Approximation Algorithms for Segment k-Means Clustering

Vancea, Vlad

Award date:
2023

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Mathematics and Computer Science
Cluster Algorithms, Geometry and Applications

Approximation Algorithms for Segment k-Means Clustering

Master Thesis

Vlad Vancea

Supervisor:
Assistant Professor Morteza Monemizadeh

Defense Committee:
Assistant Professor Morteza Monemizadeh
Associate Professor Cassio de Campos
Assistant Professor Tim Ophelders

Eindhoven, October 2023

Contents

Contents **i**

List of Figures **iii**

1 Introduction **3**
1.1 Motivation 4
1.2 Contribution 6
1.2.1 Approximation Algorithm 6
1.2.2 Coreset Construction 7

2 Related Work **8**
2.1 Point Center k -means and median 8
2.2 Segment k-means 9
2.3 Coreset Construction 10

3 Segment k-means Approximation Algorithm **12**
3.1 Segment k-means Algorithm 12
3.1.1 Algorithm Correctness 16
3.1.2 Representation Size Analysis 18
3.1.3 Approximation Factor 19
3.1.4 Performance Analysis 23

4 Coreset Algorithm **25**
4.1 (k, ϵ)-Coreset Algorithm 25
4.1.1 Approximation Analysis 27
4.1.2 Coreset Size 30

5 Experimental Results **31**
5.1 Datasets 31
5.1.1 Synthetic Datasets 31
5.1.2 Real-world Datasets 32
5.2 Segment k-means Dynamic Programming Algorithm 33
5.3 Experimental Setup 33
5.3.1 Algorithms Comparison Experimental Setup 33
5.3.2 Approximation Algorithm Experimental Setup 33
5.4 Experiments 34
5.4.1 Synthetic Datasets 34

5.4.2 Real-world Datasets 48
5.5 Performance of the Segment k-means Algorithm 52
5.5.1 Running Time Comparison 53
5.5.2 Large Datasets 53

6 Conclusions **54**
6.1 Discussion 54
6.2 Future Work 55

Bibliography **56**

List of Figures

1.1 Sample points $p_i, p_j \in P$ and their shortest distance to a segment $s \in \mathscr{S}$ 4
1.2 Sample blobs and aniso datasets. 5
1.3 Amazon Stock Open Price 5

3.1 Figure showing all of the copies of a segment s with $\theta = \frac{\pi}{4}$ 15
3.2 Segment L_j^* along with points inside $R(L_j^*, \ell, 4r)$ 20
3.3 Rectangle $R(\overline{p'q'}, 3\ell, 8r)$ showing to entirely cover $R(L_j^*, \ell, 4r)$ 20
3.4 Sample cluster around optimal segment L_j^*, together with segment $\overline{p'q'}$. 21

4.1 Sample segment b_i showing two of its associated grids. One grid having measurements, $3\ell, 4r$, while the larger grid being double in size. Each grid consists of cells, where the smaller grid has cells of side length $\epsilon \times \min(3\ell, 4r)$, while the larger grid has cells double in size. 27
4.2 Point in P that is as close to B as it is to O 28
4.3 Point in P that is closer to B than to O. 28
4.4 Point in P that is closer to O than to B 29

5.1 Segment produced by SLS and algorithm 1 on the vertical dataset. 35
5.2 Result produced by the dynamic programming algorithm on the two-horizontal dataset. 35
5.3 Result produced by the approximation algorithm on the two-horizontal dataset. . . 36
5.4 Sample result of running approximation 1 on the two-horizontal dataset with minimal cost. 37
5.5 Figure depicting a situation when using $\theta = \frac{\pi}{2}$ is not desirable for the two-horizontal dataset. 37
5.6 Result produced by SLS and algorithm 1 on the diagonal dataset. 38
5.7 Result produced by the dynamic programming algorithm on the cross dataset. . . . 38
5.8 Set of segments with lowest associated cost when running algorithm 1 on the cross dataset. 39
5.9 Result of running algorithm 1 on the cross dataset with associated cost 0. 39
5.10 Result produced by the dynamic programming algorithm on the noisy horizontal dataset. 40
5.11 Sample result of running algorithm 1 on the noisy horizontal dataset. 40
5.12 Results of running algorithm 1 on the noisy horizontal dataset with the smallest associated cost. 41

5.13 The left figure shows the moons dataset used for this experiment. The right figure shows the segments produced by running the dynamic programming algorithm on the moons dataset. 41
5.14 Result with lowest cost when running algorithm 1 on the moons dataset. 42
5.15 Segments with the smallest associated cost when running algorithm 1 on the moons dataset. 43
5.16 The left figure shows the blobs dataset used for this experiment. The right figure shows the segments produced by running the dynamic programming algorithm on the blobs dataset. 43
5.17 Result with lowest cost when running algorithm 1 on the blobs dataset. 44
5.18 Segments with lowest associated cost when running algorithm 1 on the blobs dataset. 44
5.19 The left figure shows the circles dataset used for this experiment. The right figure shows the segments produced by running the dynamic programming algorithm on the circles dataset. 45
5.20 Set of segments with lowest cost when running algorithm 1 on the circles data set. 45
5.21 Segments produced by algorithm 1 on the circles dataset, with the best set of paramteres found. 46
5.22 Result produced by the dynamic programming algorithm on the Aniso dataset. . . . 46
5.23 Output of algorithm 1 on the Aniso dataset. 47
5.24 Expected output of algorithm 1 on the Aniso dataset. 47
5.25 Output of algorithm 1 on the housing prices dataset. 48
5.26 Results produced by algorithm 1 on the housing dataset on 10 experiments. 48
5.27 Non-optimal set of parameters applied to algorithm 1 on the housing dataset. 49
5.28 Algorithm 1 result on the cargo ship dataset. 49
5.29 Result of running algorithm 1 on the Amazon stock dataset. 50
5.30 Result produced by algorithm 1 on the Amazon dataset after processing. 51
5.31 Result of running algorithm 1 on the movies dataset. 51
5.32 Result of running algorithm 1 on the movies dataset with $c = \frac{1}{6}$. 52
5.33 Result of running algorithm 1 on the movies dataset without applying the filter. . . 52

Abstract

Data clustering is a prominent topic in Computer Science [9] and Machine Learning [31]. Applications of data clustering include recommendation engines [43], customer segmentation [24], and biological data analysis [36]. Algorithms solving the k-means clustering problems are amongst the most prominent data clustering algorithms [33]. In the k-means problem, we are given a set P of points in a d-dimensional Euclidean space. The goal is to find a set C^*, of k centers in this space, so that the sum of squared distances of points in P to their nearest center in C^* is minimized. However, datasets can have concrete features for which clustering with point centers might not be suitable. Hence, there exists demand for algorithms which have the ability to produce centers that are more general than points [9]. One such generalization would be *the segment k-means clustering* problem, where the goal is to compute a set of segment centers, instead of point centers for which the sum of squared distance is minimized. To the best of our knowledge there exist only two known results for this problem:

- **Time-based segment k-means clustering**: Rosman, Volkov, Feldman, Fisher III, and Rus [34] introduced a version of the problem where points are associated with time intervals. The goal is to approximate the segment k-means problem according to the points' time intervals. This problem is a special case of the general segment k-means problem.
- **Dynamic programming approach:** The dynamic programming algorithm from [25] (Chapter 6.3) solves the general version of the segment k-means problem, using non-streaming data as input. This algorithm returns a set of segments whose cost is within a small error value to the optimal cost. However, the running time of this algorithm is $O(n^3)$, which could be detrimental even for small datasets.

Ideally, we would like to develop an approximation algorithm for the segment k-means problem which improves on the running time of the dynamic programming approach. In this thesis we provide a linear time, bicriteria approximation algorithm for the segment k-means problem. The following results are available in this thesis:

- **Bicriteria approximation algorithm:** We first provide a bicriteria approximation algorithm that solves the segment k-means problem using general data as input. Our algorithm is a randomized approximation algorithm that produces at most $k \log n$ segments. The running time of our algorithm is $O(n \log^2 n + k^2 \log^2 k \log n)$ which is near linear in n.
- (k, ϵ)**-coreset:** We next devise a (k, ϵ)-coreset construction for this problem.
- We benchmark our bicriteria approximation algorithm against the dynamic programming algorithm on a variety of synthetic datasets. We conduct extensive experiments on synthetic and real-world datasets to optimize and tune parameters for the approximation algorithm on every dataset.

Preface

The research in this thesis has been conducted due to my interest in algorithms. I am happy that I was able to develop an algorithm which is able to achieve a better result than a published algorithm on many datasets.

I would like to thank my family and friends for all of the support they have offered during this thesis. I would also like to thank my thesis supervisor Morteza, for helping me develop this algorithm and also for providing me with ideas to improve the results of this thesis.

Chapter 1

Introduction

Clustering is a classical topic in computer science [9], machine learning [31, 4], and data mining [32, 6, 7] with a tremendous number of applications [7], including:

- **Recommendation Engines:** A recommender system [8] collects information about preferences of its users (customers) for a set of items (services or objects). The information is acquired either implicitly (monitoring users' behaviour) or explicitly (collecting users' rating). Modern recommendation engines can also use demographic data (nationality, age, gender) or social media profile (followers, posts) of its users.

  We use clustering algorithms [15, 33] to group similar users based on their preferences, social profile features and behaviours [43]. The users in each cluster are recommended with items based on the common preferences and behaviours of their respective cluster.

  We also use clustering algorithms for recommender systems in the online shopping setting, to recommend products to users based on the items existing in their baskets [38].

- **Stock Market Classification:** Portfolio management and asset selection is one of the most difficult decisions to be made by individual investors and financial institutions [27]. The uncertainty in the levels of fluctuation of a stock, makes it difficult to find a selection of stocks, with high probability of returns, to invest in.

  We can use clustering algorithms [33] to cluster company stocks based on their yearly returns. Financial investors can use the produced clusters to make a decision on which stocks should be added to their portfolio to maximize returns. Some investors might choose a selection of stocks from a cluster containing stocks with consistent returns, while others might also choose stocks from a cluster with higher returns but larger volatility.

One of the most popular clustering problems is the k-means problem [33]. The well-known heuristic for this problem is so-called the k-means algorithm [19], also known as Lloyd's algorithm. Although the algorithm does not have a provable guarantee [37], its simplicity and the quality of clustering usually reported, makes it one of the most used algorithms for clustering problems. Formally, the k-means problem is defined as:

Definition 1 (k-means clustering)**.** *[3] Let P be a set of n points in $\mathbb{R}^2$, and $k \in \mathbb{N}$, a parameter. The task is to find a set S, containing k cluster centers in $\mathbb{R}^2$, such that $\text{COST}(P,S)$ is minimized.*

We define $\text{COST}(P,S)$ to be the sum of squared distances between points in P and their nearest center in S. Formally, $\text{COST}(P,S)$ is defined as:

$$\text{COST}(P,S) = \Sigma_{p\in P}\text{DIST}(p,S)^2$$

where $\text{DIST}(p,S)$ denotes the Euclidean distance between a point $p \in P$ and its nearest point center $s \in S$.

There exist variants of the k-means clustering problem where the set of centers differ from points in the 2-dimensional Euclidean plane. In particular, we consider the case where centers are segments in the plane. In general, for k-means problems using segments as centers, the length of each segment is not bound. In this thesis we introduce an additional parameter, $\ell \in \mathbb{R}^+$. ℓ denotes the maximum length of each segment. We call this problem, *segment k-means clustering*, and define it as follows:

Definition 2 (Segment k-means clustering)**.** *Given a point set $P \subset \mathbb{R}^2$ in a two-dimensional Euclidean space and two parameters $k \in \mathbb{N}$ and $\ell \in \mathbb{R}^+$. The task is to find a set $\mathcal{S}$, containing k segment centers of length at most ℓ, such that $\text{COST}(P,\mathcal{S})$ is minimized.*

The cost function for the segment k-means problem, $\text{COST}(P,\mathcal{S})$, is defined in a similar manner to the cost function for the regular k-means clustering problem. The only difference is the method of measuring the distance between points in P and center segments in $\mathcal{S}$. We define $\text{DIST}(p,\mathcal{S})$ as the shortest distance between a point $p \in P$ and the nearest segment $s \in \mathcal{S}$. To compute the shortest distance between a point and a segment, we distinguish two cases, as seen in figure 1.1.

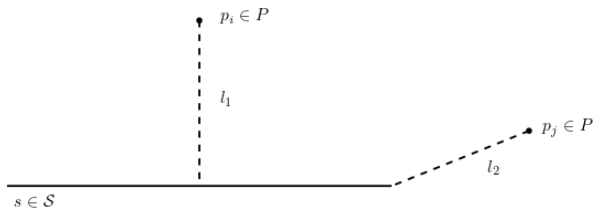

Figure 1.1: Sample points $p_i, p_j \in P$ and their shortest distance to a segment $s \in \mathcal{S}$

If for a point $p_i \in P$, the projection of p_i onto its nearest segment $s \in \mathcal{S}$, is perpendicular, then $\text{DIST}(p_i,\mathcal{S})$ will be denoted by the length of the projection, as shown by line l_1 in figure 1.1. For points of type $p_j \in P$, for which the projection onto $s \in \mathcal{S}$ is not perpendicular, $\text{DIST}(p_j,\mathcal{S})$ is defined as the Euclidean distance between p_j and the closest endpoint of s. $\text{DIST}(p_j,\mathcal{S})$ is denoted as the length of line l_2 in figure 1.1.

Throughout this thesis, the cost between a point set P and a set of centers $\mathcal{S}$, $\text{COST}(P,\mathcal{S})$, will be the sum of squared distances between points in P and the nearest center in $\mathcal{S}$.

1.1 Motivation

In this section we provide the motivation for solving the segment k-means problem instead of other clustering techniques by using two example datasets.

Motivating Example: Synthetic Datasets. In figure 1.2 we depict an instance of the Blobs dataset from the Scikit-learn package [35] in Python. The instance is generated with 500 entries.

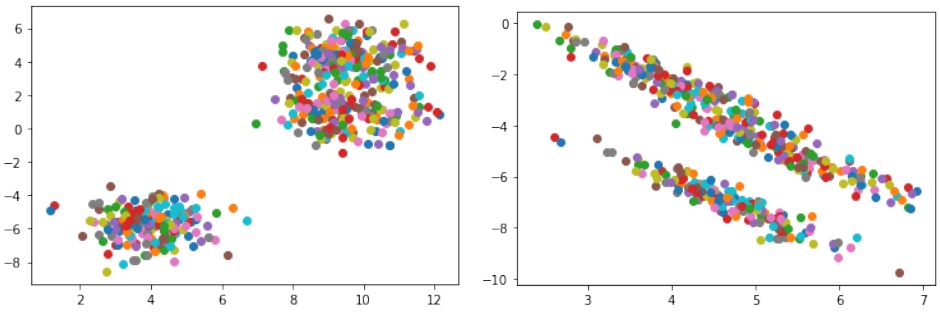

Figure 1.2: Sample blobs and aniso datasets.

On the left figure, the regular Blobs data is shown, while on the right, we present the Blobs dataset on which a transformation matrix T, is applied. The application of T on the Blobs data constructs anisotropic data, better known as Aniso data.

We present these figures to show how different types of data can be summarized by solving different variations of the k-means problem. On the Blobs dataset, due to the circular nature, it is suitable to solve the point center k-means clustering problem, ideally creating a point center for each cluster. On the Aniso data, however, the data is distributed into longer and thinner clusters, thus solving the segment k-means problem would be suitable, creating 2 segments in the process.

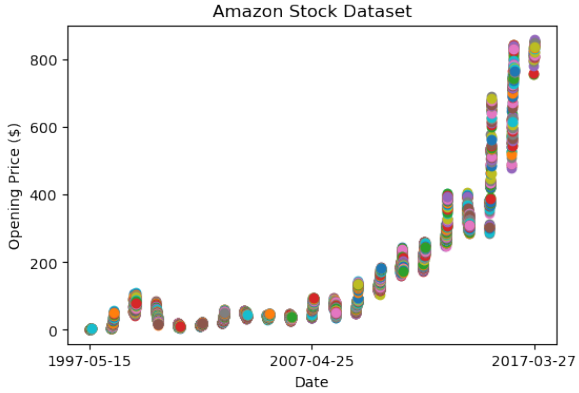

Figure 1.3: Amazon Stock Open Price

Motivating Example: Real-world Dataset. In figure 1.3 we present the open price for the Amazon Stock, **AMZN**, over 5000 days starting in 1997. Linear regression [16], is a popular analysis technique to predict the value of a variable, based on the value of different variables. When applying a linear regression algorithm on the Amazon stock dataset, one would compute the coefficients of a linear equation, to find the line of *best-fit* for the given input of data. A line of *best-fit* for the Amazon stock data is shown on the left of figure 1.4. This line is generated using the linear regression algorithm whose source code is available at [13]. Considering that for majority of data points in figure 1.3, the price of the stock was stable in the range $[0, 200]$, linear regression fails to identify the steep increase in price over the latter years. For a more detailed summary of the data, one could solve the segment k-means problem on the Amazon Stock data, as shown on the right of figure 1.4. With the appropriate value for k, it is possible to produce a set of k-segments

which are able to summarize the portion of the data where the price is stable, as well as having the ability to depict the steep increase in price.

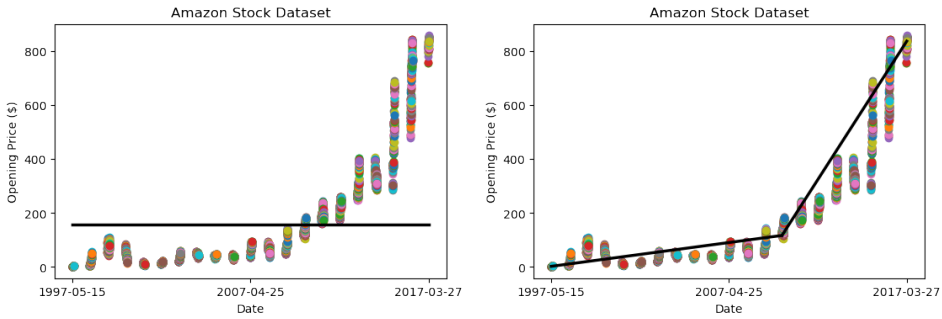

Figure 1.4: The left figure shows the result of applying a linear regression algorithm [13] on the Amazon stock dataset. On the right, we show a possible segment k-means result on the Amazon stock dataset, with $k = 2$.

1.2 Contribution

In this section we discuss the main contributions of this thesis. First, we present the bicriteria approximation algorithm designed to solve the segment k-means problem. Afterwards we present the coreset construction.

1.2.1 Approximation Algorithm

In this thesis we design a bicriteria algorithm (3) that approximates the segment k-means clustering problem. The core idea of our algorithm is inspired by the paper of Har-Peled et. al. [14], which solves the point center k-means problem. The algorithm in paper [14] attempts to cover all of the input points by a set of balls, such that the radius of the largest ball is minimized. The center of each ball is then added to the solution set.

We use the intuition of the algorithm described in [14] to develop an approximation algorithm solving the segment k-means problem. Instead of using a set of balls, we use a set of rectangles having width, $w = 8r$, where r is defined as:

$$r = \frac{\Sigma_{p \in P} \text{DIST}(p, \mathcal{O})}{n}$$

$\mathcal{O}$ is the optimal set of segments for the segment k-means problem of point set P. We attempt to cover the entire point set P with rectangles of minimal size.

The algorithm requires at most $\log n$ iterations to produce a set containing $k \log n$ segments of length 3ℓ. We prove our algorithm has a constant approximation factor with respect to the cost of the optimal solution. The time complexity of the algorithm is $O(n \log^2 n + k^2 \log^2 k \log n)$. Refer to chapter 3 for the detailed construction of the approximation algorithm together with the associated proofs.

In chapter 5 we perform experiments using the approximation algorithm on synthetic and real-world datasets. We compare the results produced by our algorithm against the dynamic programming algorithm in [25]. The results show that our algorithm is able to produce segments

with lower cost than the dynamic programming algorithm for a selection of datasets. In addition, the time required for our algorithm to terminate on larger datasets is significantly quicker than the dynamic programming algorithm.

