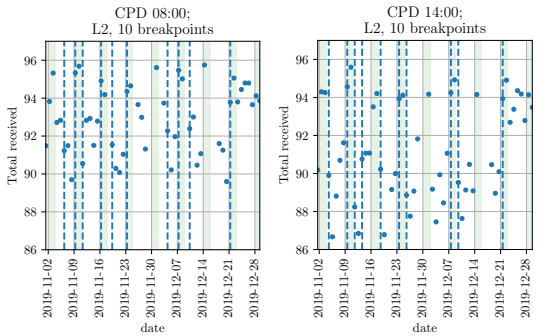


Figure D.4: Change point detection on single hours in November and December using 10 breakpoints, using the L_2 cost. The fit is quite good, but not perfect. This is also true when using 5 and 15 breakpoints (not shown). For other samples, the fit is poor in all cases, and we do not show these plots.

D.3 Full *Dynp* Results

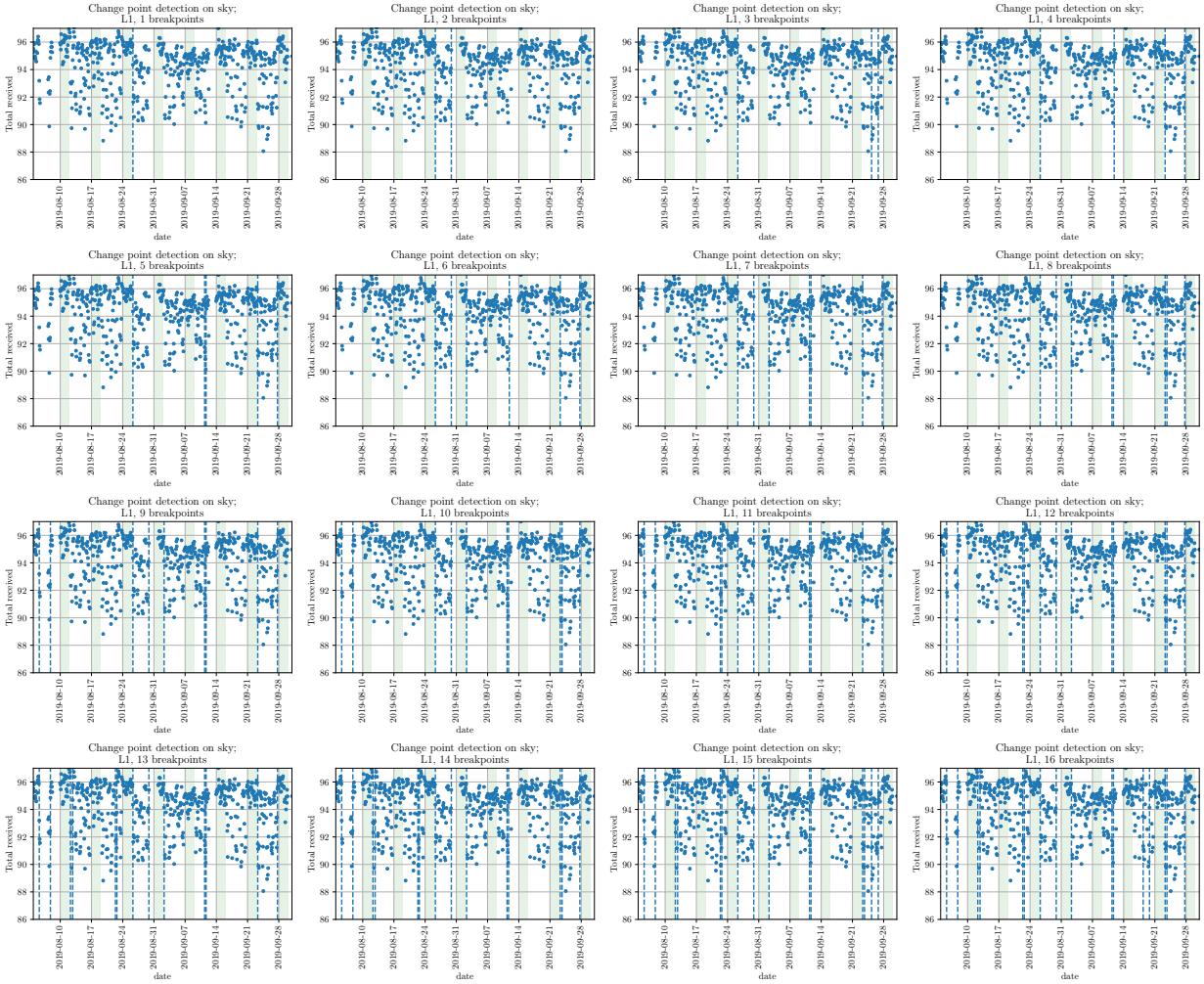


Figure D.5: *Dynp* for 1, 2, ..., 16 breakpoints with the L_1 cost on the August and September data

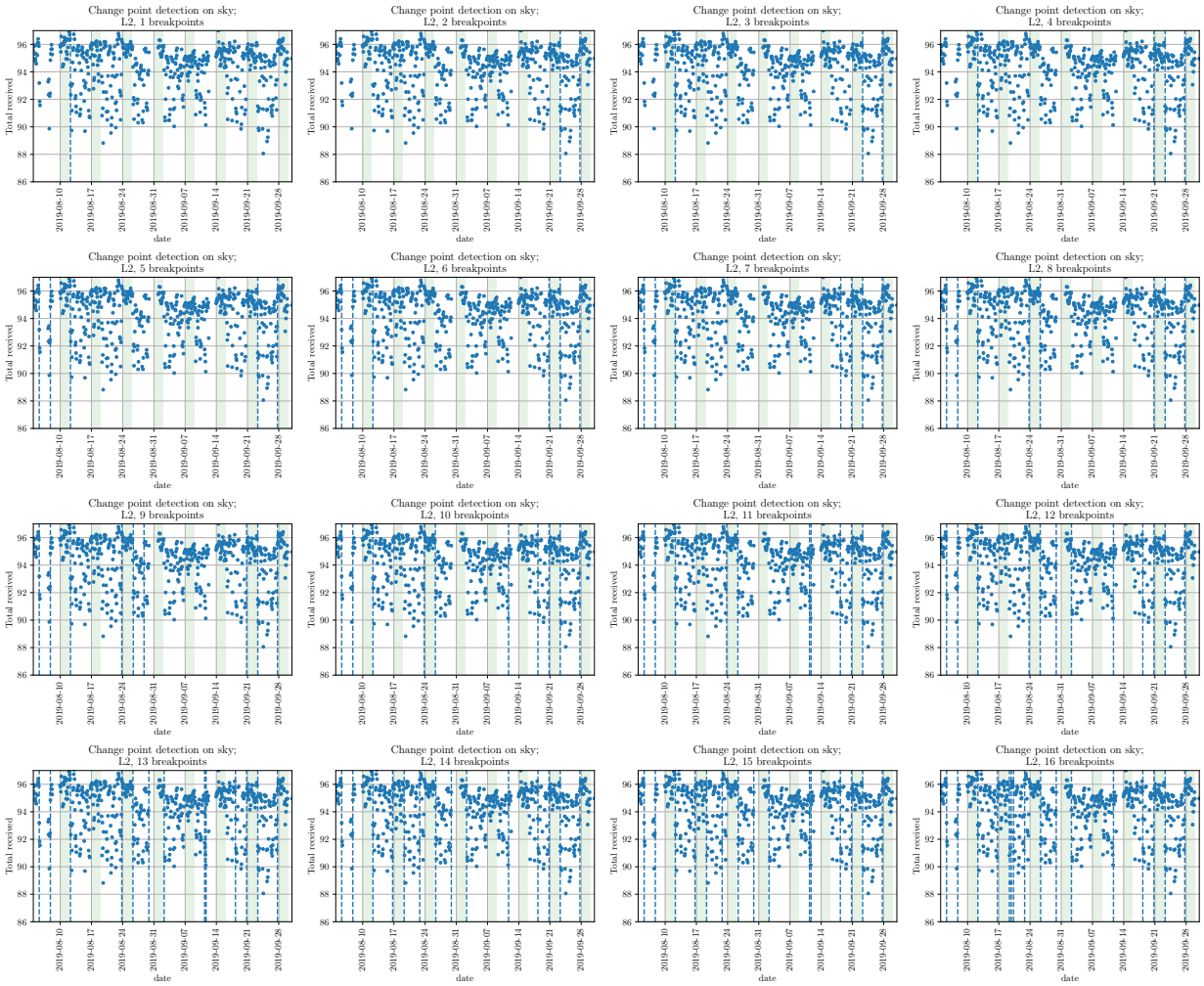


Figure D.6: *Dynp* for 1, 2, ..., 16 breakpoints with the L_2 cost on the August and September data

