Implementation & Application of Binary Segmentation Algorithm for Change Point Detection

William Nixon - 13521123[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521123@std.stei.itb.ac.id

***Abstract*— This paper presents the implementation of a binary segmentation algorithm for change point detection in a given dataset. The algorithm offers efficient and fast signal segmentation by recursively detecting change points and splitting the signal into sub-segments with a specified cost function. With a computational complexity of O(nlogn), where n is the number of samples, the algorithm is suitable for large datasets. It is also customizeable and can run with multiple custom cost functions. Experimental evaluations on the algorithm includes a case study on segmentation in Bandung's COVID Dataset.**

***Keywords*— Binary Segmentation Algorithm, Algorithm, Change Point Detection, Implementation, Signal Segmentation, Computational Complexity, COVID Dataset.**

I. Introduction

Change point detection is a fundamental problem in various domains, including signal processing, finance, and environmental monitoring. The ability to identify and characterize transitions in data is crucial for understanding underlying patterns, making informed decisions, and taking appropriate actions. In this paper, we focus on the implementation of a binary segmentation algorithm for change point detection in signals and its application to COVID-19 data in Indonesia.

The binary segmentation algorithm offers a sequential approach to signal segmentation, where a single change point is detected in the complete input signal. The signal is then split into two sub-signals based on this change point, and the process is recursively applied to each sub-signal. This approach enables efficient segmentation of signals with a computational complexity of O(nlogn), where n represents the number of samples.

Through this study, we aim to highlight the practical utility of the binary segmentation algorithm for change point detection and its relevance in addressing real-world problems such as monitoring and managing the COVID-19 pandemic. The choice of using COVID-19 data for testing the application of binary segmentation algorithm is motivated by the nature of the pandemic.

The number of active cases in a given region can be considered as a change point, fluctuating over time. By detecting these change points, we can identify periods with rising or falling cases, helping to assess the severity of the situation and determine appropriate measures for disease control.

II. Definitions

A. Change Point Detection

Change point detection refers to the task of identifying points or locations in a sequence of data where the underlying statistical properties of the data change. These changes can manifest as shifts in mean, variance, distribution, or other relevant characteristics. A change point, also known as a structural break, is the specific point in the sequence where the change in properties occurs. Change point detection aims to locate and characterize these change points, providing insights into the dynamics and transitions within the data.

Change point detection problems usually consists of 3 components, which include:

1. Search Function, the search function is responsible for exploring and examining the potential change points in the input sequence. It systematically scans the data and identifies candidate locations where a change in the underlying properties may occur.
2. Cost Function, measures the degree of change at a specific location or split point in the input sequence. The cost function serves as a criterion for selecting the most suitable change points based on fit.
3. Constraint, determines the point at which a given search function should stop searching.

Together, the search function and the cost function form the core components of a change point detection algorithm. The search function identifies potential change points in the data, while the cost function evaluates the significance of these points based on the observed discrepancies, while satisfying the constraint. By combining these components, change point detection algorithms can efficiently and effectively locate and characterize transitions in the input sequence.

A change point detection algorithm usually accepts a set of input sequence X as an argument, returning the index of change points in said sequence as follows:

The input sequence: $X = \{x_1, x_2, ..., xn\}$
The index of the change point: t ($t \in \{1, 2, ..., n-1\}$)

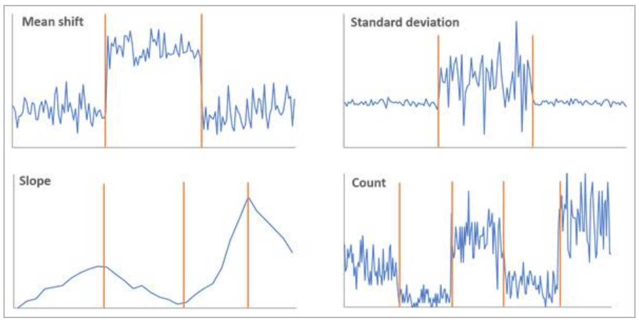


Figure A.1. Change Points in Various Types of Distribution Shift Denoted by Orange Vertical Lines (Source: https://pro.arcgis.com)

B. Some Cost Functions

Some of the cost functions that we are going to implement in the paper are:

1. Least Absolute Deviation (LAD/L1):
   This cost function detects changes in the median of a signal. The Least Absolute Deviation (L1-norm or absolute error), is a robuest estimator of a shift in the central point (mean, median, mode) of a distribution. It computes the sum of the absolute differences between the observed data points and the median between the interval. It minimizes the sum of the absolute residuals, making it robust against outliers.