1.2.2 Coreset Construction

In chapter 4 we present a coreset [1] algorithm which produces a (k,ϵ)—coreset. The aim of a coreset is to compress any dataset P into a smaller dataset S, while maintaining a small approximation error with regards to any set of segments K:

$$|\text{COST}(S,K) - \text{COST}(P,K)| \leq \epsilon \cdot \text{COST}(P,K)$$

In chapter 4 we first provide a formal definition of a coreset. We then proceed by providing the algorithm constructing a (k,ϵ)-coreset, S. For a set of segments B, produced by the approximation algorithm from section 3.1, we prove that $|\text{COST}(S,B) - \text{COST}(P,B)| \leq \epsilon \cdot \text{COST}(P,B)$, where $\epsilon = \frac{\epsilon}{88}$. We also prove the coreset to be of size $O(k \cdot \log^2 n \cdot \frac{\ell}{\epsilon^2 \cdot r})$.

Chapter 2

Related Work

This chapter discusses the background and related work in the field of approximation algorithms for clustering problems. Firstly we discuss algorithms which solve the k-means and median problem in the classical manner, where the reported centers are points in the Euclidean space. Next we present papers which provide algorithms solving the segment k-means problem. Lastly we discuss related work on coreset constructions.

2.1 Point Center k -means and median

k-means and k-median clustering are classic problems in the fields of computer science [9], data mining [32] and machine learning [31]. There are many known algorithms which solve the k-means clustering problem. The k-means problem is believed to have first been discussed in the paper by MacQueen J. [19], in the late 1960s. The most common heuristic for the k-means clustering problem is well-known as the k-means algorithm [19] and uses random k centroid initialization [29].

Practical Algorithms The k-means algorithm using random initialization of centroids consists of the following steps [28]:

1. From a point set P, uniformly at random select k points. Each point becomes a cluster center.
2. Assign each point from P to its closest cluster center.
3. For each cluster, compute a new centroid by calculating the average coordinates of all points in that cluster.
4. Repeat steps 2 and 3 until the algorithm converges to a set of centroids.

A known drawback of the k-means algorithm is the usage of random initialization of k centroids. The k-means++ algorithm [5] is another heuristic solving the k-means clustering problem that improves the initialization of centroids. Rather than using random initialization, the authors propose an initialization technique which samples points with probability based on their squared distance to the closest existing centers. Namely, points which are further away from already existing centers have a higher probability of being sampled than points that are closer. Using the k-means++ initialization, the resulting clustering is proven to have $O(\log k)$ approximation factor [5].

Constant factor approximation algorithm In [18], an algorithm to solve the k-median problem, in $O(n)$ time, having $(1+\epsilon)$-approximation factor with probability $\Omega(\epsilon)$, is provided. The algorithm consists of the following steps:

1. Sample a set of points from P.
2. Apply the k-median algorithm from [26] on the sample, producing a set of centers C_k in the process.
3. The points in P with the largest distance to any center in C_k are re-sampled, and the k-median algorithm from [26] is applied again, hence, producing a new set of centers C'_k.
4. The union of C_k and C'_k is returned as the output.

Coreset k-means Har-Peled et al. [14] provides an approach to approximate both k-means and k-median clustering problems. The paper also provides a coreset construction which computes a weighted set $S \subset P$ of size $O(k\epsilon^{-d}\log n)$. One could compute k-means/median clustering on S instead of P and have a $(1+\epsilon)$-approximation factor. The algorithm is proven to have linear running time using constant k and ϵ values. A distinctive feature of the algorithm in [14] is the ability to maintain a $(1+\epsilon)$ approximation factor for k-means/median clustering problems having a stream of points as input.

2.2 Segment k-means

The segment k-means clustering problem is a generalization of the k-means/median problem. Rather than producing a set of k point centers as output, a set of k segment centers is produced. There exist two heuristics solving the segment k-means clustering problem which are relevant for this thesis. One heuristic is a dynamic programming algorithm, which solves the segment k-means problem with general input. The second algorithm is an approximation algorithm which solves the k-means problem with streaming input.

Dynamic Programming Algorithm For relatively smaller input sets, in the book by Kleinberg and Tardos [25], there exists a dynamic programming algorithm (Chapter 6.3) for solving a version of the segment k-means problem. The algorithm uses the following observation: for point set P, containing points $p_1 \dots p_n$, the last point, p_n, is assigned to a unique segment, s_k. Segment s_k starts at some point p_i, with $i \leq n$. If we can find the point, p_i, where s_k starts, it is possible to add s_k as part of the output. We would then remove points $p_i \dots p_n$ from P, and proceed recursively with $P_i = P \setminus \{p_i \dots p_n\}$. The recursive formula providing the optimal solution is defined as,

$$OPT(n) = e_{i,n} + C + OPT(i-1)$$

where $e_{i,n}$ denotes the cost function of the last segment, and C is a constant. The cost function is defined to be the sum of squared distances between each point in P and the set of segments. The running time of the algorithm is shown to be as high as $O(n^3)$.

Approximation Algorithm The paper by Feldman et al [34] provides a different approach to solve the segment k-means problem. Contrary to the dynamic programming algorithm in [25], the algorithm in [34] takes as input a streaming set of points. The approximation algorithm runs in $O(\log n)$ time and produces $O(k \log n)$ segments.

The intuition used for the construction of the algorithm is that when $k = 1$, we can easily compute a 1-segment using SVD (Singular Value Decomposition). Thus, the idea behind the algorithm is to split the input signal of points into a suitable amount of time intervals, in such a way that at each interval, a 1-segment can be computed. More formally, the algorithm contains the following steps:

1. If $n \leq 2k + 1$, solve the 1-segmentation problem.
2. Partition P into sets $P_1, \ldots, P_{2k}$. Let $p, p' \in P$ be two points such that $p \in P_i$ and $p' \in P_{i+1}$, with $i \leq 2k$. Let t_i and t_{i+1} be the time intervals of P_i and P_{i+1} respectively. It follows that $t_i < t_{i+1}$.
3. For each set, P_i, of the partition, compute the 2-approximation, denoted as g_i, of the 1-segment.
4. Out of the $2k$ segments, select $k + 1$ segments with the lowest, $\text{COST}(P_i, g_i)$.
5. Re-partition the points with a large approximation.

In addition to the approximation algorithm solving the streaming segments k-means problem, the authors provide a coreset construction. The coreset algorithm approximates the points in P by a local representation. The size of the coreset is proven to be $O(\frac{k \log n}{\epsilon^2})$. The authors prove that the size of the coreset is small enough to apply a dynamic programming algorithm. Thus, the coreset construction is proven to have $(1 + \epsilon)$-approximation factor.

2.3 Coreset Construction

In [34] and [14], the authors provide coreset constructions. As defined in [34], a coreset is a subset of the input set. The main attribute of a coreset is the ability to achieve a similar approximation factor when running an approximation algorithm on the coreset instead of running the algorithm on the entire input set. The coreset is also significantly smaller in size [10].

The paper by Dan Feldman [10] provides an insight to the types and use of coresets. Some of the benefits of coresets include optimization, answering queries and boosting heuristics of algorithms. The paper provides multiple construction types of coresets. The most relevant coreset construction type for this thesis is the *Grid* construction. The *Grid* coreset construction consists of partitioning the input into small clusters, and then randomly sampling a point as the representative of the cluster. The representative of each cluster is weighted by the number of points in the cluster. The *Grid* coreset construction is said to be the first type of coreset and was used for covering problems initially [2]. The approximation error of a *Grid* coreset is known to be smaller than other coresets, however, the time and space complexity could be exponential in d due to the potential large number of clusters.

In [12] the authors introduce the notion of a weak (ϵ, k)-coreset. A weak coreset is a weighted subset $S \subset P$ together with a set of centers, T, such that T is a $(1+\epsilon)$-approximation for the optimal set of centers of P. Then for every set of k centers in T, the cost of the centers from T, on coreset S,

is a $(1+\epsilon)$ approximation with regards to the cost of P. The authors claim that for any unweighted set of points P, there exists a weak (ϵ, k)-coreset for the k-means family of problems.

Chapter 3

Segment k-means Approximation Algorithm

In this chapter we develop a bicriteria approximation algorithm for the segment k-means problem. We first define a bicriteria approximation algorithm.

Definition 3 (Bicriteria Approximation Algorithm)**.** *[11] Let $\alpha \in \mathbb{N}, \beta \in \mathbb{R}^+$ be two parameters where $\alpha \geq k$. For a set of points $P \subset \mathbb{R}^2$, an (α, β)-bicriteria approximation algorithm for segment k-means clustering produces a set $\mathscr{S}$, containing α segments of length at most $\ell \in \mathbb{R}^+$. The approximation guarantee of $\mathscr{S}$ is within a factor β from the optimal set of segments for P.*

Next, we state the bicriteria approximation algorithm that we obtain in this section. We will use this algorithm to develop a (k, ϵ)-coreset for the segment k-means problem in the next chapter.

Theorem 4 (Bicriteria Approximation Algorithm)**.** *Let $P \subset \mathbb{R}^2$ be a point set of size n, and let $k \in \mathbb{N}$ and $\ell \in \mathbb{R}^+$ be two parameters. Then, there exists a randomized approximation algorithm that with probability $(1 - \frac{1}{k})^{\log n}$, returns a set $\mathscr{S}$, containing at most $k \log n$ segments of length 3ℓ each, such that:*

$$\textsc{cost}(P, \mathscr{S}) \leq 16 \cdot \textsc{cost}(P, \mathscr{O}) \ ,$$

where $\mathscr{O}$ is an optimal set of k segments of length at most ℓ for point set P. The running time of this algorithm is $O(n \log^2 n + k^2 \log^2 k \log n)$.

As we observe, the running time of this algorithm is near linear in n which is significantly faster than the running time of the dynamic programming from [25]. The running time of the dynamic programming algorithm is shown to be as high as $O(n^3)$.

3.1 Segment k-means Algorithm

We first give the pseudocode of our approximation algorithm in Algorithm 1. Next, we provide an overview of this algorithm where we provide a description of each step in the algorithm. Finally, we prove that this algorithm fulfills the requirements of Theorem 4.

Algorithm 1 Segment k-means

Input: Set of points P, parameters k, ℓ, and $r = \frac{\Sigma_{p\in P}\text{DIST}(p,\mathcal{O})}{n}$
Output: A set $\mathcal{S}$ of segments in accordance with Theorem 4.

Let $S, \mathcal{S} \leftarrow \emptyset$, $0 \le \theta \le \frac{\pi}{2}$, $j \leftarrow \log n, \alpha \in \mathbb{N}$, and $w \leftarrow 8r$
function MAIN
  while $|P| > 1$ **do**
    $S'' \leftarrow$ CANDIDATESEGMENT(P, k, ℓ, θ)
    $\mathcal{S}, P \leftarrow$ FILTERING-CLUSTERING(P, S'', r, j)
  Return $\mathcal{S}$

---

function CANDIDATESEGMENT(P, k, ℓ, θ)
  if $|P| \ge \alpha k \log k$ **then**
    Sample a set S, of $\alpha k \log k$ points from P, uniformly at random and let $S'' \leftarrow \emptyset$
  else $S \leftarrow P$
  Let S' be the set of candidate segments connecting every pair of points in S
  for segment $\overline{pq} \in S'$ **do**
    if DIST$(p, q) > 3\ell$ **then**
      Remove segment $\overline{pq}$ from set S', i.e., $S' \setminus \overline{pq}$
    else
      Stretch $\overline{pq}$ from both end-points equally to obtain new segment $\overline{p'q'}$ with DIST$(p', q') = 3\ell$
      Replace short segment $\overline{pq}$ by stretched segment $\overline{p'q'}$ in S', i.e., $S' = S' \cup \overline{p'q'} \setminus \overline{pq}$
  for $\overline{pq} \in S'$ **do**
    Rotate each segment $\overline{pq}$ around its center, using θ as angle of rotation (See Figure 3.1)
    Add segment $\overline{pq}$ and $\frac{\pi}{\theta}$ copies of segment $\overline{pq}$ to set S''
  Return S''

---

function FILTERING-CLUSTERING(P, S'', r, j)
  Let $S^* \subset S''$ be a set containing segments $\overline{pq} \in S''$ for which $|P \cap R(\overline{pq}, 3\ell, w)| \ge \frac{|P|}{2k}$, where $R(\overline{pq}, a, b)$ is the rectangle of height a and width b centered at segment $\overline{pq}$
  if $S^* \neq \emptyset$ **then**
    $\mathcal{S}_{temp}, P_{temp} \leftarrow$ COMPUTESEGMENTS(P, S^*, w)
    if $|P_{temp}| \le \frac{|P|}{2}$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{S}_{temp}$ and $P \leftarrow P_{temp}$
    else
      for $i \in [2 \dots j]$ **do**
        Let $\tilde{S} \subset \mathcal{S}$ be a set containing segments $\overline{pq} \in \mathcal{S}$, for which $|P \cap R(\overline{pq}, 3\ell, 2^i w)| \ge \frac{|P|}{2k}$
        if $\tilde{S} \neq \emptyset$ **then**
          $\mathcal{S}_{temp}, P_{temp} \leftarrow$ COMPUTESEGMENTS$(P, \tilde{S}, 2^i w)$
          if $|P_{temp}| \le \frac{|P|}{2}$ **then** $P \leftarrow P_{temp}$
  Return $\mathcal{S}$, P

---

function COMPUTESEGMENTS(P, S'', x)
  Let $P_{temp} \leftarrow P$
  while $S'' \neq \emptyset \wedge P_{temp} \neq \emptyset$ **do**
    Let $\overline{p'q'} \in S''$ be the segment for which $C_{\overline{p'q'}} = P_{temp} \cap R(\overline{p'q'}, 3\ell, x)$ is of maximum size
    $\mathcal{S}_{temp} = \mathcal{S}_{temp} \cup \overline{p'q'}$, $S'' = S'' \setminus \overline{p'q'}$ and $P_{temp} = P_{temp} \setminus C_{\overline{p'q'}}$
    for every segment $\overline{pq} \in S''$ **do**
      if $|P_{temp} \cap R(\overline{pq}, 3\ell, x)| < \frac{|P|}{2k}$ **then**
        $S'' = S'' \setminus \overline{pq}$
  Return $\mathcal{S}_{temp}, P_{temp}$

High-Level Algorithm Overview. Our approximation algorithm, provided in Algorithm 1, contains four subroutines. The main subroutine is denoted by MAIN. The other three subroutines are called either by MAIN or by the other subroutines. These three subroutines are CANDIDATESEGMENT, COMPUTESEGMENTS and FILTERING-CLUSTERING. In the next paragraph, we provide a high-level overview of the algorithm. Afterwards, we explain in detail each subroutine of the algorithm.

Our algorithm consists of at most $t = \log n$ iterations $\mathscr{E}_1, \cdots, \mathscr{E}_i, \cdots, \mathscr{E}_t$. During an arbitrary iteration $\mathscr{E}_i$, we obtain up to k segments in a way that the union of these segments can cover (or cluster) at least half of the remaining (so-called *uncovered*) points in P. Once we find such a set of k segments, we remove the points that are covered (or clustered) using these segments from P, and add these segments to our solution set $\mathscr{S}$. If more than $O(k \log k)$ points are left to be clustered in P, we continue with the next iteration. If we are left with fewer than $O(k \log k)$ points, then the sampling step in CANDIDATESEGMENT cannot be applied. Thus, to generate a set of segments for the remaining points in P, we skip the sampling step and move directly to the step which generates the set of segments S', using the pairs of points in P. The algorithm then proceeds with the remaining steps.

Let us fix an iteration, $\mathscr{E}_i$. During this iteration we sample $O(k \log k)$ points uniformly at random from P. We enumerate every pair of sampled points to find a set of candidate segments. All computed segments whose length is greater than 3ℓ are removed from the set of candidate segments. The remaining segments are stretched equally from both endpoints to reach a segment of length 3ℓ. Next, using these candidate segments we have two options:

- **Detect new clusters:** We can find a subset containing at most k segments from the candidate set of segments, such that within a distance, $w = 4r$, of each of these segments, there are at least half of the uncovered points in P. We define r to be average distance between the points in P and the segments in the optimal solution $\mathscr{O}$.

  However, for many point sets the optimal set of segments $\mathscr{O}$, is not known, meaning we will not have access to the value for r. Thus, we will have to estimate the value of r. To estimate the value of r, suppose that a point set P has bounded spread ratio,

  $$\rho = \frac{\max_{p,q \in P} \text{DIST}(p,q)}{\min_{p,q \in P} \text{DIST}(p,q)}$$

  Then, we know that $\sum_{p \in P} \text{DIST}(P, \mathscr{O}) \leq n \cdot \rho$. So, we take $x = \lceil \log_{1+\epsilon}(n \cdot \rho)) \rceil$ guesses, $(1+\epsilon)^1, \cdots, (1+\epsilon)^i, \cdots, (1+\epsilon)^x$, for the value of $\text{DIST}(P, \mathscr{O})$. Trivially, one of these guesses will have $(1+\epsilon)$-approximation of $\text{DIST}(P, \mathscr{O})$. For each guess, we let $r_i = \frac{(1+\epsilon)^i}{n}$. We then select the smallest value of r for which the algorithm is able to cluster all of the points using at most $k \log n$ segments.

- **Extend existing clusters:** If the first case is not possible, we then choose at most k segments from solution set $\mathscr{S}$ so that within distance $2^i w$ of these segments (for the minimal $i \in [\log n]$), there are at least half of the remaining points in P.

In either case, the points that are now covered by the k segments are removed from P. If the points are covered by detecting new clusters, then those segments are added to the solution set $\mathscr{S}$. Otherwise, no additional segments are added to $\mathscr{S}$. At this point, the current iteration terminates, and we proceed to the next iteration.

Detailed Algorithm Overview. So far we have provided a high-level overview of the algorithm. In this paragraph, we explain the steps of the algorithm in detail.

1. MAIN: The purpose of the MAIN function is to invoke other subroutines to produce a solution set $\mathscr{S}$, of segments. We maintain a while loop until $|P| <= 1$. At each iteration we first make a call to the CANDIDATESEGMENT function, creating a set of candidate segments S''. We call function FILTERING-CLUSTERING with input set S'', to find a subset $\mathscr{S}_{temp}$ of S''. If segments in $\mathscr{S}_{temp}$ cluster at least $\frac{|P|}{2}$ points from P, we add $\mathscr{S}_{temp}$ to the solution set $\mathscr{S}$. Let P_{temp} be the points remaining after processing the segments in $\mathscr{S}_{temp}$. We continue at the next iteration with $P = P_{temp}$.

2. CANDIDATESEGMENT: - The CANDIDATESEGMENT subroutine is used to initialize a candidate set of segments. The first step in the subroutine is to check the number of points remaining in P. If $|P| \geq \alpha k \log k$, then we sample a set of points $S \subset P$, of size $\alpha k \log k$, uniformly at random. Otherwise, we let $S = P$. Let S' be a set of segments connecting each pair of points in S. Formally, let $p, q \in S$, be two arbitrary points in S. We add all segments, $\overline{pq}$ to S'.

   According to theorem 4, the approximation algorithm must produce segments of length at most 3ℓ. To satisfy the length condition of each segment $\overline{pq} \in S'$, we do the following:

   - **Case 1:** $\textsc{dist}(p,q) \leq 3\ell$**:** We stretch segment $\overline{pq}$ equally from both endpoints until a new segment $\overline{p'q'}$ is created for which $\textsc{dist}(p',q') = 3\ell$. Observe that points, p, q, p', q' are collinear. We remove $\overline{pq}$ from S' and add $\overline{p'q'}$ to S'.
   - **Case 2:** $\textsc{dist}(p,q) > 3\ell$**:** Segment $\overline{pq}$ is removed from S'.

   The next step in the subroutine is to create copies of the remaining segments in S' having different orientation. We introduce a new angular constant θ, where $0 \leq \theta \leq \frac{\pi}{2}$. We rotate each segment $\overline{pq} \in S'$ around its center using θ as the angle of rotation. For any value of θ we produce $\frac{\pi}{\theta}$ copies of segment $\overline{pq}$. Let S'' be the set containing each segment $\overline{pq}$ from S', together with its rotated copies.

   In figure 3.1 we present a segment $s = \overline{pq} \in S'$, together with its copies. We use $\theta = \frac{\pi}{4}$, in the process creating $\frac{\pi}{\theta} = 4$ segments including s. Neighboring segments, s and s_θ, have angular distance equal to θ.