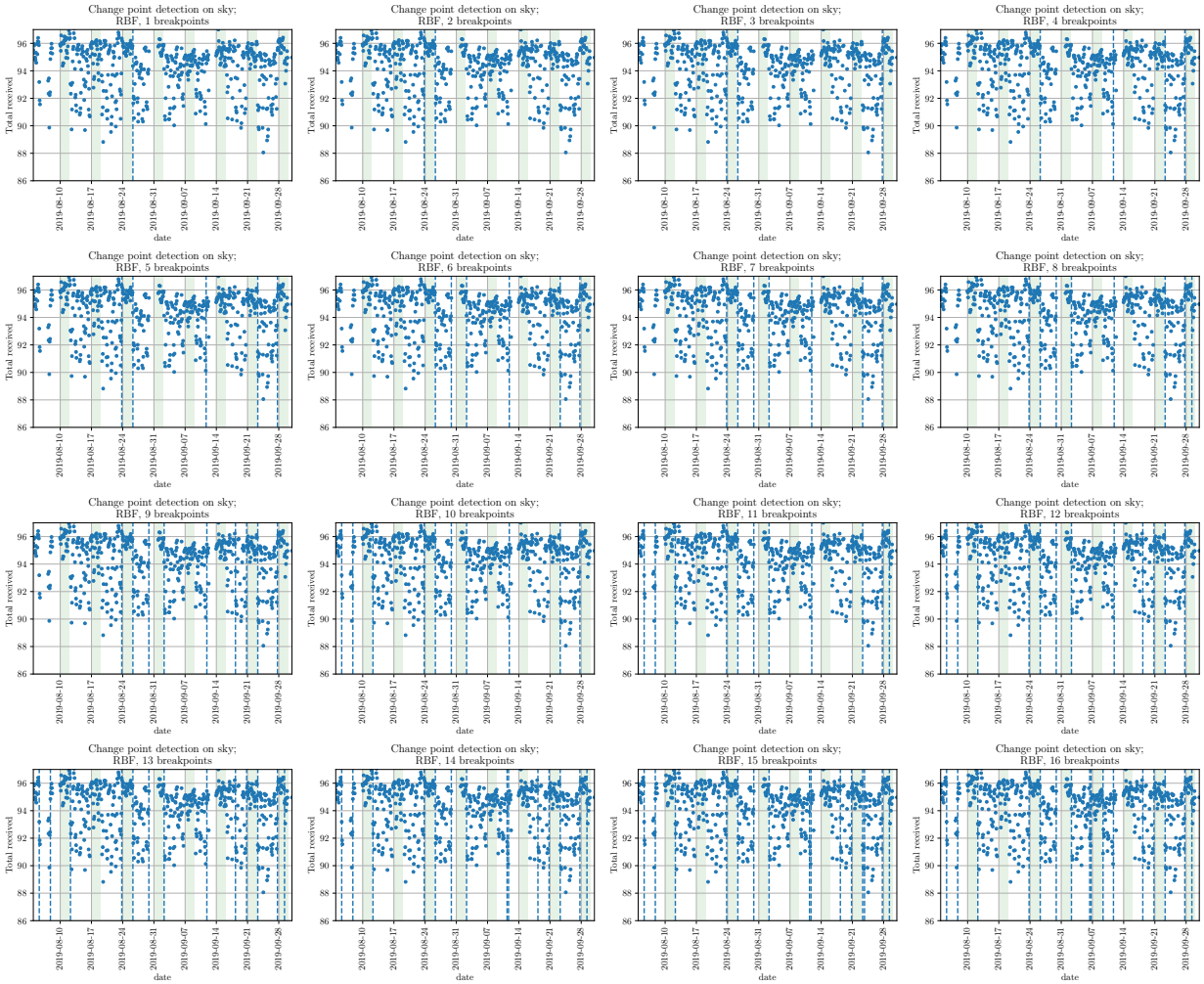


Figure D.7: *Dynp* for 1, 2, ..., 16 breakpoints with the *RBF* cost on the August and September data