   Objective function: minimize $\Sigma|Y_i - \hat{Y}_i|$
   Notation: min $\Sigma|Y_i - \hat{Y}_i|$

2. Least Squared Deviation (LSD/L2):
   This cost function detects mean-shifts in a signal. The Least Squared Deviation cost function (L2-norm or mean squared error), calculates the sum of squared differences between the observed data points and the estimated average values. It minimizes the sum of squared residuals and is commonly used in linear regression problems.

   Objective function: minimize $\Sigma(Y_i - \bar{Y_i})^2$
   Notation: min $\Sigma(Y_i - \bar{Y_i})^2$

C. Binary Segmentation (Search Function)

Binary segmentation is search function in change point detection that follows the divide and conquer paradigm. In binary segmentation, the input sequence is divided into two sub-segments by detecting a single change point. The algorithm recursively applies the same process to each resulting sub-signal until no further change points are detected or a stopping criterion is met.

The steps required by the search function in determining the change points are:

1. Initialization: Define the initial interval as the full segment, set cost thresholds and hard limits.
2. Segmentation: Divide the initial interval into two sub-segments, evaluate the cost of each sub segment independently. The cost function will be fitted into the sub-segments.
3. Cost evaluation: Cost function will compute the cost of each fitted sub-interval in phase 2. The purpose of this cost evaluation is to gain a numeric value of the effectiveness of the segmentation itself.
4. Change point determination: If the cost of either sub-interval exceeds the predefined threshold or significance level, then a potential change point is found. We will select the subsegment with the highest cost as the candidate for change point.
5. Recursive process: Recursive application of step 1-4 will be done on sub-interval before the candidate change point and the sub-interval after it. The process recursively continues while a hard limit for change point has not been reached, or the cost of any sub interval has not fallen below a predefined limit.

By combining the search function (segmentation) and the cost function, binary segmentation iteratively identifies change points by evaluating the cost of sub-intervals and splitting them until no further significant changes are detected.

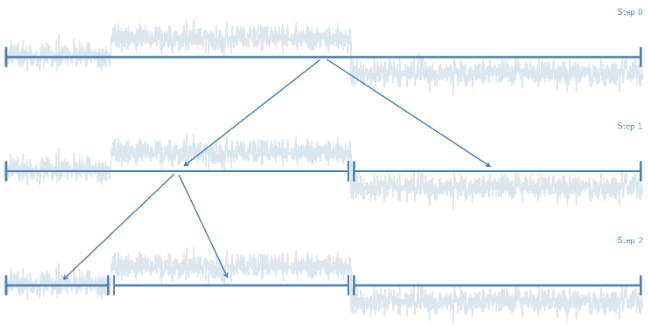


Figure C.1. Binary Segmentation In Action, Segmenting A Given Signal Whenever A Change Point Has Been Detected (Source: https://centre-borelli.github.io/ruptures-docs/user-guide/detection/binseg/)

D. Binary Segmentation Search as a Divide and Conquer Algorithm

A divide and conquer algorithm is a problem-solving approach that involves breaking down a complex problem into smaller sub-problems, solving them independently, and then combining the solutions to solve the original problem. The main idea is to divide the problem into manageable parts, conquer each part by solving it recursively or iteratively, and then merging the solutions to obtain the final solution. This technique is often employed when the problem exhibits overlapping sub-problems or can be naturally divided into smaller instances.

Binary segmentation can be considered a divide and conquer algorithm due to its iterative nature and the recursive splitting of the input signal.

The two distinct steps are as follows:

1. Divide: The algorithm starts with the complete input sequence and detects a change point, dividing it into two sub-signals. This process is then repeated on each resulting sub-signal until the desired level of segmentation is achieved.
2. Conquer: Upon being bounded by a constraint (no further changes detected), each sub-segment would return the changepoint detected from itself. The final change point array is a concatenation of all of each sub-segment's change point.