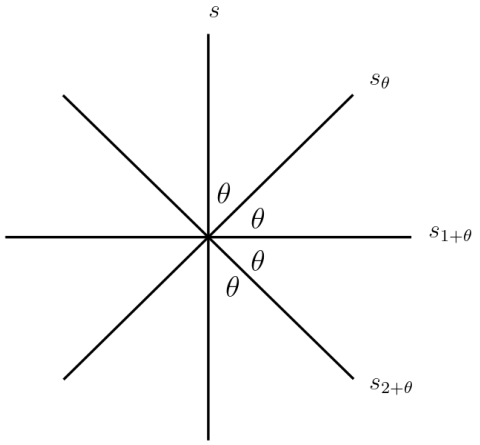

Figure 3.1: Figure showing all of the copies of a segment s with $\theta = \frac{\pi}{4}$

3. FILTERING-CLUSTERING : The FILTERING-CLUSTERING function takes as input the set of segments S'' produced by CANDIDATESEGMENT. The first step in the subroutine is to find a set $S^* \subset S''$, which contains segments $\overline{pq} \in S''$ such that $|P \cap R(\overline{pq}, 3\ell, w)| \geq \frac{|P|}{2k}$. The notation $R(\overline{pq}, a, b)$, denotes the boundary of a rectangle, centered at segment $\overline{pq}$, having length a, and width, b. A standard rectangle centered at segment $\overline{pq}$, would be of type, $R(\overline{pq}, 3\ell, w)$, with $w = 8r$. Let r denote the average distance between points in P and the optimal set of segments $\mathcal{O}$, i.e $r = \frac{\sum_{p \in P} \text{DIST}(p, \mathcal{O})}{n}$. Informally, around each segment $\overline{pq} \in S''$ we place a rectangle centered at $\overline{pq}$ having width, $w = 8r$. If there exist $\frac{|P|}{2k}$ points inside of the rectangle, then $\overline{pq}$ is added to S^*.

   We apply COMPUTESEGMENTS on S^*, producing sets, $\mathscr{S}_{temp}$ and P_{temp} in the process. We have two cases:

   - If $|P_{temp}| \leq \frac{|P|}{2}$, then we add $\mathscr{S}_{temp}$ to $\mathscr{S}$, and $P = P_{temp}$. We then proceed with the next iteration of the while loop in the MAIN function.
   - If $|P_{temp}| > \frac{|P|}{2}$, it follows that with segments from S^* we have not been able to find up to k segments covering at least $\frac{|P|}{2}$ uncovered points. We then create larger rectangles for segments already existing in $\mathscr{S}$ such that at least $\frac{|P|}{2}$ points are covered by these rectangles. Formally, we introduce a new variable i. For $i \in [2, \ldots, \log n]$, around each segment $\overline{pq} \in \mathscr{S}$ we create a rectangle having width $2^i w$. Let $\tilde{S} \subset \mathscr{S}$ be a set containing segments $\overline{pq} \in \mathscr{S}$, for which $|P \cap R(\overline{pq}, 3\ell, 2^i w)| \geq \frac{|P|}{2k}$. We apply COMPUTESEGMENTS with $\tilde{S}$, once again, producing new sets, $\mathscr{S}_{temp}$ and P_{temp}. Once we find the value of i which returns the set P_{temp}, with $|P_{temp}| \leq \frac{|P|}{2}$, we let $P = P_{temp}$ and proceed with the next iteration in MAIN. Notice that the segments in $\mathscr{S}_{temp}$, produced by applying COMPUTESEGMENTS on $\tilde{S}$ will produce segments which are already in $\mathscr{S}$, thus no additional segments are added to $\mathscr{S}$.

4. COMPUTESEGMENTS : The COMPUTESEGMENTS subroutine is used to filter a set of segments. The function takes as input a set of segments S^*, which contains segments $\overline{pq} \in S^*$, for which $|P \cap R(\overline{pq}, 3\ell, w)| \geq \frac{|P|}{2k}$. The first step in the function is to let $P_{temp} = P$, be a copy of the set of points in P. In S^* we are interested in finding the segment, $\overline{p'q'}$, for which $C_{\overline{p'q'}} = P_{temp} \cap R(\overline{pq}, 3\ell, w)$ is of maximum size. We add the segment $\overline{p'q'}$ to a set of segments, $\mathscr{S}_{temp}$, and remove it from S^*. Formally $\mathscr{S}_{temp} = \mathscr{S}_{temp} \cup \overline{p'q'}$ and $S^* = S^* \setminus \overline{p'q'}$. The points in $C_{\overline{p'q'}}$ are removed from P_{temp}, i.e $P_{temp} = P_{temp} \setminus C_{\overline{p'q'}}$. Notice that when removing $C_{\overline{p'q'}}$ from P_{temp}, there exists the possibility that for some segments, $\overline{pq} \in S^*$, $|P_{temp} \cap R(\overline{pq}, 3\ell, w)| < \frac{|P|}{2k}$. Since we are only interested in segments for which $|P_{temp} \cap R(\overline{pq}, 3\ell, w)| \geq \frac{|P|}{2k}$, such segments are removed from S^*, i.e $S^* = S^* \setminus \overline{pq}$. We continue searching for segments $\overline{p'q'} \in S^*$ for which $C_{\overline{p'q'}} = P_{temp} \cap R(\overline{pq}, 3\ell, w)$ is of maximum size while $S^* \neq \emptyset$ and $P_{temp} \neq \emptyset$. At the end of the while loop we return the segments produced in $\mathscr{S}_{temp}$, together with the points remaining in P_{temp} to be used in the FILTERING-CLUSTERING function.

3.1.1 Algorithm Correctness

At each iteration of algorithm 1 we wish to cluster an additional $\frac{|P|}{2}$ points from P. There are two options for clustering $\frac{|P|}{2}$ points at any iteration. Either we are able to sample $\alpha k \log k$ points from

P, and produce k segments using the sampled points such that each segment clusters $\frac{|P|}{2k}$ points. Otherwise, we increase the width of each segment in $\mathscr{S}$ to $2^i w$, for $i \in [2, \ldots, \log n]$, until $\frac{|P|}{2}$ points are covered. In this section we first compute the probability that for any sample of size $\alpha k \log k$, we are able to produce a set of segments which cover $\frac{|P|}{2}$ points. We also prove that if we are not able to produce such a set of segments, then there exists a value for i such that if we increase the width of each rectangle associated to a segment in $\mathscr{S}$ we are able to cluster $\frac{|P|}{2}$ points.

Theorem 5. *Let $\mathscr{O} = \{L_1^*, \cdots, L_k^*\}$ be the optimal set of segments for a point set P. Suppose for each segment $L_j^* \in \mathscr{O}$ there is a corresponding cluster C_j^*. Let $S = \{s_1, \cdots, s_t\}$ be a set of sampled points from P, where $t = \alpha k \log k$. The probability that S contains 2 points from each cluster corresponding to a segment in $\mathscr{O}$ is $1 - \frac{1}{k}$.*

Proof. Let $C_j'^*$ be the points in cluster C_j^* whose distance from their closest segment $L_j^* \in \mathscr{O}$ is at most $2r$, where $r = \frac{\sum_{p \in P} \text{DIST}(p, \mathscr{O})}{n}$. Using the Markov inequality [40], the probability that points have distance $2r$ from L_j^* is $\frac{1}{2}$. Thus, the following property holds regarding the number of points in $C_j'^*$:

$$|C_j'^*| \geq \frac{|C_j^*|}{2}$$

Let C_j^* be a cluster associated to a segment L_j^* in the optimal solution $\mathscr{O}$ for which we know $|C_j^*| \geq \frac{|P|}{2k}$. Using the property defined previously, it follows that $|C_j'^*| \geq \frac{|P|}{4k}$.

Now we study the probability that an arbitrary point $s_i \in S$ is not in $C_j'^*$. The probability of a point $s_i \in S$ being sampled uniformly at random from a cluster of size $\frac{P}{4k}$ is $\mathbb{P}\Big[s_i \in C_j'^*\Big] \leq \frac{\frac{|P|}{4k}}{|P|} \leq \frac{1}{4k}$. Thus, for the probability that s_i is not in $C_j'^*$ we have, $\mathbb{P}\Big[s_i \notin C_j'^*\Big] \leq (1 - \frac{1}{4k})$.

We split the sampled set S into two subsets S_1 and S_2 having equal size. We study the probability that $S_1 \cap C_j'^* = \emptyset$ and $S_2 \cap C_j'^* = \emptyset$. Informally, we study the probability that there do not exist two points in cluster $C_j'^*$ that are also in S. We first compute the probability that $S_1 \cap C_j'^* = \emptyset$, using $\alpha \geq 16$ and the Taylor series [41]:

$$\mathbb{P}\Big[|S_1 \cap C_j'^*| = 0\Big] \leq (1 - \frac{1}{4k})^{\frac{\alpha}{2} k \log k} \leq e^{\frac{\alpha}{2} k \log k \cdot \frac{1}{4k}} \leq e^{\frac{\alpha}{8} \log k} \leq \frac{1}{k^{\alpha/8}} = \frac{1}{k^2} \ ,$$

The probability for $S_2 \cap C_j'^* = \emptyset$ is also $\frac{1}{k^2}$ and can be computed in a similar manner to the probability of $S_1 \cap C_j'^* = \emptyset$.

So far, we have computed the probability that there do not exist two points in the sample set S from a cluster $C_j'^*$. However, there could exist up to k clusters $C_1^*, \cdots, C_k^*$, associated to segments in $\mathscr{O}$ for which $|C_j^*| \geq \frac{|P|}{2k}$. To find the probability that two points from each cluster $C_1^*, \cdots, C_k^*$ are not located in S, we apply the union bound [30]:

$$\sum_{j=1}^{k} \mathbb{P}\Big[|S \cap C_j^*| < 2\Big] \leq k \cdot \frac{1}{k^2} \leq \frac{1}{k} \ .$$

Thus, with probability at least $1 - \frac{1}{k}$, for each cluster C_j^* which contains at least $\frac{|P|}{2k}$ points, S contains at least two points from C_j^*. Suppose that with any two points from a cluster C_j^* containing $\frac{|P|}{k}$ points, we are able to produce a segment that covers $\frac{|P|}{2k}$ points (Proven in section

3.1.3). Then, the probability that at any iteration we are able to produce at most k segments, each covering $\frac{|P|}{2k}$ points, is $1-\frac{1}{k}$.

□

Recall the FILTERING-CLUSTERING function in algorithm 1. If we are unable to produce a set of at most k segments each clustering at least $\frac{|P|}{2k}$ points, we propose a solution using segments already existing in the solution set, $\mathscr{S}$. Namely, we introduce a variable $i \in [0,\dots,\log n]$, which is used to increase the width of each rectangle of a segment in $\mathscr{S}$ until $\frac{|P|}{2}$ points are covered. We proceed to prove that if at any iteration we are unable to find a set of at most k suitable segments, there exists a value for i which can be used to cover $\frac{|P|}{2}$ points.

Lemma 6. *Suppose $j \leq \log n$ is an iteration of the* MAIN *function for which* COMPUTESEGMENTS *is unable to produce a set of segments which cover $\frac{|P|}{2}$ points. We increase the width the rectangle associated to each segment $\overline{pq} \in \mathscr{S}$ from $R(\overline{pq}, 3\ell, w)$ to $R(\overline{pq}, 3\ell, 2^i w)$ until we are able to cluster an additional $\frac{|P|}{2}$ points. Then, there exists a value $i \in \mathbb{N}$, such that if we increase the width of the rectangles associated to segments in $\mathscr{S}$, we are able to cover $\frac{|P|}{2}$ points.*

Proof. Let $i = \log n$. The maximum width of a rectangle is given by, $w = 2^i \times 8r = 2^{\log n} \times 8r = n \times 8r$. Recall that, $r = \frac{\Sigma_{p\in P}\text{DIST}(p,\mathscr{O})}{n}$. It follows that the largest possible width of a rectangle is,

$$w = 8n \times \frac{\Sigma_{p\in P}\text{DIST}(p,\mathscr{O})}{n} = 8 \times \Sigma_{p\in P}\text{DIST}(p,\mathscr{O})$$

Thus, since the width of a rectangle can be up to 8 times greater than the total distance of all points in P to a segment in $\mathscr{O}$, with high probability, we are able to cover $\frac{|P|}{2}$ points at any iteration. □

3.1.2 Representation Size Analysis

In this section we estimate the number of segments produced by the approximation algorithm 1.

Lemma 7. *Approximation algorithm 1 returns a set of segments, $\mathscr{S}$, containing at most $k \log n$ segments.*

Proof. We prove algorithm 1 produces a set of segments of size, $|\mathscr{S}| \leq k \log n$, using the algorithm description. Recall that at each iteration of the algorithm, we are interested in producing a set of segments which covers (clusters) $\frac{|P|}{2}$ points, while each segment covers $\frac{|P|}{2k}$ points. To match the constraints, it must follow that at each iteration, up to k segments are produced. There exists the possibility that at each iteration, we do not produce any segments, and instead cluster $\frac{|P|}{2}$ points by increasing the width of the rectangle of each segment in $\mathscr{S}$. In those situations, no additional segments are produced. However, we have shown that with probability $1-\frac{1}{k}$, at each iteration we are able to produce k segments which cluster $\frac{|P|}{2}$ points. Since after each iteration, at least $\frac{|P|}{2}$ additional points are clustered, it follows that after at most $\log n$ iterations, $P = \emptyset$. Thus, if at each iteration k segments are produced with high probability, and the algorithm consists of at most $\log n$ iterations, the number of segments in $\mathscr{S}$ is given by,

$$|\mathscr{S}| \leq k \log n$$

□

3.1.3 Approximation Factor

In this section we estimate the cost of the segments in $\mathscr{S}$ with regards to the optimal segments in $\mathscr{O}$. Let $\text{COST}(P, \mathscr{O})$ denote the optimal cost and let $\text{COST}(P, \mathscr{S})$ be the cost of the segments produced by algorithm 1 on a point set P. We prove in this section that $\text{COST}(P, \mathscr{S}) \leq 16 \times \text{COST}(P, \mathscr{O})$.

Theorem 8. *The output of algorithm 1 has a* 16*-approximation guarantee.*

Proof. Recall the FILTERING-CLUSTERING subroutine in algorithm 1. At each iteration of the algorithm, FILTERING-CLUSTERING is invoked to cluster at least $\frac{|P|}{2}$ points either by using a set of at most k new segments, or by extending the width of the rectangles associated to segments existing in $\mathscr{S}$. We proceed to estimate the approximation guarantee of $\mathscr{S}$ by computing the maximum distance between a point and the segment in $\mathscr{S}$ that it is clustered by.

To this end, we partition the set P into sets $P_0, \dots P_i$, with $i \leq \log n$ as follows: Let P_i be the set which contains points from P that are clustered by a segment in $\mathscr{S}$ with a rectangle of width, $w = 2^i \cdot 8r$. Using the sets $P_0, \dots, P_i$, we can define the approximation factor as:

$$\text{COST}(P, \mathscr{S}) \leq \Sigma_{i=1}^{i} |P_i| \cdot (2^i \times 4r)^2$$

where $|P_i|$ denotes the number of points inside the set P_i. Note that the maximum distance between a point and a segment in $\mathscr{S}$ is $2^i \times 4r$ instead of $2^i \times 8r$. This is true because each segment is located at the center of its associated rectangle.

As a global argument, the sets $P_0, \dots P_i$, in total, contain n points. In lemma 6, we have proven that with rectangles of width $2^i \cdot 8r$, where $i = \log n$, we are able to cover all points in P, with high probability. Thus, in the worst case, the largest possible value for $\text{COST}(P, \mathscr{S})$ is given when the set P_i, with $i = \log n$, contains n points:

$$\begin{aligned} \text{COST}(P, \mathscr{S}) &\leq n \cdot (2^{\log n} \times 4r)^2 = 16r^2 \times n^3 \\ &\leq 16 \times n^3 \times \frac{\Sigma_{p \in P} \text{DIST}(p, \mathscr{O})^2}{n^2} \end{aligned}$$

Realize that $\Sigma_{p \in P} \text{DIST}(p, \mathscr{O})^2 = \text{COST}(P, \mathscr{O})$, thus,

$$\text{COST}(P, \mathscr{S}) \leq 16 \times n^3 \times \frac{\text{COST}(P, \mathscr{O})}{n^2} \leq 16n \times \text{COST}(P, \mathscr{O})$$

As proven so far, algorithm 1 produces a result having an approximation guarantee which is linear in n with respect to the optimal result, $\mathscr{O}$. However, so far we assume that all of the points are inside the set P_i, with $i = \log n$. We proceed to prove that with high probability, each segment in $\mathscr{S}$ clusters $\frac{|P|}{2k}$ points within a distance of $4r$.

Let $\mathscr{O} = \{L_1^*, \dots, L_k^*\}$ be the set of optimal segments. We associate each optimal segment to a cluster of points in P, $C = \{C_1^*, \dots, C_k^*\}$. Suppose the number of points in each cluster is $\frac{|P|}{k}$. Let $S = \{s_1, \dots, s_t\}$ be a set of sampled points from P, with $t = \alpha k \log k$. We have shown in theorem 5 that with probability $1 - \frac{1}{k}$, there exist two points from each optimal cluster in S. If we are able to sample two points from each cluster, it follows that we are able to create a segment $\overline{pq}$ for each optimal cluster.

We fix a cluster C_j^*, which is associated to an optimal segment L_j^*. With probability, $1 - \frac{1}{k^2}$, the set S contains at least two points from C_j^*. Using the two sampled points, we create a segment $\overline{pq}$, along with copies of $\overline{pq}$ being rotated by a small angular constant θ. We proceed to prove that if

there exists a segment $\overline{p'q'}$ which is a copy of $\overline{pq}$ having different rotation, for which the angular distance between $\overline{p'q'}$ and L_j^* is 0, then we are able to cluster $\frac{|P|}{2k}$ points from C_j^* within a distance of $4r$.

The average distance between a point in P and a optimal segment in $\mathcal{O}$ is defined by r. Using Markov's inequality, the probability of a point being at distance greater than $2r$ from its optimal segment is $\frac{1}{2}$. Thus, with high probability, at least one half of the points in cluster C_j^* are within distance $2r$ from L_j^*. Since C_j^* contains $\frac{|P|}{k}$ points, it follows that there are at least $\frac{|P|}{2k}$ points within distance $2r$ to L_j^*. The points having distance at most $2r$ to L_j^* can be modeled as being inside of rectangle $R(L_j^*, \ell, 4r)$ as shown in figure 3.2.

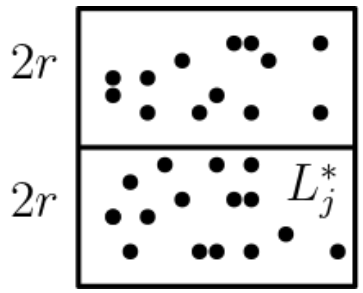

Figure 3.2: Segment L_j^* along with points inside $R(L_j^*, \ell, 4r)$

In the worst case, suppose points $p, q \in S$ from cluster C_j^* are at one of the corners inside $R(L_j^*, \ell, 4r)$. Then we create a segment $\overline{pq}$, which we stretch to have length 3ℓ. Then we rotate $\overline{pq}$ by an angle θ, in such a way that we construct a segment $\overline{p'q'}$, whose angular distance to L_j^* is 0. Then, centered at $\overline{p'q'}$ we place the boundary of a rectangle $R(\overline{p'q'}, 3\ell, 8r)$, as shown in figure 3.3.

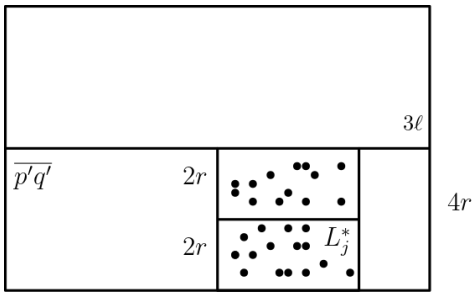

Figure 3.3: Rectangle $R(\overline{p'q'}, 3\ell, 8r)$ showing to entirely cover $R(L_j^*, \ell, 4r)$

As shown in the figure, if the angular distance between $\overline{p'q'}$, and L_j^* is 0, we are able to sample two points in any corner of $R(L_j^*, \ell, 4r)$ and still be able to entirely cover the points in cluster C_j^* which have distance at most $2r$ from L_j^*, with $R(\overline{p'q'}, 3\ell, 8r)$. Notice that if the points sampled are closer to L_j^*, we are able to use a larger value for constant θ and still be able to entirely cover $R(L_j^*, \ell, 4r)$. A similar segment can be constructed for each optimal cluster if at least two points are sampled.