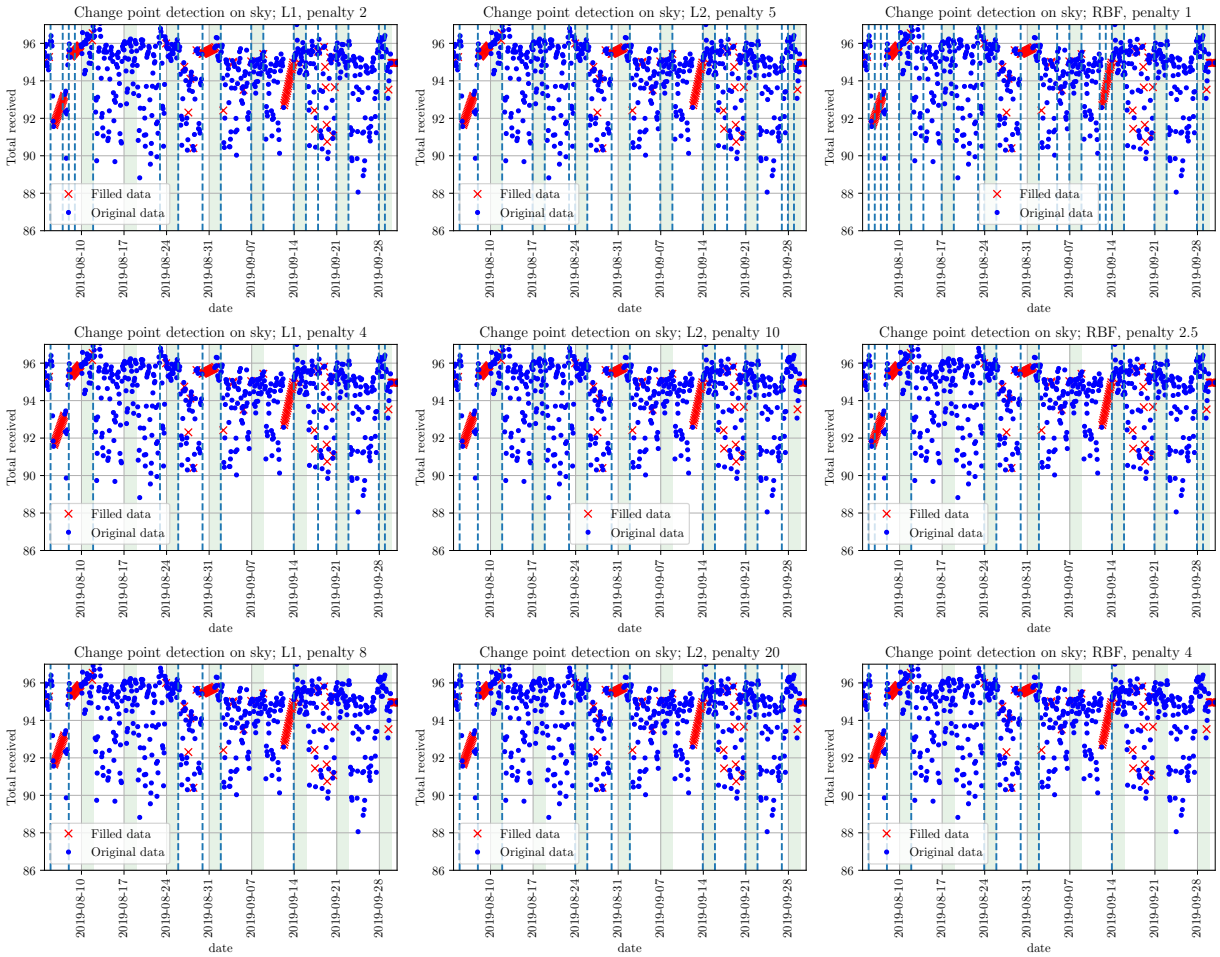