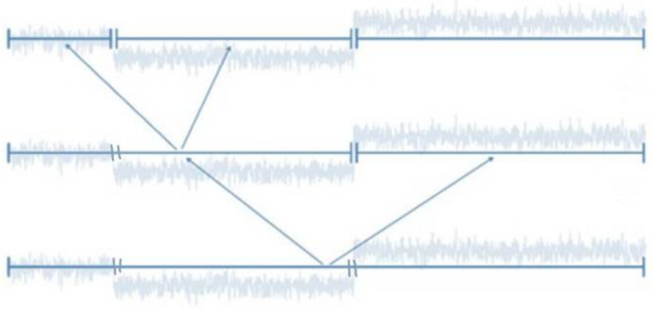


Figure D.1. Conquer Phase, Propagating Sub Segment's Change Point Back To Original Segment (Source: https://centre-borelli.github.io/ruptures-docs/user-guide/detection/binseg/)

E. *Several COVID19 Dataset Metrics*

COVID-19 active cases refer to the number of individuals who are currently infected with the COVID-19 virus and are actively undergoing treatment or isolation. It represents the difference between the total number of confirmed COVID-19 cases and the sum of recovered cases and deaths.

COVID-19 daily recoveries refer to the number of individuals who have recovered from the COVID-19 virus within a 24-hour period. It represents the number of people who were previously infected and have now completed their treatment or isolation successfully.

Tracking both metrics provides crucial information about the current state of the COVID-19 outbreak in a specific region or population. Utilizing change point detection techniques on COVID-19 active cases can help identify significant shifts in the infection rate, indicating periods of increased transmission or the effectiveness of control measures. This information can aid in making informed decisions regarding public health interventions and resource allocation.

III. Implementation And Analysis

We will be implementing Binary Segmentation and 2 Cost Functions, L1 and L2 Cost to compare the differences both these cost functions can make. The code will be implemented in Python, and the excerpt found in this document will only include snippets important parts, in other words, it is not complete / compilable. Please refer to Part V for the complete source code.

A. *Implementation of Cost Function*

```
class CostL1:

    def error(self, start, end) -> float:
        if end - start < self.min_size:
            raise NotEnoughPoints
        sub = self.signal[start:end]
        med = np.median(sub, axis=0)

        return abs(sub - med).sum()
```

```
class CostL2:

    def error(self, start, end):
        if end - start < self.min_size:
            raise NotEnoughPoints
        return self.signal[start:end].var(axis=0).sum() * (end - start)
```

Both cost functions classes implement a function called cost, which returns the corresponding cost for values in range from start till end. CostL1 Takes the sum of absolute difference between all of the values and the median value, while CostL2 returns the sum of all the variances in the interval, multiplied by the end and the start date to account for the number of ranges being evaluated.

B. *Implementation of Binary Segmentation*

```
class Binseg():

    def _seg(self, n_bkps=None):
        bkps = [self.n_samples]
        stop = False
        while not stop:
            stop = True
            new_bkps = [
                self.single_bkp(start, end) for start, end in pairwise([0] + bkps)
            ]
            bkp, gain = max(new_bkps, key=lambda x: x[1])

            if bkp is None:  # all possible configuration have been explored.
                break

            if n_bkps is not None:
                if len(bkps) - 1 < n_bkps:
                    stop = False

            if not stop:
                bkps.append(bkp)
                bkps.sort()

        partition = {
            (start, end): self.cost.error(start, end)
            for start, end in pairwise([0] + bkps)
        }
        return partition

    def single_bkp(self, start, end):
        segment_cost = self.cost.error(start, end)
        if np.isinf(segment_cost) and segment_cost < 0:  # if cost is -inf
```

```
            return None, 0
        gain_list = list()
        for bkp in range(start, end, self.jump):
            if bkp - start >= self.min_size and end - bkp >= self.min_size:
                gain = (
                    segment_cost
                    - self.cost.error(start, bkp)
                    - self.cost.error(bkp, end)
                )
                gain_list.append((gain, bkp))
        try:
            gain, bkp = max(gain_list)
        except ValueError:  # if empty sub_sampling
            return None, 0
        return bkp, gain

    def fit(self, signal):
        self.signal = signal.reshape(-1, 1)
        self.n_samples, _ = self.signal.shape
        self.cost.fit(signal)

        return self

    def predict(self, n_bkps=None):
        partition = self._seg(n_bkps=n_bkps)
        bkps = sorted(e for s, e in partition.keys())
        return bkps
```