Thus, since for each optimal cluster we are able to produce a segment that clusters at least $\frac{|P|}{2k}$ points, and there are k clusters, it follows that at any iteration we have the ability to cluster $\frac{|P|}{2}$

points within a distance of $4r$. This result matches the conditions described in the FILTERING-CLUSTERING subroutine, meaning that the segments produced for each optimal cluster will be added to $\mathscr{S}$. We use this result to estimate the approximation factor. If at any iteration of the approximation algorithm we are able to cluster $\frac{|P|}{2}$ points within a distance $4r$, the approximation factor becomes:

$$\begin{aligned}
\text{COST}(P, \mathscr{S}) &\leq \Sigma_{p \in P} \text{DIST}(p, \mathscr{S})^2 \\
&\leq n \times (4r)^2 \\
&\leq n \times 16r^2 \\
&\leq n \times 16 \frac{\Sigma_{p \in P} \text{DIST}(p, \mathscr{O})^2}{n^2} \\
&\leq n \times 16 \frac{\text{COST}(P, \mathscr{O})}{n^2} \\
&\leq \frac{16}{n} \times \text{COST}(P, \mathscr{O}) \\
&\leq 16 \times \text{COST}(P, \mathscr{O})
\end{aligned}$$

□

The proof regarding the approximation guarantee, so far, uses a strong assumption that for each optimal segment $L_j^* \in \mathscr{O}$ we are able to create a segment $\overline{pq} \in \mathscr{S}$ whose angular distance to L_j^* is 0. We proceed to compute the approximation guarantee of the algorithm using a weaker assumption.

Consider an optimal segment L_j^* and a segment $\overline{p'q'} \in \mathscr{S}$ generated using two points sampled from the cluster of L_j^*, as shown in figure 3.4.

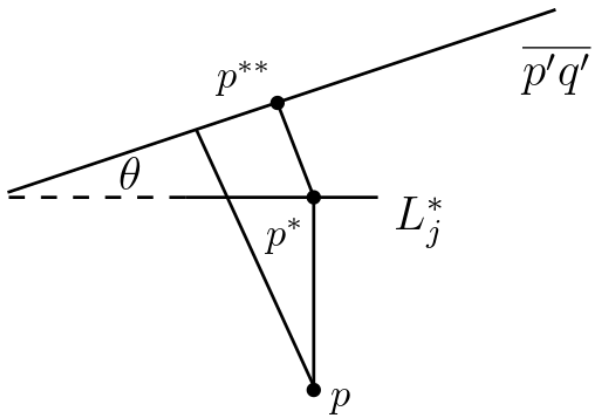

Figure 3.4: Sample cluster around optimal segment L_j^*, together with segment $\overline{p'q'}$.

We can bound the distance between point p and the segment $\overline{p'q'} \in \mathscr{S}$ as follows:

$$\text{DIST}(p, \overline{p'q'}) \leq \text{DIST}(p, p^*) + \text{DIST}(p^*, p^{**})$$

Notice that the distance $\text{DIST}(p^*, p^{**})$, is equivalent to computing $|L_j^*| \cdot \sin\theta$. Since the length of each optimal segment is at most ℓ, it follows that $\text{DIST}(p^*, p^{**}) \leq \ell \cdot \sin\theta$. We now attempt to find the lower bound for the value of θ. Suppose $\frac{\ell}{c} \leq r \leq \ell$, for some constant $c \leq O(\log n)$.

Centered at $\overline{p'q'}$ we place a rectangle $R(\overline{p'q'}, 3\ell, 8r)$. We can use the dimensions of this rectangle to estimate the value of $\sin\theta$. It follows that $\sin\theta = \frac{\epsilon r}{\ell}$. Since we assume that $\frac{\ell}{c} \leq r \leq \ell$, the following inequality holds:

$$\frac{\epsilon\frac{\ell}{c}}{\ell} \leq \sin\theta \leq \frac{\epsilon \cdot \ell}{\ell} = \frac{\epsilon}{c} \leq \sin\theta \leq \epsilon$$

Thus, the lower bound for the value of θ is:

$$\theta \geq sin^{-1}\frac{\epsilon}{c}$$

Using $\theta \geq sin^{-1}\frac{\epsilon}{c}$, we can bound $\text{DIST}(p, \overline{p'q'})$ for each optimal cluster as follows:

$$\begin{aligned}
\sum_{p\in P} \text{DIST}(p, \overline{p'q'}) &\leq \sum_{p\in P} \text{DIST}(p, p^*) + \text{DIST}(p^*, p^{**}) \\
&\leq \sum_{p\in P} \text{DIST}(p, p^*) + \ell \cdot \sin\theta \\
&\leq \sum_{p\in P} \text{DIST}(p, p^*) + \ell \cdot \frac{\epsilon r}{\ell} \\
&\leq \sum_{p\in P} \text{DIST}(p, p^*) + \epsilon r
\end{aligned}$$

Recall that $r = \frac{\sum_{p\in P} \text{DIST}(p,\mathcal{O})}{n}$. $\text{DIST}(p, p^*)$ denotes the distance between a point p and the optimal segment, thus $\sum_{p\in P} \text{DIST}(p, p^*) = \sum_{p\in P} \text{DIST}(p, \mathcal{O}) = n \cdot r$. Then,

$$\begin{aligned}
\sum_{p\in P} \text{DIST}(p, \overline{p'q'}) &\leq n \cdot r + \epsilon \cdot r \\
&\leq r \cdot (n + \epsilon)
\end{aligned}$$

Notice that $\sum_{p\in P} \text{DIST}(p, \overline{p'q'})$ as shown in figure 3.4 is equivalent to computing the distance between each point in P to its closest segment in $\mathcal{S}$, i.e $\text{DIST}(P, \mathcal{S})$. Thus, $\text{COST}(P, \mathcal{S})$ has the following lower bound:

$$\begin{aligned}
\text{COST}(P, \mathcal{S}) &\leq (r \cdot (n + \epsilon))^2 \\
&\leq \frac{\sum_{p\in P} \text{DIST}(p, \mathcal{O})^2}{n^2} \cdot (n + \epsilon)^2 \\
&\leq \frac{\text{COST}(P, \mathcal{O})}{n^2} \cdot (n + \epsilon)^2 \\
&\leq \text{COST}(P, \mathcal{O}) + \frac{2\epsilon}{n} \cdot \text{COST}(P, \mathcal{O})
\end{aligned}$$

Thus, when using $\theta \geq sin^{-1}\frac{\epsilon}{c}$, it follows that:

$$\text{COST}(P, \mathcal{S}) \leq \frac{\epsilon}{n} \cdot \text{COST}(P, \mathcal{O})$$

3.1.4 Performance Analysis

In the section we asses the running time of algorithm 1. We do so by estimating the performance required for each subroutine in the algorithm.

Lemma 9. *Producing a candidate set of segments using the* CANDIDATESEGMENT *function has* $O(k^2 \log^2 k)$ *time complexity.*

Proof. Taking a sample of size $\alpha k \log k$ uniformly at random from a point set P has $O(\alpha k \log k)$ complexity. The sample is stored in a set S. To construct all possible segments using pairs of points in S, we must use nested iteration over S. The time complexity of producing the set of segments S', is $O(k^2 \log^2 k)$. All other steps inCANDIDATESEGMENT are performed on S', and can be achieved in $O(k^2 \log^2 k)$ time. Hence, the time complexity of the CANDIDATESEGMENT subroutine is $O(k^2 \log^2 k)$. □

Lemma 10. *Given a set of segments* S'' *produced by* CANDIDATESEGMENT*, the time complexity of applying* FILTERING-CLUSTERING *on* S'' *is* $O(n \log n + nk)$.

Proof. The first step in FILTERING-CLUSTERING is to find a set $S^* \subset S''$ containing segments $\overline{pq} \in S''$ such that $|P \cap R(\overline{pq}, 3\ell, w)| \geq \frac{|P|}{2k}$. Considering S'' is produced using CANDIDATESEGMENT, the size of S'' is $O(k^2 \log^2 k)$, meaning the time complexity of constructing segment set S^* is $O(k^2 \log^2 k)$.

On S^* we apply the COMPUTESEGMENTS function. In COMPUTESEGMENTS we iterate through S^* to find the segment $\overline{p'q'} \in S^*$ for which $|P \cap R(\overline{p'q'}, 3\ell, w)|$ is of maximum size. Once we find $\overline{p'q'}$, we remove all points in $P \cap R(\overline{p'q'}, 3\ell, w)$, from P. This is the equivalent of computing $P \setminus P \cap R(\overline{p'q'}, 3\ell, w)$, which requires $O(n)$ time, if $|P| = O(n)$. To cover $\frac{|P|}{2}$ points with segments in S^*, we must find a set $\hat{S} \subset S^*$, with $\hat{S} = \{\overline{p'q'}_1, \ldots, \overline{p'q'}_k\}$. Each segment $\overline{p'q'} \in \hat{S}$ covers at least $\frac{|P|}{2k}$ points. For any $i, j \leq k$ the following property must also hold in $\hat{S}$: $(P \cap R(\overline{p'q'}_i, 3\ell, w)) \cap (P \cap R(\overline{p'q'}_j, 3\ell, w)) = \emptyset$. Informally, for any pair of segments in $\hat{S}$, the points inside the rectangle of each segment should be mutually exclusive. If we are able to produce a set of segments, $\hat{S} \subset S^*$ using COMPUTESEGMENTS, then the running time of the subroutine is $O(nk)$, since the set subtraction operator is used for each segment in $\hat{S}$, and $|\hat{S}| = k$. Thus, the time complexity of finding a suitable set of segments $\hat{S} \subset S^*$, using the COMPUTESEGMENTS subroutine is $O(nk)$.

If we are unable to produce a set $\hat{S}$ containing segments which together cover $\frac{|P|}{2}$ points with the segments in $S^* \subset S''$, we provide a solution using the segments in $\mathscr{S}$. For $i \in [2, \ldots, \log n]$, we increase the width of $R(\overline{pq}, 3\ell, w)$ to $R(\overline{pq}, 3\ell, 2^i w)$ for each segment $\overline{pq} \in \mathscr{S}$. At each iteration of the for-loop, we take a subset, $\tilde{S} \subset \mathscr{S}$, that contains segments $\overline{pq} \in \mathscr{S}$ for which $P \cap R(\overline{pq}, 3\ell, 2^i w)$ clusters at least $\frac{|P|}{2k}$ points. We apply COMPUTESEGMENTS on $\tilde{S}$. If COMPUTESEGMENTS produces a set of segments $\hat{S} \subset \tilde{S}$ as defined previously, then we are able to cover $\frac{|P|}{2}$ points, and we do not have to continue the iteration of the for-loop. The time complexity for producing $\hat{S} \subset \mathscr{S}$ would also be $O(nk)$. Notice however, that it is possible to apply COMPUTESEGMENTS on a set $\tilde{S} \subset \mathscr{S}$ at most $\log n$ times until we find a suitable set of segments $\hat{S}$. For all the iterations where we are not able to find $\hat{S}$, the running time of applying COMPUTESEGMENTS on $\tilde{S}$ is $O(n)$ at each iteration if $\tilde{S} \neq \emptyset$ at that iteration. Thus, the worst case running time is given when we are only able to produce the set $\hat{S}$ after $\log n$ iterations, and has running-time $O(n \log n + nk)$.

Using the two results produced in this paragraph, the worst case time complexity of the FILTERING-CLUSTERING function is $O(n \log n + nk)$. □

Lemma 11. *The time complexity of algorithm 1 is* $O(n \log^2 n + k^2 \log^2 k \log n)$.

Proof. To asses the time complexity of algorithm 1, we can estimate the running time of the MAIN function. The MAIN function maintains a while loop until $|P| \leq 1$. In MAIN we call the CANDIDATESEGMENT and FILTERING-CLUSTERING functions. After each successful iteration of the while loop, FILTERING-CLUSTERING removes $\frac{|P|}{2}$ points. It follows, that after at most $\log n$ iterations, $|P| = 0$. In lemma 9 we have shown that the running time of CANDIDATESEGMENT is $O(k^2 \log^2 k)$. In lemma 10 we have shown the time complexity of FILTERING-CLUSTERING to be $O(n \log n + nk)$. Combining the results of lemmas 9 and 10, the time complexity of algorithm 1 is $O(n \log^2 n + nk \log n + k^2 \log^2 k \log n) \approx O(n \log^2 n + k^2 \log^2 k \log n)$. □

Chapter 4

Coreset Algorithm

In this chapter we provide a coreset algorithm which can be used for segment k-means clustering problems. A (k, ϵ)-coreset is formally defined as:

Definition 12 ((k, ϵ)-coreset)**.** *Let P be a set of points in $\mathbb{R}^2$. A weighted set $S \subset P$ is called a (k, ϵ)-coreset for any set $K \subset \mathbb{R}^2$ of segments if,*

$$|\text{COST}(S, K) - \text{COST}(P, K)| \leq \epsilon \cdot \text{COST}(P, K)$$

More extensively, a coreset is a data summarization technique which approximates data in some provable manner with respect to a cost function. The goal of a coreset is to compute a model which minimizes the cost function on the small coreset rather than on the entire data, without hindering the performance by more than a small constant factor [21].

The coreset algorithm we construct in this chapter is an instance of a *grid* [10] coreset construction. The cost of a set of segments produced by approximation algorithm 1 on the coreset has a ϵ-approximation error with respect to a point set P. The size of the coreset is $O(k \cdot \log^2 n \frac{\ell}{\epsilon^2 \cdot r})$.

Theorem 13 (Coreset Algorithm)**.** *Let P be a point set in the two-dimensional Euclidean space of size n. Let $k \in \mathbb{N}$ be a natural number and $0 < \epsilon \leq 1$ be an error parameter. Then, there exists an algorithm that returns a (k, ϵ)-coreset S, of size $|S| = O(k \log^2 n \cdot \frac{\ell}{\epsilon^2 r})$.*

4.1 (k, ϵ)-Coreset Algorithm

We provide the pseudocode of the coreset algorithm solving theorem 13 in Algorithm 2. In the next paragraph we provide a detailed description of each step in the algorithm.

Algorithm 2 Coreset Construction

Input: Set of points P, set of segments computed by algorithm 1, B, and optimal set of segments O.

Output: A coreset S

Let $S = \emptyset$ and $r = \frac{\Sigma_{p \in P} \text{DIST}(p,O)}{n}$
while $P \neq \emptyset$ **do**
  for $i \in [0 \dots \log n]$ **do**
    for segment $b \in B$ **do**
      Let G_b, be a grid of cells inside a rectangle, $R(b, 2^i \cdot 3\ell, 2^i \cdot 4r)$
      Let $x = 2^i \cdot \epsilon \min(3\ell, 4r)$ be the side length of a cell, $c \in G_b$
      for cell $c \in G_b$ **do**
        if $c \neq \emptyset$ **then**
          Sample a point $p \in c$, uniformly at random and let p become the representative of c, denoted as rep_c
          Let $q_c = P \cap c$ be the points inside of cell c, and let rep_c have weight w_c with $w_c = |q_c|$
          $S = S \cup \text{rep}_c$ and $P = P \setminus q_c$
Return S

In Algorithm 2 we present the pseudocode of the coreset algorithm. In this paragraph we explain each step of the algorithm.

The coreset algorithm 2 takes as input three parameters. A set of points P, the optimal set of segments for the segment k-means clustering problem of P, O, and a set of segments produced by approximation algorithm 1, B. The core idea of the coreset algorithm is to create a grid of cells around each segment $b \in B$ such that all points $p \in P$ are inside of a cell. Using this intuition, the algorithm consists of the following steps:

1. The first step in the algorithm is to create a grid of cells G_b, for each segment $b \in B$. The grid G_b takes the shape of a rectangle, $R(b, 2^i \cdot 3\ell, 2^i \cdot 4r)$, for i in range $i \in [0, \dots, \log n]$. Grid G_b consists of cells having side length, $x = 2^i \cdot \epsilon \min(3\ell, 4r)$.

2. The next step in the algorithm is to sample a point p, uniformly at random from every non-empty cell, $c \in G_b$.

3. The sampled point p becomes the representative of cell c. We denote the representative of cell c as rep_c. Let $q_c = P \cap c$, be a set containing the points inside c. The representative of each cell, rep_c, is weighted with $w_c = |q_c|$.

4. We add each representative point rep_c to the coreset S, i.e $S = S \cup \text{rep}_c$ and remove the points inside each cell, i.e $P = P \setminus q_c$.

5. If $P \neq \emptyset$ we return to step 1 but increment the value of i to create grids having larger cells.

6. Once $P = \emptyset$, we return the coreset S, containing the representative of each cell.

One aspect to notice is that when we increase the dimensions of each grid, for $i \in [0, \dots, \log n]$, the size of a cell inside a grid only increases for the square annulus, $R(b, 2^i \cdot 3\ell, 2^i \cdot 4r) \setminus R(b, 2^{i-1} \cdot 3\ell, 2^{i-1} \cdot 4r)$. This situation is shown in figure 4.1. The reason for such a construction is that

once we have to increase the dimension of the grid surrounding a segment $b \in B$, from $R(b, 2^{i-1} \cdot 3\ell, 2^{i-1} \cdot 4r)$ to $R(b, 2^i \cdot 3\ell, 2^i \cdot 4r)$, the points inside $R(b, 2^{i-1} \cdot 3\ell, 2^{i-1} \cdot 4r)$ have already been removed in the previous iteration. Hence, if we only create cells for the square annulus, the total number of cells produced for the coreset algorithm decreases.

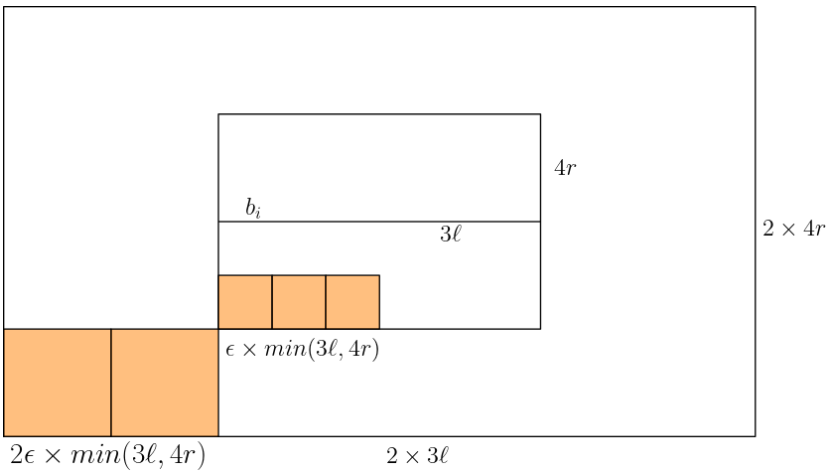

Figure 4.1: Sample segment b_i showing two of its associated grids. One grid having measurements, $3\ell, 4r$, while the larger grid being double in size. Each grid consists of cells, where the smaller grid has cells of side length $\epsilon \times \min(3\ell, 4r)$, while the larger grid has cells double in size.

4.1.1 Approximation Analysis

Theorem 14. *Let $P \subset \mathbb{R}^2$ be a set of n points in the 2-dimensional Euclidean space. Let $k \in \mathbb{N}$ be a natural number and $0 \leq \epsilon \leq 1$ be an error parameter. Algorithm 2 produces a (k, ϵ)-coreset.*

Proof. First we prove that the output of the coreset construction is indeed a (k, ϵ)-coreset. Recall that O is the optimal set of segments for P, and let B be a set of segments produced by approximation algorithm 1 on point set P. Each segment in B has length 3ℓ. For every point $p \in P$, let rep_p be its representative. Recall that every point p is associated(charged) with a unique representative. To this end, we define the coreset error as $ERROR = |\text{COST}(P,B) - \text{COST}(S,B)|$, where S denotes the computed coreset. Using the coreset algorithm description it follows that for any point $p \in P$ the following inequality holds:

$$\text{DIST}(p,B) \leq \text{DIST}(p, \text{rep}_p) + \text{DIST}(\text{rep}_p, B)$$

Using this inequality, together with the definitions of $\text{COST}(P,B)$ and $\text{COST}(S,B)$ respectively, we can define the coreset error as follows:

$$\begin{aligned} ERROR &= |\text{COST}(P,B) - \text{COST}(S,B)| \\ &= \Sigma_{p \in P}(\text{DIST}(p,B)^2 - \text{DIST}(\text{rep}_p, B)^2) \\ &= \Sigma_{p \in P}((\text{DIST}(p,B) - \text{DIST}(\text{rep}_p, B))((\text{DIST}(p,B) + \text{DIST}(\text{rep}_p, B)) \\ &= \Sigma_{p \in P}(\text{DIST}(p, \text{rep}_p) \cdot (2\text{DIST}(p,B) + \text{DIST}(p, \text{rep}_p)) \end{aligned}$$

Using the definition of the *ERROR* variable, we distinguish three cases with regards to the distance of any arbitrary point $p \in P$ to O and B. Throughout this section we assume that $4r < 3\ell$.