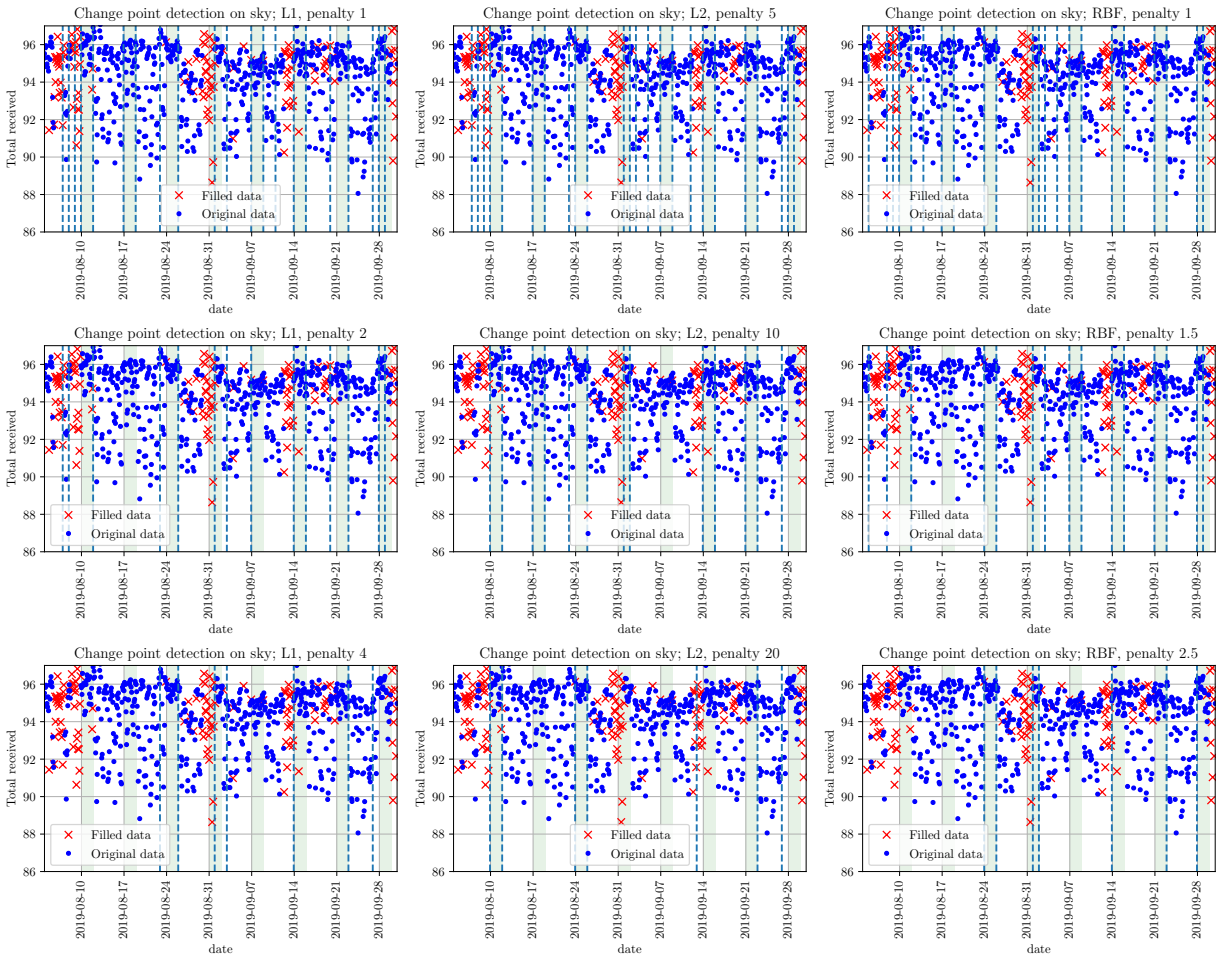