The _seg function is a method that performs the actual segmentation. It takes an optional parameter n_bkps, which specifies the desired number of breakpoints. Initially, the function sets the breakpoints to include the entire length of the input data. It then will divide and conquer the segment iteratively, entering a loop where it tries to find the best breakpoint based on the cost function. It calculates the cost for all possible breakpoints and selects the one with the highest gain. Upon doing so, it will create new subsegments for the loop to evaluate (Divide). For each subsegment explored, it will append the results to a global array bkps (Conquer). If n_bkps is specified, the loop continues until the number of breakpoints reaches n_bkps. It returns a partition dictionary that contains the start and end indices of each segment along with their associated error costs.

The single_bkp function calculates the gain for a single breakpoint given a start and end index. It first computes the error cost for the entire segment. If the segment cost is negative infinity or less than 0, it means the cost is invalid, and the function returns None and 0 gain. Otherwise, it iterates over possible breakpoints within the segment, considering only those that satisfy the minimum segment size condition. For each valid breakpoint, it calculates the gain by subtracting the error costs of the left and right segments from the total segment cost. The function returns the breakpoint and its corresponding gain.

The fit function is used to prepare the input data for segmentation. It takes a signal parameter. The function reshapes the signal to have a single column and determines the number of samples. It then fits the cost function to the signal, which is necessary for subsequent calculations.

The predict function is the main interface for obtaining the segmented breakpoints. It takes an optional parameter n_bkps to specify the desired number of breakpoints. It calls the _seg function to obtain the partition dictionary. It then extracts the start and end indices from the dictionary keys and sorts them to obtain the breakpoints in ascending order. The function returns the sorted list of breakpoints.

C. Application in Bandung COVID19 Dataset Data

The data for this study was obtained from the official website of the West Java Provincial Government. It is publicly available at https://opendata.jabarprov.go.id/id/dataset/perkembangan-harian-kasus-terkonfirmasi-positif-covid-19-berdasarkan-kabupatenkota-di-jawa-barat.

The dataset was downloaded as a zip file containing an Excel file with metrics data for daily deaths, active cases, and more. The data was loaded into a Python notebook for analysis. The implemented binary segmentation algorithm was then applied to detect change points in the COVID-19 active cases data. The algorithm identified significant transitions in the number of active cases, indicating potential changes in the virus spread. The change points were determined using cost functions such as Least Absolute Deviation (Cost1) or Least Squared Deviation (Cost2).

The detected change points were then visualized using Matplotlib, a Python library for data visualization. Matplotlib enabled the creation of informative graphs representing the COVID-19 active cases over time, highlighting the identified change points.

D. Analysis Of Results

Subsections D.1. and D.2. will show the results of analysis obtained using the algorithm and an insight as to how change point intervals can be used applicatvely.