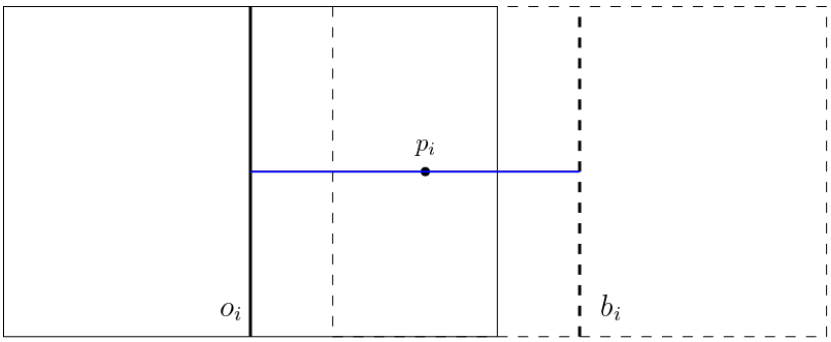

Figure 4.2: Point in P that is as close to B as it is to O

Case 1: The subset of points in P that are as close to B as they are to O: Formally, let R be an arbitrary distance, we are interested in the set $P_R = \{p \in P | \text{DIST}(p,B) \leq R \wedge \text{DIST}(p,O) \leq R\}$. We use $r = \frac{\text{DIST}(P,O)}{n}$ to bound $\text{DIST}(p,B)$. Since r denotes the average distance for a point in P to a segment in O, and there exists a segment in B that is as close as the segment in O, we can bound $\text{DIST}(p,B) = \text{DIST}(p,O) \leq r$, for any point $p \in P_R$. To bound the distance between p and its representative, rep_p, we use the knowledge that they both lie inside of the same grid cell. Thus, the distance between p and rep_p can be at most the size of the side length of the grid cell. Hence, $\text{DIST}(p, \text{rep}_p) \leq \epsilon \cdot 4r$. Merging both results,

$$\begin{aligned} ERROR &= \Sigma_{p \in P_R}(\text{DIST}(p, \text{rep}_p) \cdot (2\text{DIST}(p,B) + \text{DIST}(p, \text{rep}_p)) \\ &\leq \Sigma_{p \in P_R}(\epsilon \cdot 4r \cdot (2r + \epsilon \cdot 4r)) \\ &\leq 3\epsilon \Sigma_{p \in P_R}(4r)^2 \\ &\leq 3\epsilon \cdot 16 \Sigma_{p \in P_R}(r)^2 \\ &\leq 3\epsilon \cdot 16 \text{COST}(P,O) \end{aligned}$$

In the previous chapter, we have proven that the segments produced by the approximation algorithm 1 have a 16-approximation guarantee with regards to the optimal solution. It follows that:

$$ERROR \leq 3\epsilon \cdot \text{COST}(P,B)$$

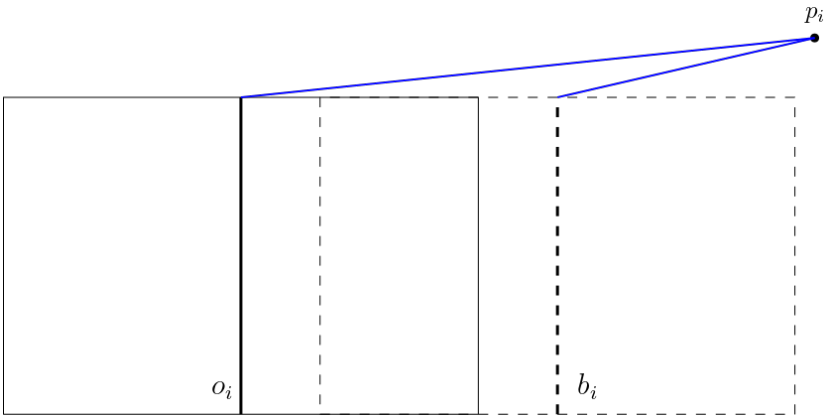

Figure 4.3: Point in P that is closer to B than to O.

Case 2: The subset of points in P that are closer to B than O. Formally we define the set P_B as follows:

$$P_B = \{p \in P \setminus P_R | \text{DIST}(p,B) < \text{DIST}(p,O)\}$$

For every point $p \in P_B$, it holds that $\text{DIST}(p,B) < \text{DIST}(p,O)$. Suppose p is inside a grid, G_b of type, $R(b, 2^i 3\ell, 2^i 4r)$. The size of a cell inside such grid is $x = 2^i \epsilon \cdot 4r$. Since p is in a grid of this size, it follows that $\text{DIST}(p,B) \geq 2^{i-1} 4r$. Thus, we can bound the distance between p and its representative, rep_p, as $\text{DIST}(p, \text{rep}_p) \leq 2^i \epsilon \cdot 4r \leq 2\epsilon \cdot \text{DIST}(p,B) \leq 2\epsilon \cdot \text{DIST}(p,O)$. Thus, the error for P_B is:

$$\begin{aligned} ERROR &= \Sigma_{p \in P_B} (\text{DIST}(p, \text{rep}_p) \cdot (2\text{DIST}(p,B) + \text{DIST}(p, \text{rep}_p)) \\ &\leq \Sigma_{p \in P_B} 2\epsilon \cdot \text{DIST}(p,O) \cdot (2\text{DIST}(p,O) + 2\epsilon \cdot \text{DIST}(p,O)) \\ &\leq 5\epsilon \Sigma_{p \in P_B} \text{DIST}(p,O)^2 \\ &\leq 5\epsilon \text{COST}(P,O) \end{aligned}$$

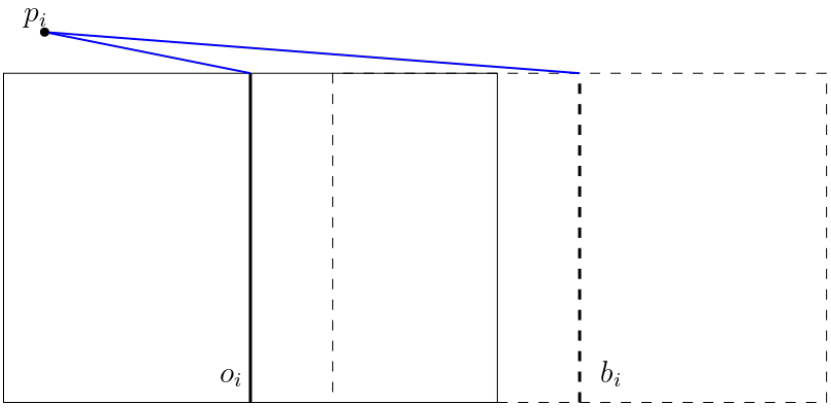

Figure 4.4: Point in P that is closer to O than to B

Case 3: The subset of points in P that are closer to O than B. Let P_O be defined as follows:

$$P_O = P \setminus (P_R \cup P_B)$$

The average distance between a point $p \in P$ and segment in O is defined by variable r. Since, points $p \in P_O$, are closer to a segment in O than to a segment in B, it follows that $\text{DIST}(p,B) > r$. Therefore, it means that points $p \in P_O$ could be inside a grid having cells of side length $2^i \epsilon \cdot 4r$. Since points $p \in P_O$ are inside of cells of side length $2^i \epsilon \cdot 4r$, it follows that $\text{DIST}(p,B) \geq 2^{i-1} \epsilon \cdot 4r$. Combining, $\text{DIST}(p,B) \geq 2^{i-1} 4r$, and the fact that both p and rep_p lie inside the same cell, we can bound $\text{DIST}(p, \text{rep}_p)$ as:

$$\text{DIST}(p, \text{rep}_p) \leq 2^i \epsilon \cdot 4r \leq 2\epsilon \cdot 2^{i-1} 4r \leq 2\epsilon \cdot \text{DIST}(p,B)$$

Using this bound, the error for point set P_O becomes:

$$\begin{aligned} ERROR &= \Sigma_{p \in P_O} (\text{DIST}(p, \text{rep}_p) \cdot (2\text{DIST}(p,B) + \text{DIST}(p, \text{rep}_p)) \\ &\leq \Sigma_{p \in P_O} (2\epsilon \cdot \text{DIST}(p,B) \cdot (2\text{DIST}(p,B)) + 2\epsilon \cdot \text{DIST}(p,B)) \\ &\leq 5\epsilon \cdot \Sigma_{p \in P_O} (\text{DIST}(p,B)^2) \\ &\leq 5\epsilon \cdot \text{COST}(P,B) \end{aligned}$$

Combined result: Combining the results of all 3 cases, the coreset error becomes:

$$\begin{aligned} ERROR &= \Sigma_{p\in P}(\text{DIST}(p,\text{rep}_p)\cdot(2\text{DIST}(p,B)+\text{DIST}(p,\text{rep}_p)) \\ &\leq 3\epsilon\cdot\text{COST}(P,B)+5\epsilon\cdot\text{COST}(P,O)+5\epsilon\cdot\text{COST}(P,B) \\ &\leq 8\epsilon\cdot\text{COST}(P,B)+5\epsilon\cdot\text{COST}(P,O) \end{aligned}$$

In the previous chapter we have proven that segments produced by the approximation algorithm 1 have a 16-approximation guarantee using a strong assumption, thus the coreset error becomes:

$$\begin{aligned} ERROR &\leq 8\epsilon\cdot\text{COST}(P,B)+5\epsilon\cdot 16\cdot\text{COST}(P,B) \\ &\leq 88\epsilon\cdot\text{COST}(P,B) \end{aligned}$$

We replace ϵ with $\frac{\epsilon}{88}$, then the coreset error becomes:

$$|\text{COST}(S,B)-\text{COST}(P,B)|\leq\epsilon\cdot\text{COST}(P,B) \tag{4.1}$$

□

4.1.2 Coreset Size

In this section we use the coreset algorithm described in section 4.1 to estimate the size of the coreset. Let $b\in B$ be a segment produced by applying algorithm 1 on a point set P. Around segment b, we create a grid of cells, G_b, whose boundary is defined by a rectangle, $R(b,2^i\cdot 3\ell,2^i\cdot 4r)$. The side length of each cell in G_b, is defined by, $x=2^i\epsilon\cdot 4r$. Let $A_b=2^{2i}3\ell\cdot 4r$ denote the surface area of grid G_b. To compute the number of cells in each grid, it suffices to divide A_b by the surface area of a cell, x^2. The surface area of a cell is given by:

$$x^2=(2^i\epsilon\cdot 4r)^2=2^{2i}\epsilon^2\cdot 16r^2$$

In section 4.1.1 we have used $\epsilon=\frac{\epsilon}{88}$, thus:

$$x^2=\frac{\epsilon^2\cdot 2^{2i}\cdot 16r^2}{88^2}$$

Now that we have both the surface area of a grid, and the surface area of a cell, the number of cells, #cells, in one grid G_b, is given by:

$$\begin{aligned} \#\text{cells} &= \frac{2^{2i}3\ell\cdot 4r}{\frac{\epsilon^2\cdot 2^{2i}\cdot 16r^2}{88^2}} \\ &= \frac{2^{2i}\cdot 12\cdot r\ell\cdot 88^2}{\epsilon^2\cdot 2^{2i}\cdot 16r^2} \\ &= \frac{7744\ell}{\epsilon^2\cdot r} \end{aligned}$$

We have shown that approximation algorithm 1 produces a set containing at most $k\log n$ segment. Around each segment we produce at most $\log n$ grids, in the coreset algorithm. Thus, the total number of cells produced by the coreset algorithm, #total, is:

$$\#\text{total}\leq k\cdot\log^2 n\cdot\frac{7744\ell}{\epsilon^2\cdot r}$$

Since for each cell, exactly one representative is added to the coreset S, the size of the coreset is:

$$|S|\leq k\cdot\log^2 n\cdot\frac{7744\ell}{\epsilon^2\cdot r}=O(k\cdot\log^2 n\cdot\frac{\ell}{\epsilon^2\cdot r})$$

Chapter 5

Experimental Results

In this chapter we present the experimental results of approximation algorithm 1. First we discuss the synethetic and real-world datasets used for the experiments. Afterwards we present the dynamic programming segment k-means clustering algorithm that is used to compare results with algorithm 1. We then present the experimental setup. Lastly, we perform experiments with algorithm 1 on synthetic and real-world datasets in the attempt to optimize parameters.

5.1 Datasets

5.1.1 Synthetic Datasets

We use a selection of synthetic datasets having well-defined characteristics as well as datasets from the Scikit-learn library in Python [35]. All of the datasets are in 2-dimensional Euclidean space.

Datasets with Special Characteristics

To benchmark algorithm 1 on datasets having a known optimal solution, we have created the following datasets:

- **Vertical Dataset:** - The vertical dataset consists of a set of points located along a vertical line. The dataset contains 20 points, with unit distance between consecutive points.
- **Two Horizontal Dataset:** - The two horizontal dataset consists of a set of points located along two parallel horizontal lines. Each line consists of 20 points.
- **Cross Dataset:** - The cross dataset consists of a set of points along a horizontal and vertical line of equal length, intersecting at their centers. Each line consists of 10 points.
- **Diagonal Dataset:** - The diagonal dataset consists of a set of points along the line with equation $y = x$. The dataset contains 20 points.
- **Horizontal Dataset with noise:** - The horizontal dataset with noise contains a set of points along a horizontal line, together with additional points scattered in the input space.

Scikit-learn Datasets

In addition to the synthetic datasets discussed in the previous section, we also perform experiments on the following datasets from the Scikit-learn library. Each dataset from Scikit-learn is generated with 500 points.

- **Moons Dataset:** - The moons dataset consists of a set of points along two half circles with the same radius, but different centers.
- **Circles Dataset** - The circles dataset consists of a set of points along two circles having the same center, but different radius. The circle with smaller radius is entirely inside of the larger circle.
- **Blobs Dataset:** - The blobs dataset generates two gaussian clusters of points. Each cluster contains the same number of points, but a different center.
- **Aniso Dataset:** - The aniso dataset is generated by applying a transformation matrix on the blobs dataset.

5.1.2 Real-world Datasets

We use a selection of real-world datasets with distinctive number of points and characteristics. Each dataset contain points in the 2-dimensional Euclidean space. All of the real-world datasets are publicly available on Kaggle [22].

- **Housing Prices Dataset** [42] - The Housing Prices Dataset consists of a set of houses and their attributes. The attributes of each house include the sale price, surface area, number of bedrooms and number of bathrooms. The dataset contains 545 points. We take the sale price and surface area of each house to create points in 2-dimensional Euclidean space. Each point has coordinates of type, (price, area).
- **Global Cargo Ships Dataset** [17] - The Global Cargo Ships Dataset consists of a set of cargo ships together with their attributes. Each cargo ship contains information regarding the year of production and the gross tonnage. The dataset contains 4000 points. We are interested in seeing the increase in gross tonnage capacity of cargo ships over time. Thus, we create $2D$ points having coordinates of type, (year, tonnage).
- **Amazon Stock Dataset** [23] - The Amazon Stock Dataset consists of information about the Amazon stock starting in 1997. The dataset contains information regarding the opening price, lowest price and highest price of the stock, on the stock market, on each open day. The dataset does not contain daily information on the price of the Amazon stock due to the stock market being closed on weekends and public holidays. For the experiments, we take the first 5000 days available in the dataset. We construct points in $2D$ space having the following coordinates, (day, opening price).
- **Top 500 Movies** [20] - The Movies Dataset consists of a list of 500 of the top performing Hollywood movies of all time. The dataset contains information regarding the budget of each movie as well as domestic and international gross revenues. The coordinates of the points constructed for the experiments are of type, (budget, gross revenue).

5.2 Segment k-means Dynamic Programming Algorithm

To benchmark the performance of approximation algorithm 1 we use the dynamic programming algorithm developed in [25]. `Segmented-Least Squares` is a dynamic programming approach of computing a set of segments S for an input set of points P, such that $\text{COST}(P,S)$ is within a chosen error value from the optimal solution. In section 2.2 we have provided the intuition behind the algorithm. A Python implementation of the `Segmented-Least Squares` algorithm is publicly available on GitHub [39]. The algorithm from [39] has been slightly adapted to output the coordinates of the endpoints of each produced segment in order to perform the comparison between the two algorithms. Throughout this section we will refer to the dynamic programming as SLS, in short for `Segmented-Least Squares`.

5.3 Experimental Setup

The segment k-means approximation algorithm 1 is implemented in Python 3.8.5. The experiments are performed on an Asus Rog G15 laptop, having an Intel $i7$ processor (2.6 GHz) and 16 GB of RAM.

5.3.1 Algorithms Comparison Experimental Setup

To test the robustness of the segment k-means approximation algorithm 1 we propose the following experimental setup. Notice that algorithm 1 contains several parameters which are not available for the algorithm in [25]. Namely, SLS does not take k as input for the number of segments to be produced, rather, it finds a set of segments whose cost is within an error parameter from the optimal solution for a given input dataset. Additionally, the approximation algorithm takes ℓ as a parameter which denotes the maximum length of each segment, while SLS allows segments of any length. For a meaningful comparison between our proposed algorithm and SLS we propose the following setup:

- **Step 1:** - Run the dynamic programming algorithm on a synthetic dataset. The number of segments produced by SLS on each synthetic dataset, becomes the value of k for algorithm 1, if $k > 1$.
- **Step 2:** - Run algorithm 1 using the value of k produced by SLS, 10 times for each dataset, while allowing segments of arbitrary length to be produced. The reason for performing 10 experiments on each dataset with algorithm 1 is that the approximation algorithm uses random sampling, meaning with each experiment we will achieve a different result.
- **Step 3:** - On both algorithms we compute the cost between the dataset and the segments produced by each algorithm respectively. We report the cost of the solution produced by SLS and compare it with the result having the lowest cost in the experiments performed with algorithm 1. In chapter 1, we have defined the cost function for this paper to be the sum of squared distances between a dataset and a set of centers.

5.3.2 Approximation Algorithm Experimental Setup

The only meaningful comparison between algorithm 1 and SLS takes place when algorithm 1 takes the value of k computed by the dynamic programming algorithm as a parameter. However,

the approximation algorithm contains many more parameters that are not available to SLS, which could optimized for more insightful results.

The approximation algorithm contains the following parameter values:

1. **k** - Expected number of segments.
2. ℓ - Maximum length of each segment.
3. θ - Angle of revolution used for the construction of candidate segments.
4. α - Constant in $\mathbb{N}$.
5. **r** - Average distance between a point and a segment in the optimal segment k-means solution. Clearly, the optimal solution is not available for the segment k-means problem of each dataset. In chapter 3 we have provided a method of estimating the value of r using the scale of the dataset. Thus, for practical purposes, we will transform the value of r into a variable which will be dependant on the size of the input space for each dataset.

There exists an additional condition that could be changed when performing experiments with algorithm 1. Recall from the algorithm description in section 3.1, at each iteration we are interested in finding a set of segments which together cover at least $\frac{|P|}{2}$ points, while each segment covers at least $\frac{|P|}{2k}$ points. We introduce a new variable, c, which holds the value for the fraction of points that must be covered by the set of segments at each iteration. The value of c in the theoretical algorithm would have value, $c = \frac{1}{2}$. However, for some datasets it is possible that we are interested in producing more segments. To do so, it is possible to decrease the value of c. For example, with $c = \frac{1}{4}$, we are interested in finding sets of segments which only cover at least $\frac{|P|}{4}$ points in the dataset, while the rectangle associated with each segment covers at least $\frac{|P|}{4k}$ points.

After performing the comparison between algorithm 1 and SLS, if the cost produced by algorithm 1 is significantly larger than the cost produced by SLS, then we perform an additional set of experiments only with algorithm 1 with the aim to find the best combination of parameters for each dataset. When we perform the additional experiments, we are mostly interested in finding the parameters which yield the lowest cost, regardless of the number of produced segments.