Figure D.8: Results of applying *Pelt* to the August and September data having interploated missing data points. In this setting, the algorithms are restricted to only detecting breakpoints at the boarder between days.

D.4 CPD on Filled FlockLab Data

The *Dynp* and *Pelt* algorithms in *Ruptures* take the parameters *min_size* and *jump*, which respectively define the minimum allowed number of samples between two breakpoints, and the positions at which breakpoints may be assigned, since all breakpoint indices must be divisible by *jump*. We can use this to our advantage, as we already did with the *min_size* parameter when doing change point detection on the full November and December data. Setting *jump* to the number of samples per day would allow us to enforce each day belonging to a unique regime, i.e. that the only changes that we care about are due to differences between days, and not intra-day variations. In this case, we need the same number of samples per day, and we must fill the data similarly to when generating the autocorrelation plots. The results are shown in figures D.8, D.9, D.10 and D.11. We see that these breakpoints match our hypothesis better than when we only use the *min_size* restriction.

By default, *ruptures* sets *jump* to 5, presumably for performance reasons. Restricting the number of indices at which breakpoints may occur greatly reduced the complexity of the change point detection problem.



(a) (b) (c)

Figure D.9: Results of applying *Pelt* to the August and September data having imputed missing data points using KDE. In this setting, the algorithms are restricted to only detecting breakpoints at the boarder between days.

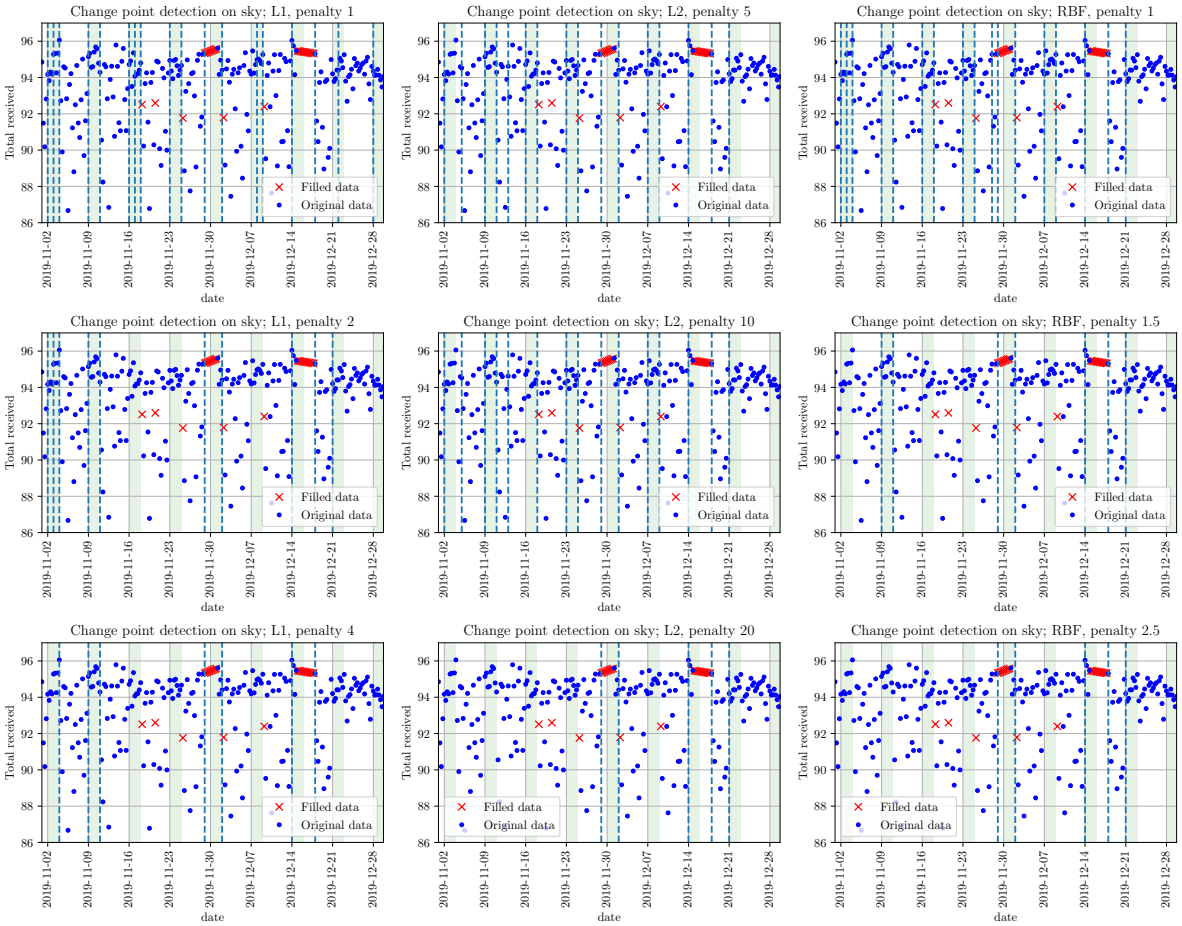


Figure D.10: Results of applying *Pelt* to the November and December data having interploated missing data points. In this setting, the algorithms are restricted to only detecting breakpoints at the boarder between days.