D.1. Bandung's Active Cases

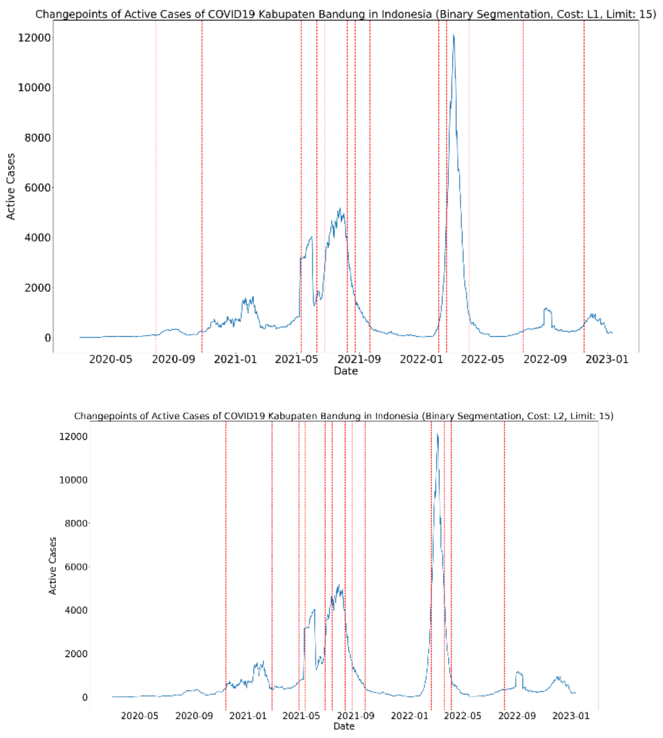


Figure D.1.1 Segmentation of Active Cases Using L1 (a) and L2 (b) Cost Function (Source: Writer)

The two above graphs show the different change points of the number of active COVID19 cases in Bandung City, denoted by vertical red lines, given by the binary segmentation using two L1 and L2 Cost Function. A hard limit of 15 change points (vertical lines) were given to both functions.

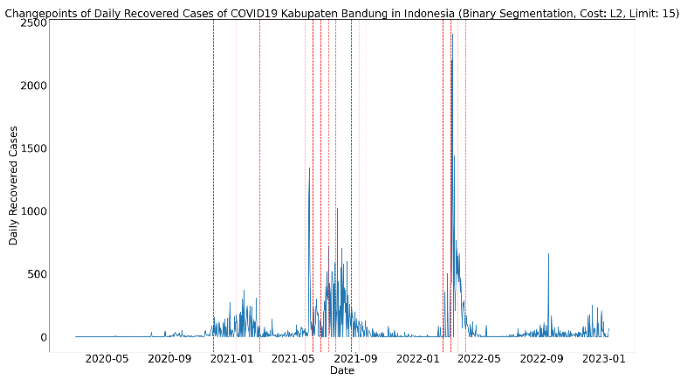
Although the two algorithms differed in how the change points were allocated in the graph, the algorithm suceeded in segmenting and detecting the change points that it deemed existed in the graph. In this case, the L1 Cost function seemed to have performed better than L2, allocating change points more sparingly. The Cost L2 function placed higher error value on sudden spikes, and as the result, ignored most of spikes that were small (most distribution changes were given when cases spiked).

This is because the L1 Cost Function depends on the median of the data on an interval, compared to the average function which with L2. The sudden spikes caused a huge impact on the average of values within the interval, as the result, the change points were detected by CostL1 there. By using the median instead of the mean, CostL1 is more robust to outliers / sudden distribution changes.

One way of extracting insight from the figure is by looking at the trend of generated change point interval. For example, using Figure D.1.1.a, Interval 0 (03/2020 – 11/2020) showed a relatively low amount of active cases, while interval 1 (11/2020 – 4/2021) showed a rise in infection, before seeing a huge spike and drop at cases on interval 2 (4/2021 – 7/2021).

From that result, we can then hypothesize for the reason behind the change in distrubution. For example, the reason behind interval 0 might be that at the COVID pandemic hasn't been common yet in Bandung during March – November 2020. As the infection spreads to the city, there is a gradual steady increase of cases on November 2020 – Mei 2021, before cases rapidly soared and dipped on Mei 2021 – July 2021, most probably as the result of governmental measures such as lockdown to contain the pandemic.

D.2. Bandung's Daily Recovered Cases



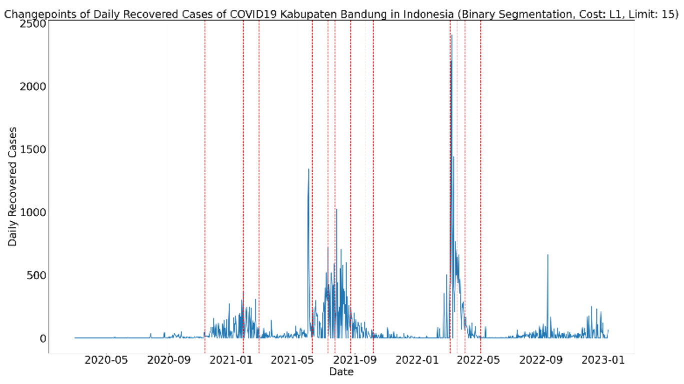


Figure D.2.1. Segmentation of Daily Recovered Cases using L1 (a) and L2 (b) Cost Function (Source: Writer)

The two above graphs show the different change points of the number of daily recovered COVID19 cases in Bandung City, similar to point D.1. with a hard limit of 15 change points.