Real-World Datasets Experimental Setup Experiments on real-world datasets are only performed with algorithm 1. The reason for performing experiments on real-world datasets is to discover the quality of clustering produced by algorithm 1 on real data. For each dataset we present the solution which was deemed to most accurately summarize the characteristics of the data, together with the set of parameters used to achieve it. For some datasets, we may present additional solutions achieved while in the process of finding a solution which was deemed to accurately summarize the data. These solutions are only shown if they are deemed to be have a worthy point of discussion, or they present a potential flaw in the algorithm design.

5.4 Experiments

5.4.1 Synthetic Datasets

In this section we present the results produced by algorithm 1 compared to SLS using the synthetic datasets mentioned in the previous section. In each section we additionally provide more experiments using different parameters on algorithm 1.

Vertical Dataset

As mentioned in the experimental setup we first run the experiment on SLS to find the value of k needed to run algorithm 1. Running SLS on the vertical dataset yields one segment, as shown in figure 5.1. Clearly COST $= 0$ since all the points lie on the segment. .

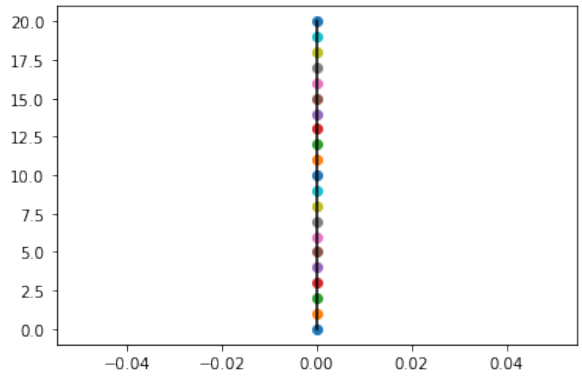

Figure 5.1: Segment produced by SLS and algorithm 1 on the vertical dataset.

According to the experimental setup, we should run algorithm 1 with $k = 1$. However, notice that the construction of algorithm 1 takes samples of points from the data set of size $\alpha k \log k$. With $k = 1$, the sample size is always 0, meaning the algorithm would not output any segments. Thus, for instances when $k = 1$ for the dynamic programming algorithm, we use $k = 2$, for algorithm 1.

Algorithm 1 produces a segment identical to SLS, as shown in figure 5.1, and thus also has COST $= 0$.

Two-Horizontal Dataset

The result of running SLS on the two-horizontal dataset is shown in figure 5.2.

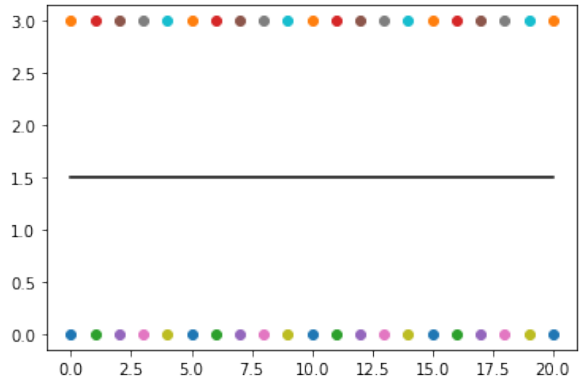

Figure 5.2: Result produced by the dynamic programming algorithm on the two-horizontal dataset.

The algorithm produces one segment, with every point having distance 1.5 to the segment. Therefore, COST $= 94.5$.

The result of running algorithm 1 on the two-horizontal dataset is shown in figure 5.3.

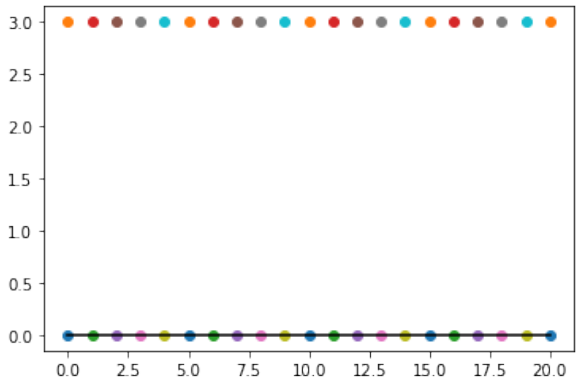

Figure 5.3: Result produced by the approximation algorithm on the two-horizontal dataset.

As captured in figure 5.3, algorithm 1 also produces one segment with $k = 2$. Without changing any other parameters, the cost is 189 since for half of the points, the distance is 3, whereas for the remaining points it is 0.

Clearly, without optimizing parameters for algorithm 1, SLS performs better in terms of minimizing the COST function. Hence, we proceed by performing experiments with algorithm 1, using different combinations of parameters.

	#segments	cost
$k = 2, \ell = 4, r = 2, \theta = \frac{\pi}{4}$	2	82.99
$k = 2, \ell = 7, r = 2, \theta = \frac{\pi}{4}$	3	61.10
$k = 2, \ell = 7, r = 1.5, \theta = \frac{\pi}{4}$	4	52.62
$k = 3, \ell = 7, r = 2, \theta = \frac{\pi}{4}$	1	94.50
$k = 3, \ell = 7, r = 0.74, \theta = 0$	3	29.61
$k = 2, \ell = 7, r = 0.74, \theta = 0$	10	1.00
$k = 2, \ell = 5, r = 1, \theta = 0$	6	66.56
$k = 4, \ell = 7, r = 1, \theta = \frac{\pi}{2}$	1	94.50
$k = 2, \ell = 7, r = 2, \theta = \frac{\pi}{2}$	5	41.79
$k = 2, \ell = 10, r = 2, \theta = \frac{\pi}{2}$	5	17.75

Table 5.1: Table showing the results of applying algorithm 1 with different parameters on the two-horizontal dataset.

In table 5.1 we present the results of applying the approximation algorithm on the two-horizontal dataset. For each combination of parameters, 10 experiment runs were performed, and the table presents the best result in terms of cost for each combination. As seen in the table, there exist combinations of parameters for which the cost is significantly smaller than the result produced by SLS. One setback of achieving a smaller cost is the larger number of segments produced. We were able to achieve the best result in terms of cost using $k = 2, \ell = 7, r = 0.74, \theta = 0$ as parameters, while using $c = \frac{1}{3}$.

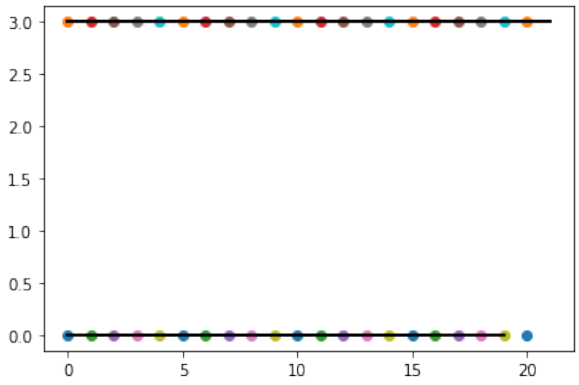

Figure 5.4: Sample result of running approximation 1 on the two-horizontal dataset with minimal cost.

Figure 5.4 shows the best result in terms of cost we have achieved while running algorithm 1 on the two-horizontal dataset. A trend shown in table 5.1 is the cost being smaller when using $\theta = 0$. The explanation for such phenomenon is rather simple. If $\theta > 0$, it is possible to create vertical or diagonal segments between the two horizontal lines. If r is large enough, the rectangle associated with such segments would cover enough points from the dataset, and thus would be added to the solution. Figure 5.5 shows a result for which such a segment is added to the solution. The parameters used to produce such a result are $k = 2, \ell = 3, r = 2, \theta = \frac{\pi}{2}$. The cost associated with this result is 311.00.

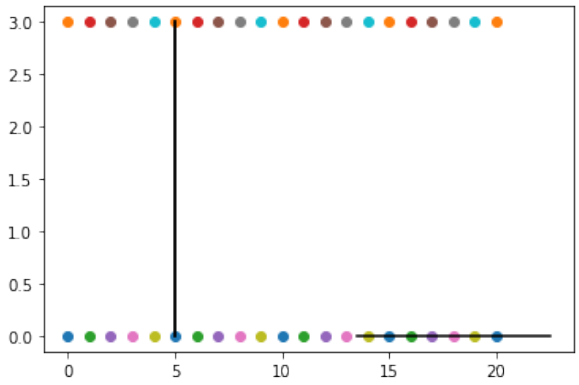

Figure 5.5: Figure depicting a situation when using $\theta = \frac{\pi}{2}$ is not desirable for the two-horizontal dataset.

Diagonal Dataset

The result of running SLS on the diagonal dataset is shown in figure 5.6. SLS produces one segment, and the cost associated with the segment is 0 as all points lie on the segment. The result of running algorithm 1 on the diagonal dataset can also be seen in figure 5.6. With $k = 2$ the algorithm produces 3 segments whose union create the optimal segment as produced by the dynamic programming algorithm. The cost associated with the 3 segments produced by algorithm 1 is also 0. Notice that it is theoretically possible to create 1 segment with associated cost 0 using algorithm 1 by initially sampling the bottom-left and top-right most points. With those 2 points

being sampled initially, a segment is created since ℓ is not used as a parameter, and the segment covers all of the points in the dataset. Thus, such a segment would satisfy the conditions for being added to the solution.

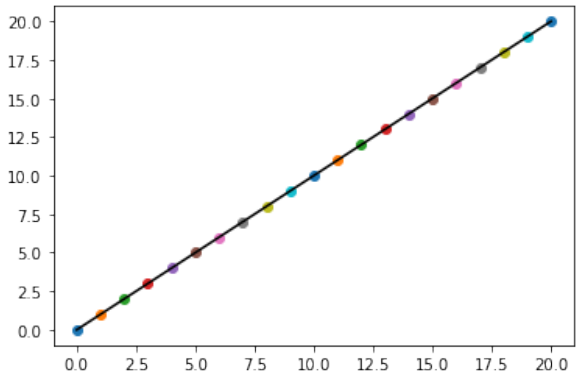

Figure 5.6: Result produced by SLS and algorithm 1 on the diagonal dataset.

Cross Dataset

The result of running SLS on the cross dataset is shown in figure 5.7.

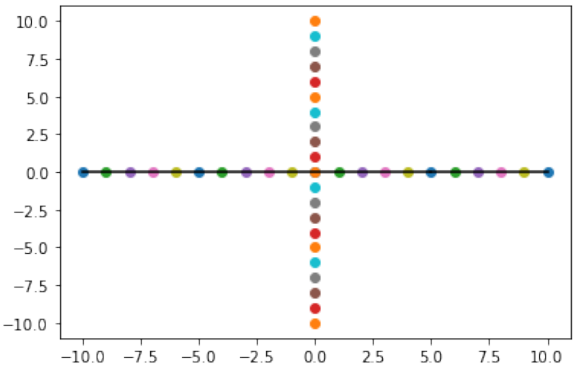

Figure 5.7: Result produced by the dynamic programming algorithm on the cross dataset.

As clearly depticted in figure 5.7 running the dynamic algorithm on the cross dataset does not produce the expected set of segments. The algorithm produces two segments having endpoints $(-10, 0), (5, 0)$ and $(5, 0), (10, 0)$ respectively. The vertical line in the cross dataset produces an INVALID VALUE ERROR and thus the algorithm does not produce a segment for the vertical line of points. The consequence of such error is the associated cost, which is 770.00.

After performing 10 experiments with algorithm 1 on the cross data set, figure 5.8 depicts the segments with the lowest associated cost.

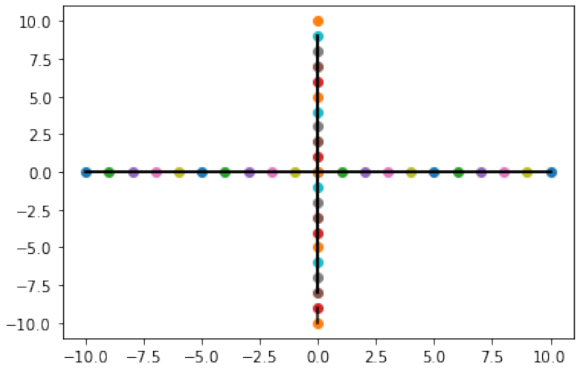

Figure 5.8: Set of segments with lowest associated cost when running algorithm 1 on the cross dataset.

As seen in the figure, all but one point is covered by segments. In total, the algorithm produced 10 segments with total associated cost equal to 1.0. Since the cross dataset consists of perpendicular lines of points, we perform experiments on different parameters using $\theta = \frac{\pi}{2}$. Table 5.2 presents the results of these experiments.

	#segments	cost
$k = 2, \ell = 7, r = 2, \theta = \frac{\pi}{2}$	4	0.0
$k = 2, \ell = 5, r = 2, \theta = \frac{\pi}{2}$	9	38.60
$k = 2, \ell = 7, r = 1, \theta = \frac{\pi}{2}$	12	0.00
$k = 2, \ell = 5, r = 1, \theta = \frac{\pi}{2}$	19	0.00

Table 5.2: Table showing the results of applying algorithm 1 on the cross dataset using $\theta = \frac{\pi}{2}$

As seen in table 5.2 there are combinations of parameters for which we are able to cover each point with a segment such that the associated cost is 0. Figure 5.9 depicts a solution with

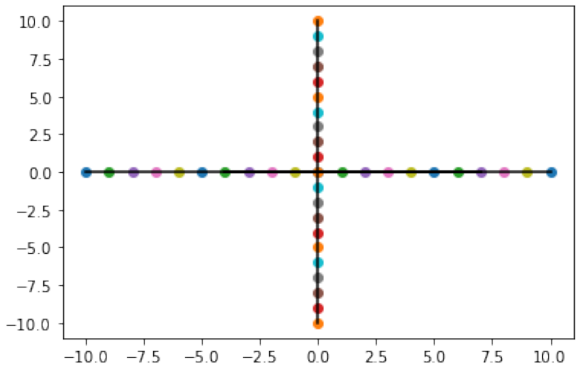

Figure 5.9: Result of running algorithm 1 on the cross dataset with associated cost 0.

$\text{COST} = 0$. The solution presented consists of 4 segments, and was achieved using the following parameters,

$$k = 2, \ell = 7, r = 2, \theta = \frac{\pi}{2}, c = \frac{1}{3}$$

Noisy Horizontal Dataset

Figure 5.10 presents the segments produced by SLS on the noisy horizontal dataset. The algorithm produces 3 segments with COST = 310.89.

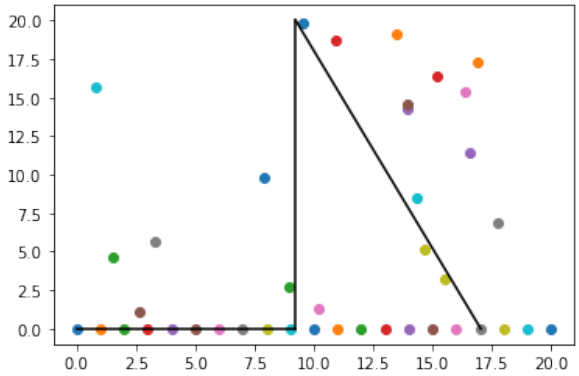

Figure 5.10: Result produced by the dynamic programming algorithm on the noisy horizontal dataset.

Since the dynamic programming algorithm produced 3 segments, for the direct comparison we run algorithm 1 with $k = 3$.

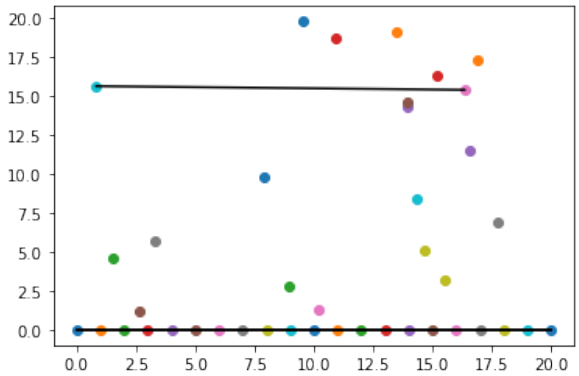

Figure 5.11: Sample result of running algorithm 1 on the noisy horizontal dataset.

Figure 5.11 shows the segments produced by algorithm 1 on the noisy horizontal dataset with $k = 3$. So far, we used $r = 2$, as default value for the comparison experiments. However, in the 10 experiments done for the noisy horizontal dataset, all of them reach the maximum recursion depth. Thus, for this experiment we use $r = 3$. Algorithm 1 produces 3 segments with COST = 294.62.

To see if it is possible to achieve lower associated cost, we run more experiments with different parameter combinations. Table 5.3 presents the achieved results with various combinations of parameters.

	#segments	cost
$k=2, \ell=7, r=3, \theta=\frac{\pi}{4}$	5	158.50
$k=2, \ell=5, r=3, \theta=\frac{\pi}{4}$	10	191.80
$k=2, \ell=5, r=3, \theta=\frac{\pi}{6}$	17	230.06
$k=2, \ell=3, r=3, \theta=\frac{\pi}{6}$	10	456.27
$k=3, \ell=7, r=3, \theta=\frac{\pi}{6}$	7	173.79
$k=2, \ell=7, r=2.5, \theta=\frac{\pi}{6}$	16	98.54
$k=2, \ell=7, r=2, \theta=\frac{\pi}{6}$	13	48.26

Table 5.3: Table showing the results of applying algorithm 1 on the noisy horizontal dataset

In figure 5.12 we present the segments with the smallest associated cost produced in the experiments. Clearly, for such a dataset to achieve minimal cost, the number of produced segments must increase drastically.

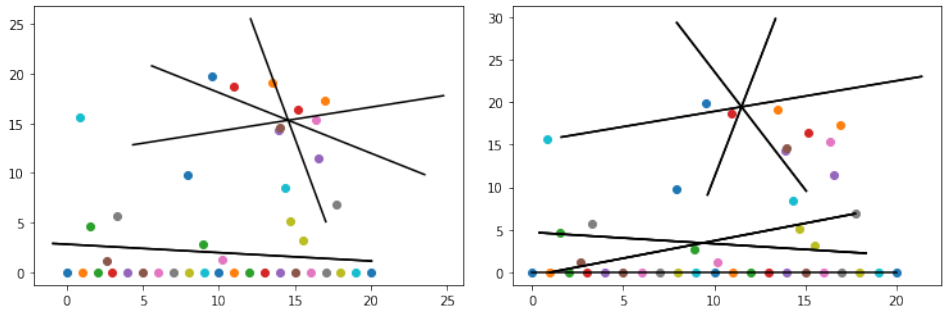

Figure 5.12: Results of running algorithm 1 on the noisy horizontal dataset with the smallest associated cost.

Moons Dataset

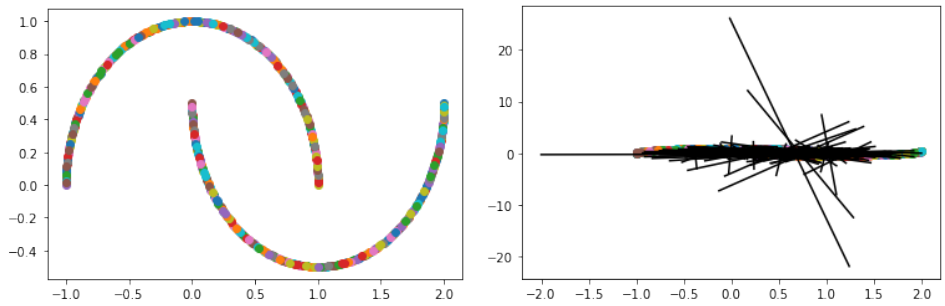

Figure 5.13: The left figure shows the moons dataset used for this experiment. The right figure shows the segments produced by running the dynamic programming algorithm on the moons dataset.

Figure 5.13 presents the moon dataset, as well as the segments produced by the dynamic programming algorithm. As depicted on the right side of the figure, the dynamic programming

algorithm produces many segments, which are much larger than the range of the points in the dataset. To be more exact, the algorithm produces 193 segments with a total cost of 1.56.

Clearly, it is not possible to run algorithm 1 with $k = 193$, as it would exceed the maximum recursion depth. Additionally, using $r = 2$ as we have done for most of the other experiments is not ideal as the associated rectangle of one segment is likely to cover all of the points in the dataset. Thus, for this experiment we will use $r = 0.5$ and $k = 2$.