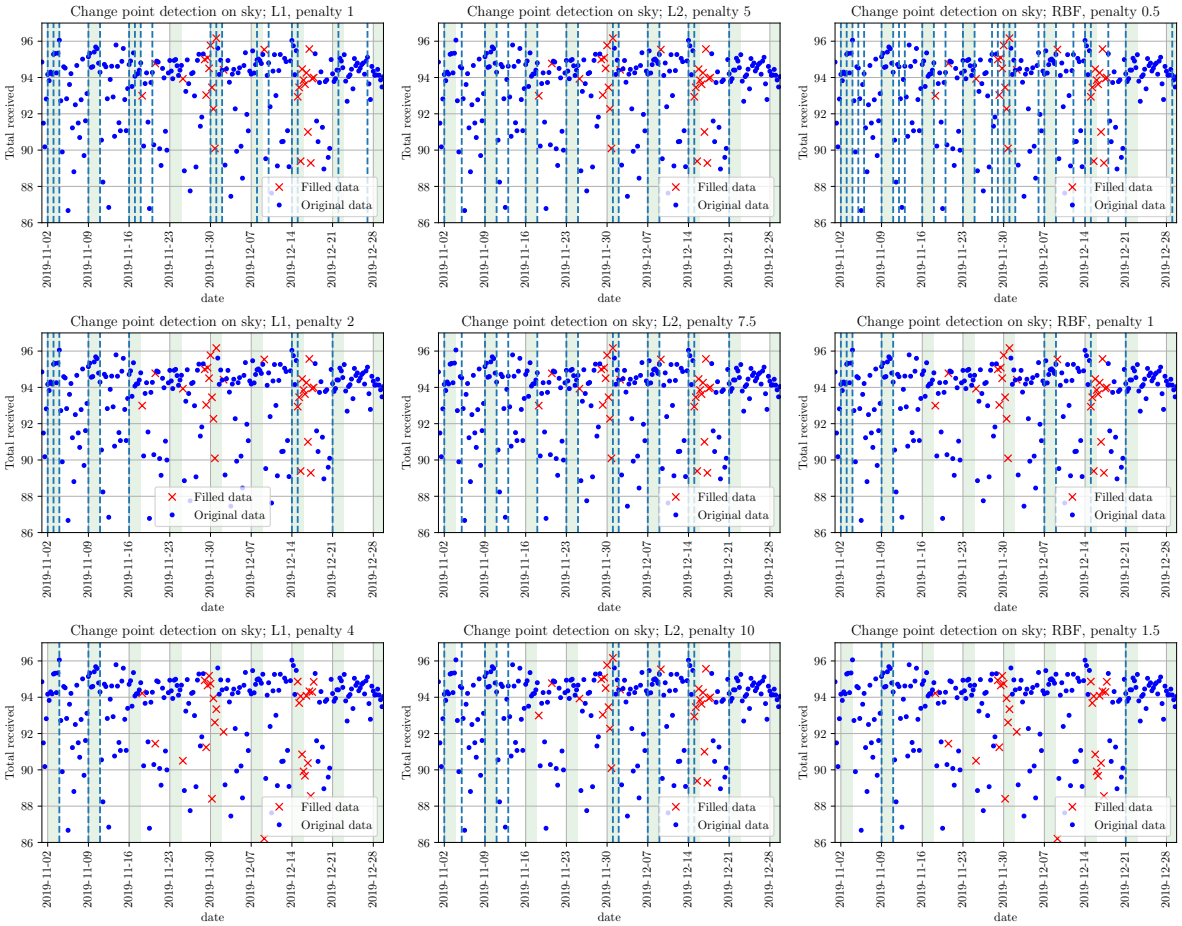


Figure D.11: Results of applying *Pelt* to the August and September data having imputed missing data points using KDE. In this setting, the algorithms are restricted to only detecting breakpoints at the boarder between days.