The algorithm suceeded in segmenting and detecting the change points that it deemed existed in the graph. The two algorithms differed in how the change points were allocated in the graph, however both cost functions seemed to have performed similarly. The allocation of change points seemed not very optimal as they were clustered tighly around spikes.

In this case, both functions placed many change points on spikes and sudden increases. This is because on the original data distribution, cases were not distributed equally across days, and instead were distributed as "spikes". This is due to the nature of the metric being collected itself.

Figure D.1.1 as the active case graph was more continuous than Figure D.2.2. This is because people that have recovered from COVID will only be counted once, meanwhile people that have COVID will have several active cases count, ranging several days, starting from they were sick until they recover. As the result, the underlying distribution for active cases was more smooth compared to recovered cases, which were more discrete in nature.

CostL1 and CostL2 as simple cost functions that accounts for average/mean did not perform very well because of the discrete nature of the dataset, being treated as outliers that skew the result. To gain better results, preprocessing discrete data or using an alternative more robust cost functions like Gaussian Process Change or Kernelized Mean Change can be used instead.

IV. Conclusion And Suggestions

In conclusion, this paper presented the implementation of a binary segmentation algorithm for change point detection. The algorithm follows a divide and conquer approach, recursively splitting the input signal based on detected change points. The algorithm was applied to COVID-19 active cases data in Bandung, showcasing its effectiveness and customizability with various cost functions. The change points identified using the binary segmentation algorithm can provide valuable insights for extracting distributions out of a given data, such as monitoring the COVID-19 situation, identifying periods of rising or falling dynamic of infection rates.

Future works that can extend this paper include experimentation on the various other types of cost function that are slope/model based and are not mean distribution shift based like CostL1 or CostL2. Other types of search functions like using the Dynamic Programming approach can also be used to do the segmentation, rather than Binary Segmentation.

V. Miscellaneous

The source code from the program used for this paper can be accessed from the following link:

https://colab.research.google.com/drive/1HB9Ksntwv6F930EfEJXI7gfuW2hUlwwA?usp=sharing

The video briefly explaining about the paper can be found on the following link:

https://www.youtube.com/watch?v=PMzVevzTd4E

VI. Acknowledgements

The author would like to express sincere gratitude and praise to the Almighty God for the blessings and facilitation provided throughout the completion of this paper entitled "Implementation & Application of Binary Segmentation Algorithm for Change Point Detection". The author would also like to extend appreciation to all the instructors of the IF2211 Algorithm Strategies course, particularly Dr. Rinaldi Munir, for their guidance and teachings during this semester.

Furthermore, the author would like to express gratitude to their family and friends for their support and encouragement throughout the duration of this semester. Their presence and words of encouragement have been a source of strength and motivation.

References

[1] C. Truong, L. Oudre, N. Vayatis. (2020). Selective review of offline change point detection methods. Signal Processing, 167:107299.

[2] H. Poor. (1985). *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag.

[3] Bai, J. (1997). Estimating multiple breaks one at a time. Econometric Theory, 13(3), 315–352.

[4] Fryzlewicz, P. (2014). Wild binary segmentation for multiple change-point detection. The Annals of Statistics, 42(6), 2243–2281.

[5] Bai, J. (1995). Least absolute deviation of a shift. Econometric Theory, 11(3), 403–436.

[6] Munir, Rinaldi. (2022). Divide and Conquer (Bagian 2). Accessed from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian2.pdf on 20 May 2023

[7] Munir, Rinaldi. (2022). Divide and Conquer (Bagian 3). Accessed from https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Divide-and-Conquer-(2021)-Bagian3.pdf on 20 May 2023

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 20 Mei 2023

[illegible]

William Nixon 13521123