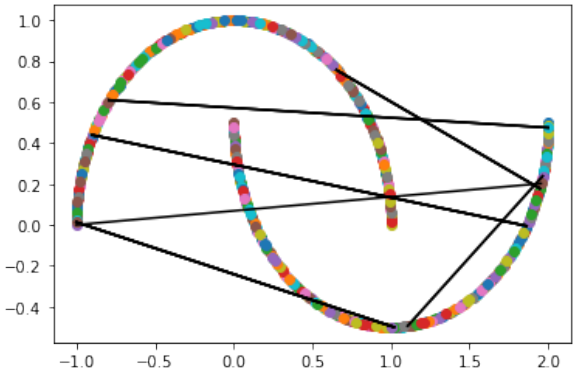

Figure 5.14: Result with lowest cost when running algorithm 1 on the moons dataset.

As seen in figures 5.13 and 5.14, the two algorithms produce completely different solutions on the moons dataset. While SLS produces 193 segments with COST $= 1.56$, algorithm 1 produces only 21 segments with COST $= 15.90$. We now proceed by running experiments with different combinations of parameters on algorithm 1 to see if it possible to achieve a cost close to the dynamic programming while using fewer segments.

	#segments	cost
$k = 2, \ell = 2, r = 0.5, \theta = \frac{\pi}{4}$	14	12.39
$k = 2, \ell = 2, r = 0.25, \theta = \frac{\pi}{4}$	20	7.34
$k = 2, \ell = 1.5, r = 0.1, \theta = \frac{\pi}{6}$	26	4.76
$k = 2, \ell = 1, r = 0.15, \theta = \frac{\pi}{6}$	46	3.36
$k = 2, \ell = 1, r = 0.1, \theta = \frac{\pi}{6}$	35	3.00
$k = 2, \ell = 1, r = 0.05, \theta = \frac{\pi}{6}$	84	1.43

Table 5.4: Table showing the results of applying algorithm 1 on the moons dataset

Table 5.4 shows results achieved by running algorithm 1 on the moons dataset with distinctive parameter combinations. In figure 5.15 we show the 84 segments produced with associated COST $= 1.43$. To achieve such a result, we used $c = \frac{1}{8}$. In doing so, we allow more segments to be produced.

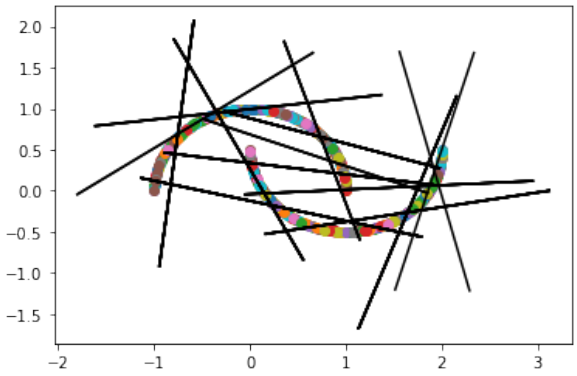

Figure 5.15: Segments with the smallest associated cost when running algorithm 1 on the moons dataset.

Blobs Dataset

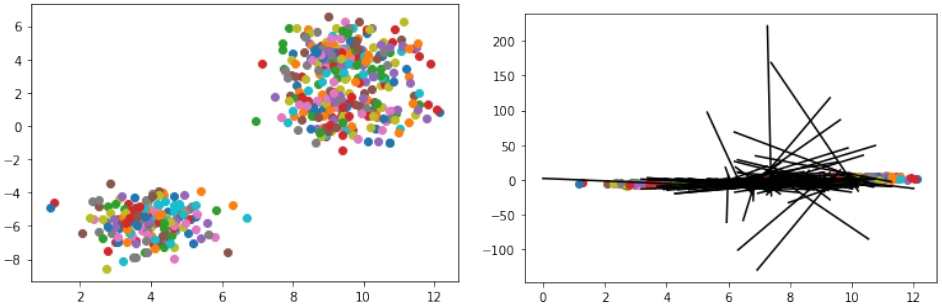

Figure 5.16: The left figure shows the blobs dataset used for this experiment. The right figure shows the segments produced by running the dynamic programming algorithm on the blobs dataset.

In figure 5.16 we present the instance of the blobs dataset used for the experiments on the left, while on the right the segments produced by SLS on the blobs dataset are shown. As with the moons dataset, the dynamic programming algorithm produces many segments in order to minimize the cost. The algorithm produced 128 segments with $\text{COST} = 96.65$. As previously, we will not run the experiments with algorithm 1 with $k = 128$, as it will not terminate. Hence, for the comparison, we run algorithm 1 with $k = 2$ and $r = 2$.

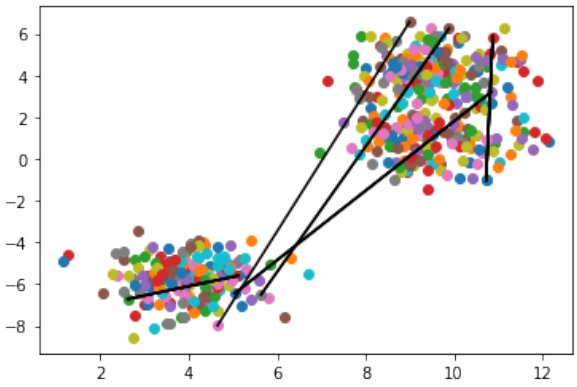

Figure 5.17: Result with lowest cost when running algorithm 1 on the blobs dataset.

In figure 5.17 we see the segments produced when performing experiments with algorithm 1 on the blobs dataset. The solution shown in the figure consists of 15 segments with associated COST $= 201.47$. As with the moons dataset we proceed by performing experiments with different parameters of algorithm 1.

	#segments	cost
$k = 2, \ell = 7, r = 2, \theta = \frac{\pi}{4}$	12	168.33
$k = 2, \ell = 4, r = 2, \theta = \frac{\pi}{4}$	12	183.11
$k = 2, \ell = 4, r = 1.5, \theta = \frac{\pi}{6}$	15	189.85
$k = 2, \ell = 3, r = 1.5, \theta = \frac{\pi}{6}$	17	82.26
$k = 2, \ell = 2, r = 1, \theta = \frac{\pi}{6}$	62	58.92

Table 5.5: Table showing the results of applying algorithm 1 on the blobs dataset

Table 5.5 shows the results achieved with various combinations of parameters with algorithm 1 on the blobs dataset. As seen in the last row of the table the lowest cost achieved is COST $= 58.92$ with 62 produced segments. In figure 5.18 we can see this result. As with the moons dataset, we had to decrease c, to $c = \frac{1}{6}$. Surprisingly, 10 experiments were performed with the parameters of the best achieved result, with $c = \frac{1}{8}$, however, we were not able to find a result with a smaller cost.

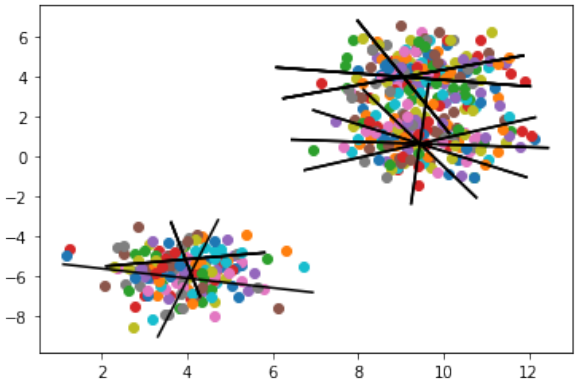

Figure 5.18: Segments with lowest associated cost when running algorithm 1 on the blobs dataset.

Circles Dataset

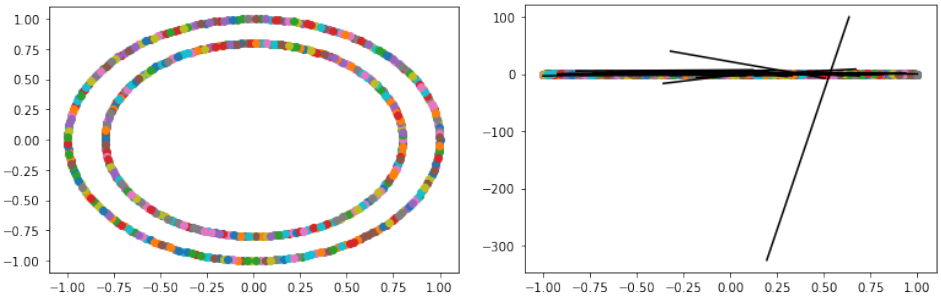

Figure 5.19: The left figure shows the circles dataset used for this experiment. The right figure shows the segments produced by running the dynamic programming algorithm on the circles dataset.

Figure 5.19 shows the circles dataset of the Scikit-learn Python package together with the set of segments produced by SLS. As with the other datasets from Scikit-learn, the dynamic programming algorithm produces many segments which cover the entire dataset. To be more exact, the algorithm produces 154 segments with associated COST $= 2.57$. To perform the direct comparison of algorithm 1 and the dynamic programming algorithm we will use $k = 2$. Since the input space of the dataset has range $[(-1,-1),(1,1)]$, we use $r = 0.2$ in algorithm 1.

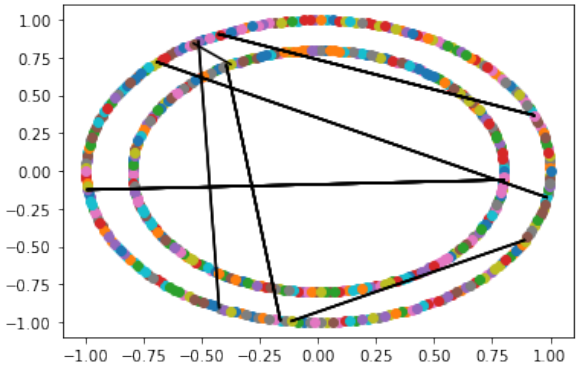

Figure 5.20: Set of segments with lowest cost when running algorithm 1 on the circles data set.

In figure 5.20 we show the set of segments produced by algorithm 1 after running 10 experiments on the circles dataset. The algorithm produced 28 segments with associated COST $= 13.70$. Clearly, such a result in terms of cost is inferior to the result of the dynamic programming algorithm, thus we begin optimizing parameters.

While optimizing parameters, the best result we have been able to achieve is a set of 38 segments with associated COST $= 2.68$. The result is depicted in figure 5.21. Although the cost is slightly larger than the cost produced by the dynamic programming algorithm, we only produced a fraction of the segments produced by the dynamic programming algorithm. This result was achieved with the following parameters,

$$k = 2, \ell = 0.67, \theta = \frac{\pi}{6}, r = 0.2, c = \frac{1}{8}$$

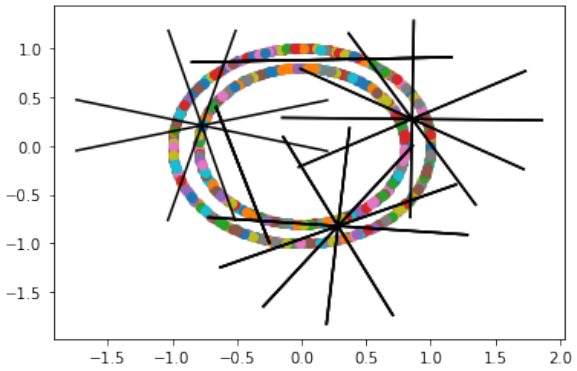

Figure 5.21: Segments produced by algorithm 1 on the circles dataset, with the best set of paramteres found.

Aniso Dataset

The result of applying SLS on the Aniso Dataset is shown in figure 5.22. The algorithm produces one segment which passes through both clusters having associated COST $= 36.40$.

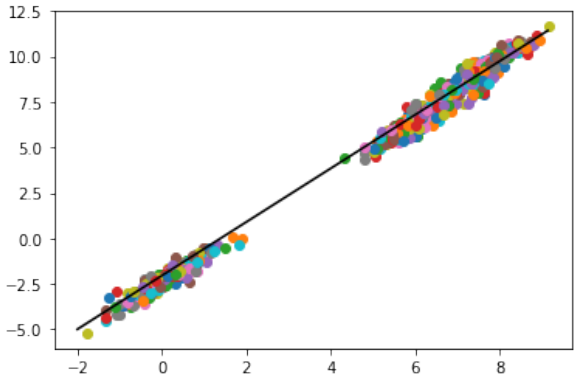

Figure 5.22: Result produced by the dynamic programming algorithm on the Aniso dataset.

When performing experiments with algorithm 1 on the aniso dataset, the lowest cost we have been able to achieve is presented in figure 5.23. The algorithm produces 10 segments having associated COST $= 11.81$. To achieve such result we only set $k = 2, r = 3$ and $c = \frac{1}{3}$.

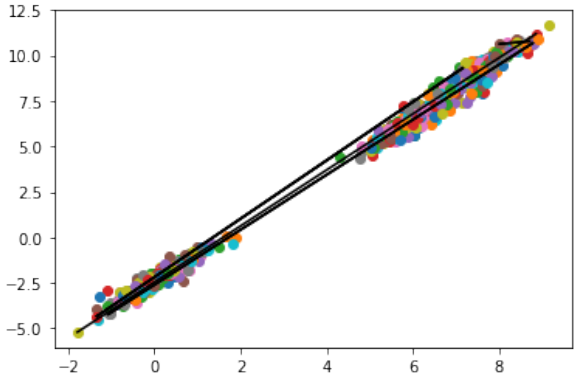

Figure 5.23: Output of algorithm 1 on the Aniso dataset.

As seen in figures 5.22 and 5.23 both algorithms produce segments which cross both clusters, rather than producing one segment per cluster. We are interested in finding a result where we produce one segment for each visible cluster. Hence, we perform experiments with algorithm 1 on the Aniso dataset, with differing values for ℓ.

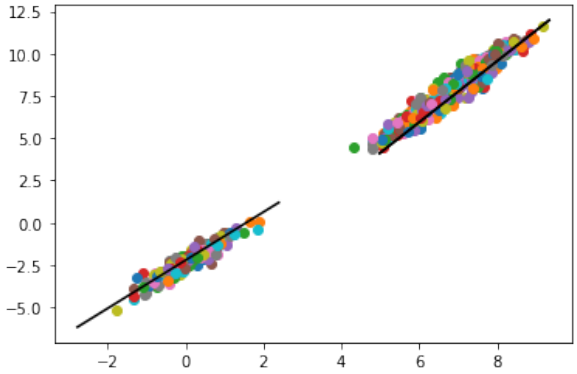

Figure 5.24: Expected output of algorithm 1 on the Aniso dataset.

The result in figure 5.24 shows a set of segments which does not cross both clusters in the dataset. The result consists of 2 segments having COST $= 21.67$. Although the result in figure 5.24 has a larger cost than in 5.23, the clustering better resembles the features of the dataset. To achieve the result in figure 5.24, the following parameters are used:

$$k = 2, r = 1.5, \ell = 3, c = \frac{1}{3}, \theta = 0$$

5.4.2 Real-world Datasets

In this section we present the results produced by algorithm 1 on the real-world datasets.

Housing Prices Dataset

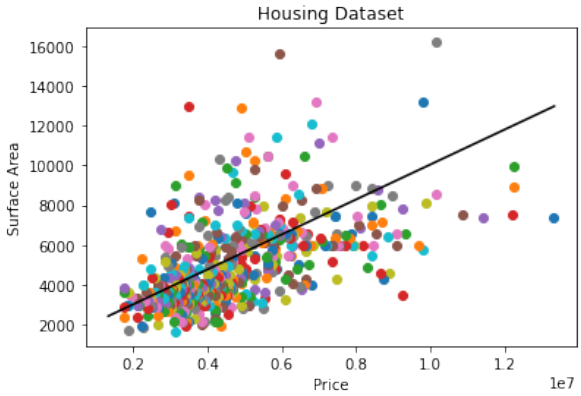

Figure 5.25: Output of algorithm 1 on the housing prices dataset.

Figure 5.25 presents the result for the segments k-means problem on the housing dataset, produced by algorithm 1. The solution consists of exactly one segment. The associated cost is rather large due to the scale of the x-axis. To achieve such a result we used the following set of parameters,

$$k = 2, \ell = 4000000, r = 3000, \theta = \frac{\pi}{4}$$

, while using the default value of $c = \frac{1}{2}$.

The reason for choosing such values for ℓ and r is to consistently find a result which fits the data well. Figure 5.26 shows the results of 10 experiments produced by running algorithm 1 on the housing dataset. As seen in the figure, using the combination of parameters mentioned previously, we are able to consistently produce segments which are close to the expected result.

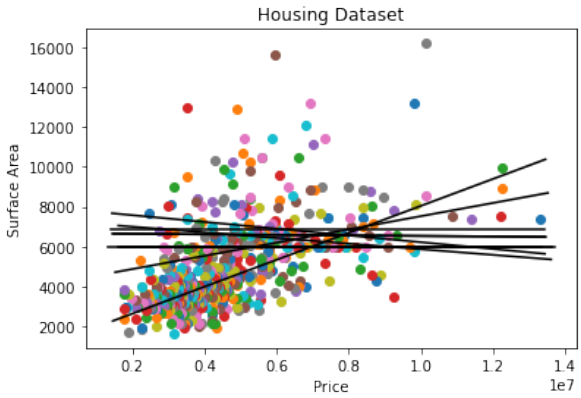

Figure 5.26: Results produced by algorithm 1 on the housing dataset on 10 experiments.

When choosing a significantly larger value for r, such as $r = 1750000$, and keeping all other parameters the same, the algorithm does not consistently produce a desirable solution.

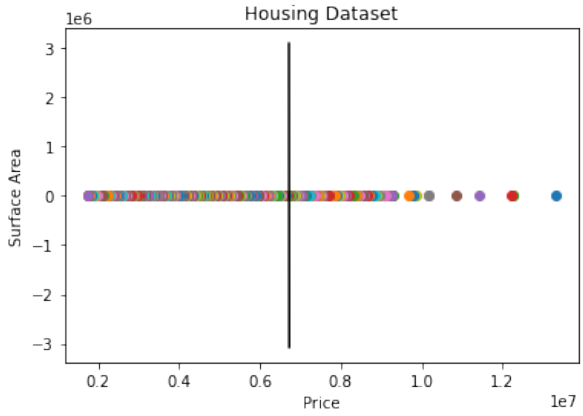

Figure 5.27: Non-optimal set of parameters applied to algorithm 1 on the housing dataset.

Figure 5.27 shows a solution produced by algorithm 1 on the housing dataset using $r = 1750000$. Clearly, the produced segment is too large, and does not produce a suitable summary of the data. The reason for the algorithm producing such a result, with $r = 1750000$, is that the rectangle associated with the segment will cover at least $\frac{1}{2}$ of the points in the input, and thus will be added to the solution.

Global Cargo Ship Dataset

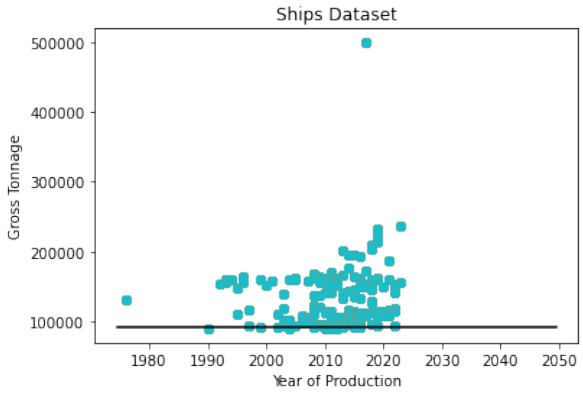

Figure 5.28: Algorithm 1 result on the cargo ship dataset.

In figure 5.28 we depict the result having lowest cost after performing experiments with algorithm 1 on the ships dataset. The parameters used to achieve the result are,

$$k = 2, \ell = 25, r = 200000, \theta = \frac{\pi}{4}$$

, whereas $c = \frac{1}{2}$. Using this combination of parameters we are able to consistently produce a result. However, when trying smaller values for ℓ and r, such as $\ell = 20$ and $r = 150000$, we reach the

maximum recursion depth without finding a result. In table 5.6 we present other parameters which have been able to produce a result, together with the cost.

	#segments	cost
$k = 2, \ell = 25, r = 200000, \theta = \frac{\pi}{4}$	1	12789175536080
$k = 5, \ell = 25, r = 150000, \theta = \frac{\pi}{4}$	1	11894537853120

Table 5.6: Results of applying algorithm 1 on the global cargo ships dataset

Amazon Stock Dataset

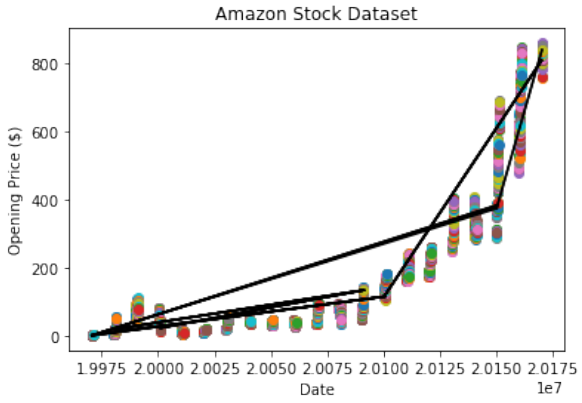

Figure 5.29: Result of running algorithm 1 on the Amazon stock dataset.

In figure 5.29 we present the result which was deemed to best summarize the Amazon stock dataset. The result consists of 16 segments, and was achieved using the following parameters:

$$k = 7, r = 50, \theta = 0, c = \frac{1}{12}$$

Notice that the value for ℓ is not mentioned in the list of parameters. The reason being that while choosing values for ℓ, we were not able to produce a result which was deemed to be suitable. Thus, we have removed the step which makes each segment to be of length 3ℓ in the algorithm to produce figure 5.29, and thus allow segments of any length to be produced.

The number of segments produced for the Amazon data stock is quite large due to using $c = \frac{1}{12}$. Such a result does not match the result we expected to be optimal. However, if we process the figure, it is possible to find trends which better match the expectations of the summarization of the dataset.

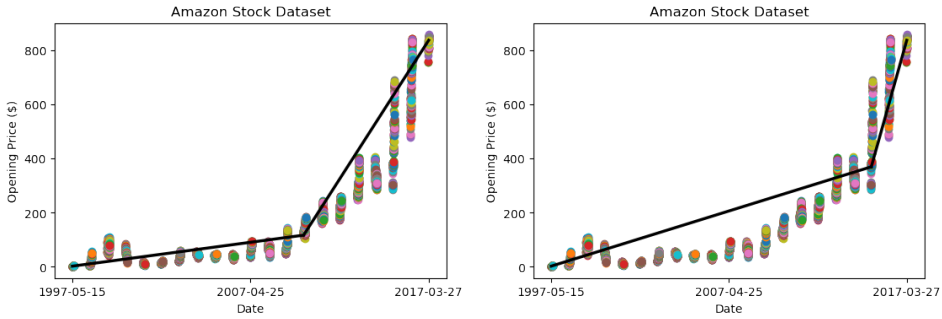

Figure 5.30: Result produced by algorithm 1 on the Amazon dataset after processing.

In figure 5.30, we present two subsets of the result in figure 5.29. The two subsets are able to depict the two main movements in the Amazon stock price. Namely, they both contain one segment, for the years where the Amazon stock had a stable price in the $[0, 200]$ range. Additionally, the subsets are also able to depict the steep increase in price in the latter years.

Movies Dataset

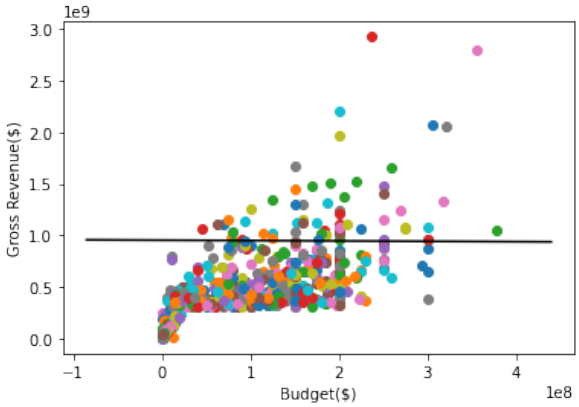

Figure 5.31: Result of running algorithm 1 on the movies dataset.

Figure 5.31 depicts the result having the lowest associated cost when solving the segment k-means problem on the movies dataset. To achieve the result of figure 5.31 we use the following parameters,

$$k = 2, \ell = 175000000, r = 200000000, \theta = \frac{\pi}{4}$$

We use $c = \frac{1}{2}$, in order to find segments that cover more points. To be able to lower the values for r and ℓ, we decrease the value of c, to $c = \frac{1}{6}$, which allows the use of $\ell = 75000000$ and $r = 100000000$. With the smaller parameters, we achieve the result in figure 5.32, which produces 39 segments.

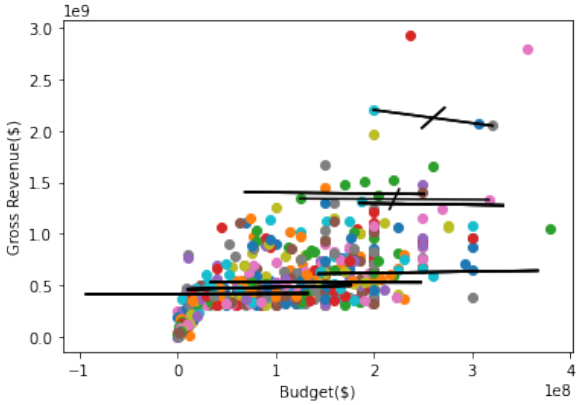

Figure 5.32: Result of running algorithm 1 on the movies dataset with $c = \frac{1}{6}$.

When not applying the filter on the length of the produced segments, we have been able to produce the result shown in figure 5.33. The result shown in the figure consists of 9 segments and has been produced using the following parameters,

$$k = 2, r = 100000000, \theta = \frac{\pi}{4}$$

, while using $c = \frac{1}{3}$. We have attempted to produce a result with $c = \frac{1}{2}$, when the length filter is not applied. However, we reach the maximum recursion depth in 10 experiments with differing values of r.

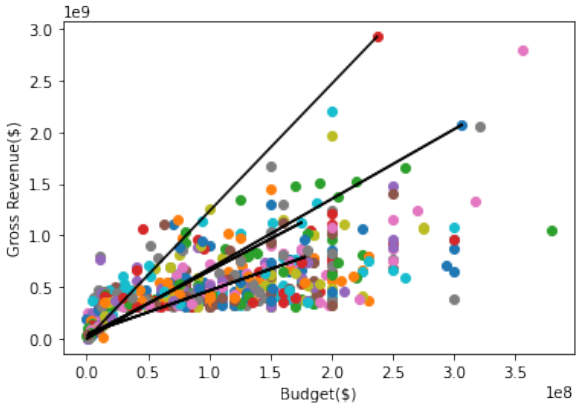

Figure 5.33: Result of running algorithm 1 on the movies dataset without applying the filter.

5.5 Performance of the Segment k-means Algorithm

In this section we look at the performance of approximation algorithm 1 compared to the SLS algorithm. Namely, we take each dataset used for experiments in the previous section and perform one additional experiment to produce the running time of both algorithms. When performing the running time experiment on algorithm 1, we use a combination of parameters used during the main experimental runs. Note that for the experiments in the previous section, the algorithms required more time to terminate, since plots were produced. For this experiment, we are only interested in the running time of the algorithms, thus we will omit the creation of plots.

5.5.1 Running Time Comparison

Algorithm	Moons	Circles	Blobs	Aniso	Housing	Cargo	Amazon	Movies
SLS	5.39	5.40	5.71	5.80	6.01	330	630	7.43
Approx	0.1	4.5	1.5	2.0	8.9	12.9	27.8	0.1

Table 5.7: Running time (in seconds) of running algorithm 1 and SLS on synthetic and real-world datasets.

In table 5.7 we show the times required for termination of both SLS and approximation algorithm 1 on various datasets. Due to the randomness of algorithm 1, we performed 5 running-time experiments for each dataset, and presented in table 5.7, the average time required for termination. As seen in the table, when using an appropriate set of parameters on algorithm 1, we are able to reach termination in fewer than 30 seconds even for the Amazon stock dataset which contains 5000 points. As for SLS, when running experiments with datasets having fewer than 1000 points, it is able to produce a segment in 5 seconds on average, however, when the size of the dataset increases, the running-time increases exponentially.

5.5.2 Large Datasets

All of the experiments so far have been performed on relatively small datasets, with the Amazon stock dataset being the largest, containing 5000 points. With the synthetic datasets from the Scikit-learn library, it is possible to select the number of points to be generated. We are interested in testing the running-time performance of algorithm 1 on datasets of much larger size. The experiments are performed on each of the synthetic datasets from Scikit-learn that have been described at the beginning of the section, using the combination of parameters which yielded the solution having the lowest cost. The results are shown in table 5.8.

Points	Moons	Blobs	Circles	Aniso
10000	1:29.20	0:08.30	0:08.66	0:11.80
50000	2:37.00	3:25.20	4:43.12	3:55.80
75000	5:36.20	6:34.00	6:57.80	6:56.70
100000	12:02.60	15:15.70	16:14.30	15:22.0

Table 5.8: Running-time (in minutes) of running algorithm 1 on Scikit-learn synthetic datasets.

As seen in table 5.8, there are differences in running time when performing experiments on different synthetic datasets with the same number of points. There are 2 possible explanations for this phenomenon: The first possible cause is the random sampling which is performed in the algorithm. It is possible that in some experiments, the algorithm sampled points which enabled the production of suitable segments earlier than for other experiments. The second possible explanation would be the difference in parameters used. Even though, the input space of the synthetic datasets is similar, each dataset has distinctive features, meaning that using a standard value of r and ℓ would not be possible for each dataset. Thus, it is possible that for the datasets where a larger value for r and ℓ is used, the running time is lower.

Chapter 6

Conclusions

In this chapter we discuss the results achieved in this paper. We then discuss how the algorithm in chapter 3 compares to the dynamic programming algorithm from [25]. We then propose potential methods to improve the algorithm.

6.1 Discussion

The research in the thesis started in chapter 3 where we constructed the approximation algorithm solving the segment k-means problem. We construct a randomized approximation algorithm which produces a set $\mathscr{S}$, containing at most $k \log n$ segments. The running time of the algorithm is $O(n \log^2 n + k^2 \log^2 k \log n)$. In chapter 4 we present a coreset construction algorithm which produces a coreset for the segment k-means problem having size $O(k \log^2 n)$. While running the algorithm designed in chapter 3 on the coreset, the result has a small error value, ϵ.

In chapter 5 we perform experiments to benchmark algorithm 1. We use a selection of synthetic and real-world datasets to perform the experiments. We run both algorithm 1 and the algorithm in [25] on synthetic datasets to benchmark the performance of our algorithm. To make the comparison fair, we only set the parameter k for algorithm 1, and keep all other parameters in their default values. For the experiments we were mainly interested in the cost produced by both algorithms rather than the number of produced segments. For the most part, when using algorithm 1 by only setting the parameter value of k , the dynamic programming algorithm is able to produce a set of segments with lower cost on the majority of the datasets. However, when setting values for the other available parameters, we have been able to find multiple combinations of parameter values that can produce a set of segments with lower cost than the dynamic programming algorithm. Although it was not the main priority, for most of the synthetic datasets, we have been able to produce results with lower cost and fewer segments than the dynamic programming implementation. The largest difference between the two algorithms can be noticed when performing experiments on the Scikit-learn synthetic datasets. For those datasets, the algorithm in [25] produces many segments, which are much larger than the input space. With algorithm 1 we were able to produce results with comparable cost with regards to the dynamic programming algorithm, while using a fraction of the segments.

We have also performed experiments using algorithm 1 on real-world datasets. For the most part we were able to produce results which are able to visually summarize the data. For the real-world datasets we were not particularly interested in the associated cost due to the potential large value, but rather we were interested in producing sets of segments which produce the expected

clustering. For some real-world datasets we had to compromise filtering the length of the segments to produce a result which clusters the dataset as expected.

6.2 Future Work

Algorithm 1 has been benchmarked against the algorithm provided in [25] on a selection of synthetic datasets. We have seen in chapter 5 that with a suitable set of values for the available parameters, algorithm 1 is able to produce a more desirable result than the algorithm in [25]. Nevertheless, being able to produce a better result also comes at a cost in terms of performance. Due to the nature of the algorithm, where we initialize a set of segments using random sampling, we had to perform 10 experiments with each combination of parameters in order to find a worthy result. In future research, a different sampling method should be used. It would be possible to have a sampling method using probability, similar to the k-means++ sampling [5]. The new sampling method would work by first sampling a random point from the dataset, and then with high probability sample points which have distance at most 3ℓ to the randomly selected point. With this sampling method, we would have the ability to entirely remove the filtering step in the algorithm, thus saving performance.

An additional method of improving performance would be to remove the variable θ. We use θ to create copies of each generated segment in order to generate a segment for each optimal cluster of points, such that the angular distance between the generated segment and the optimal segment for that cluster is at most θ. However, we have shown in chapter 3 that the value of θ may be small, which means that many copies of each segment must be generated. Instead of creating such copies, it would be possible to perform a rotational sweep centered at each generated segment. With the sweep, we are interested in the instance of each segment where there exist the most points within a distance $4r$ from the segment. By using a sweep, we remove the need to generate additional copies of each segment, which should save performance in the segment generation part of the algorithm.

Lastly, in the algorithm description we use a value r which denotes the average distance between a point and a segment in the optimal solution. However, when performing experiments, the value of r will not be available for many datasets, meaning we have to estimate the value of r based on the scale of the dataset. Instead of attempting to find segments whose associated rectangles cover a desired number of points, it would be possible to adapt the algorithm as follows: Around each segment, a rectangle is placed in such a way that a desired number of points is covered by the rectangles, while we attempt to minimize the width of the largest rectangle. In doing so, we remove the necessity of a value for r in practical situations, and we are still able to bound the approximation factor to the size of the largest rectangle.

Bibliography

[1] P. K. Agarwal, S. Har-Peled, and K.R. Varadarajan. Geometric approximations via coresets. *Combinatorial and Computational Geometry - MSRI Publications, 52:1–30*, 2005. 7

[2] Pankaj K Agarwal and Cecilia M Procopiuc. Approximation algorithms for projective clustering. *Journal of Algorithms*, 46(2):115–139, 2003. 10

[3] Sara Ahmadian, Ashkan Norouzi-Fard, Ola Svensson, and Justin Ward. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *CoRR*, abs/1612.07925, 2016. 3

[4] Alauddin Yousif Al-Omary and Mohammad Shahid Jamil. A new approach of clustering based machine-learning algorithm. *Knowledge-Based Systems*, 19(4):248–258, 2006. 3

[5] David Arthur and Sergei Vassilvitskii. K-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, page 1027–1035, USA, 2007. Society for Industrial and Applied Mathematics. 8, 55

[6] Mirking B. *Clustering for Data Mining: A Data Recovery Approach (1st ed.)*. Chapman and Hall/CRC, 2005. 3

[7] L. V. Bijuraj. Clustering and its applications. In *Proceedings of National Conference on New Horizons in IT -*, 2013. 3

[8] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez. Recommender systems survey. *Knowledge-Based Systems*, 46:109–132, 2013. 3

[9] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k -means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54:764–771, 2015. Eleventh International Conference on Communication Networks, ICCN 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Data Mining and Warehousing, ICDMW 2015, August 21-23, 2015, Bangalore, India Eleventh International Conference on Image and Signal Processing, ICISP 2015, August 21-23, 2015, Bangalore, India. 1, 3, 8

[10] Dan Feldman. Introduction to core-sets: an updated survey. *CoRR*, abs/2011.09384, 2020. 10, 25

[11] Dan Feldman, Amos Fiat, Micha Sharir, and Danny Segev. Bi-criteria linear-time approximations for generalized k-mean/median/center. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, SCG '07, page 19–26, New York, NY, USA, 2007. Association for Computing Machinery. 12

[12] Dan Feldman, Morteza Monemizadeh, and Christian Sohler. A ptas for k-means clustering based on weak coresets. In *Proceedings of the Twenty-Third Annual Symposium on Computational Geometry*, SCG '07, page 11–18, New York, NY, USA, 2007. Association for Computing Machinery. 10

[13] GeekForGeeks. https://www.geeksforgeeks.org/linear-regression-python-implementation 5, 6

[14] Mazumdar S. Har-Peled S. Coresets for k-means and k-median clustering and their applications. In *Department of Computer Science, DCL 2111; University of Illinois, pp 7-9*, 2003. 6, 9, 10

[15] IBM. https://www.ibm.com/topics/knn. 3

[16] IBM. https://www.ibm.com/topics/linear-regression. 5

[17] Ibrahim. https://www.kaggle.com/datasets/ibrahimonmars/global-cargo-ships-dataset. 32

[18] Piotr Indyk. Sublinear time algorithms for metric space problems. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing*, STOC '99, page 428–434, New York, NY, USA, 1999. Association for Computing Machinery. 9

[19] MacQueen J. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symp. on Math. Statist. and Prob., Vol. 1 (Univ. of Calif. Press, 1967), 281–297*, 1967. 3, 8

[20] Joakim Arvidsson. https://www.kaggle.com/datasets/joebeachcapital/top-500-hollywood-movies-of-all-time. 32

[21] Ibrahim Jubran, Alaa Maalouf, and Dan Feldman. Introduction to coresets: Accurate coresets. *CoRR*, abs/1910.08707, 2019. 25

[22] Kaggle. https://www.kaggle.com/. 32

[23] Kannan Ravinther. https://www.kaggle.com/datasets/kannan1314/amazon-stock-price-all-time. 32

[24] Tushar Kansal, Suraj Bahuguna, Vishal Singh, and Tanupriya Choudhury. Customer segmentation using k-means clustering. In *2018 International Conference on Computational Techniques, Electronics and Mechanical Systems (CTEMS)*, pages 135–139, 2018. 1

[25] Tardos Eva Kleinberg John. *Algorithm Design*. Pearson, 2005. 1, 6, 9, 10, 12, 33, 54, 55

[26] Madhukar R Korupolu, C Greg Plaxton, and Rajmohan Rajaraman. Analysis of a local search heuristic for facility location problems. *Journal of algorithms*, 37(1):146–188, 2000. 9

[27] S.R. Nanda, B. Mahanty, and M.K. Tiwari. Clustering indian stock market data for portfolio management. *Expert Systems with Applications*, 37(12):8793–8798, 2010. 3

[28] Sharma P. The ultimate guide to k-means clustering: Definition, methods and applications. https://www.analyticsvidhya.com/blog/2019/08/comprehensive-guide-k-means-clustering/, 2019. 8

[29] J.M Peña, J.A Lozano, and P Larrañaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters*, 20(10):1027–1040, 1999. 8

[30] Probability. https://www.probabilitycourse.com/chapter6/6_2_1_union_bound_and_exten.php. 17

[31] Surya Pryi. https://www.geeksforgeeks.org/clustering-in-machine-learning/. 1, 3, 8

[32] Sankar Rajagopal. Customer data clustering using data mining technique. *CoRR*, abs/1112.2663, 2011. 3, 8

[33] Gavita Regunath. 10 incredibly useful clustering algorithms you need to know. *Advancing Analytics*, 2022. 1, 3

[34] Guy Rosman, Mikhail Volkov, Dan Feldman, John W Fisher III, and Daniela Rus. Coresets for k-segmentation of streaming data. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc., 2014. 1, 10

[35] Scikit-learn. https://scikit-learn.org/stable/. 4, 31

[36] Eran Segal and Daphne Koller. Probabilistic hierarchical clustering for biological data. In *Proceedings of the sixth annual international conference on Computational biology*, pages 273–280, 2002. 1

[37] Akshay Sinha. K-means clustering. *Medium*, 2019. 3

[38] Sanjeevan Sivapalan, Alireza Sadeghian, Hossein Rahnama, and Asad M. Madni. Recommender systems in e-commerce. In *2014 World Automation Congress (WAC)*, pages 179–184, 2014. 3

[39] solohikertoo. https://github.com/solohikertoo/segmented-least-squares. 33

[40] Stanford. https://web.stanford.edu/class/archive/cs/cs109/cs109.1218/files/student_drive/6.1.pdf. 17

[41] Wolfram. https://mathworld.wolfram.com/TaylorSeries.html. 17

[42] Yasser. https://www.kaggle.com/datasets/yasserh/housing-prices-dataset. 32

[43] Sobia Zahra, Mustansar Ali Ghazanfar, Asra Khalid, Muhammad Awais Azam, Usman Naeem, and Adam Prugel-Bennett. Novel centroid selection approaches for kmeans-clustering based recommender systems. *Information Sciences*, 320:156–189, 2015. 1, 3