UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

Department of Information Engineering

Master's Degree in Computer Engineering

curriculum Artificial Intelligence and Robotics

Master Thesis

A study on clustering algorithms with predictions

Candidate:

Leonardo Sforzin

Supervisor:

prof. Fabio Vandin

Academic Year 2022 - 2023

October 10^{th}, 2023

Contents

Abstract

Algorithms are often analyzed in their worst case scenarios in order to have a better understanding about their performances. Moreover, it is important to highlight that in these instances algorithms cannot improve. For this reason a new approach was developed, exploiting the ability of adaptation and prediction of machine learning: algorithms with predictions. The main advantage brought by using machine learning is the fact that it is possible to exploit hidden information present in our data, for example an underlying distribution. This key concept lies at the core of the present thesis, in which we studied and developed original algorithms equipped with predictors for the k-means clustering task in a pseudo-online setting, that is, datasets are divided into subsets and we can analyze only one subset at a time. k-means clustering is a common and widely used method in machine learning and data mining whose primary objective is to partition a given set of elements into k clusters in order to minimize a specific cost function. The pseudo-online setting implies that there are multiple set of elements to cluster: in this thesis we consider the assumption that these sets belong to the same distribution. We start by providing a more in-depth definition of the k-means clustering problem, as well as presenting its most common implementation which will serve as a baseline. We then introduce the concept of the predictor and describe the algorithms with developed. In the end we report and analyze the results we obtained by testing our algorithms on various datasets, providing comparisons with the baselines both in terms of costs of the solutions and of computing time.

Chapter 1

Introduction and preliminaries

Generally speaking, we can say that "the field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience"[1]. In other words, machine learning algorithms are developed so that they have the the ability to learn and improve from the data they receive in input, while also trying to find an optimal solution to a given objective function that represents the goal to reach. In the past fifty years, many machine learning algorithms and approaches have been developed and used in many fields of human life, for example in medicine, but they are mostly employed in the fields of computer and data science.

Clustering is a popular task in machine learning and data analysis that is used to get an initial understanding of the dataset one is working with. A very common type of clustering, and also the one studied in this thesis, is the k-means problem, in which there is a set $P \subset \mathbb{R}^d$ of n points and the goal is to find a set $C \subset \mathbb{R}^d$ of centers of cardinality k such that it minimizes the objective cost:

$$cost(P, C) = \sum_{p \in P} \min_{c \in C} \quad (p - c)^2. \tag{1}$$

There also exist other types of clustering, which are quite different from the k-means one both in how they work and in how and where they are used. For example, hierarchical clustering is often used to group together similar objects in a tree-like structure, where the similarity is higher at the "roots" of the tree. Another example can be represented by density based clustering, which doesn't require the number of clusters to be specified in advance, and it can find clusters of arbitrary

shape, unlike k-means clustering which only produces spherical clusters.

One important feature of the clustering task is that it is unsupervised, meaning that clustering algorithms work with unlabeled data and learn patterns from them without any type of guidance. The reason behind this lies in the fact that the goal is to find an underlying structure or similarities of the data which are not known a priori. What we have just explained is a simplified definition of a machine learning technique called "unsupervised learning"[2].

1.1 Problem setting

In this thesis our main objective is to study the k-means problem on sets P of points that come from the same distribution $\mathcal{D}$ in a pseudo-online setting. We call it "pseudo-online" because, given a dataset, we divide it into smaller subsets and we can analyze only one subset at a time, unlike the standard online setting in which the input is processed one element at a time in a serial fashion (i.e. the points of a pointset). This division of a dataset into smaller subsets simulates the guarantee that the subsets come from a common distribution $\mathcal{D}$.

From now on then, we will treat a given dataset as a set $\mathcal{P}$ of subsets P^i such that $\mathcal{P} = \{P^1, P^2, ...\}$, all coming from $\mathcal{D}$, and at each time we can only analyze one subset P^i at a time.

In order to exploit the presence of an underlying distribution $\mathcal{D}$ we have developed algorithms that make also use of predictors trained on $\mathcal{P}$, which is in line with the framework of algorithms with predictions. However, before analyzing the aforementioned algorithms, we will now illustrate the standard algorithms used for the k-means problem.

1.2 The algorithms

One of the most used algorithms employed to solve the k-means problem is the one obtained from the combination of `k-means++` [3] followed by `Lloyd`'s algorithm.

`k-means++` is used to get an initial set of k centers from a set P by using a probability distribution created on P itself, as it can be seen in the pseudocode below:

Algorithm 1: `kmeans++`(P, k)

1 **Input:** Set $P \subset \mathbb{R}^d$ of n points; $k > 1 \in \mathbb{N}$
2 **Output:** Set $S \subset \mathbb{R}^d$ of k centers
3 $c_1 \leftarrow$ random point chosen from P with uniform probability;
4 $S \leftarrow \{c_1\}$;
5 **for** $2 \leq i \leq k$ **do**
6 **foreach** $x \in P \setminus S$ **do**
7 $\pi(x) \leftarrow \frac{(d(x,S))^2}{\sum_{y \in P \setminus S}(d(y,S))^2}$;
8 $c_i \leftarrow$ random point in $P \setminus S$ according to $\pi(\cdot)$;
9 $S \leftarrow S \cup \{c_i\}$;
10 return S;

For the sake of completeness and clarity, we define the distance between a point x and a set S, seen in line 7 in algorithm 1, in the following way:

$$d(x, S) = \min_{y \in S} d(x, y)$$

This definition will hold also for any of the following algorithms in this thesis.

`Lloyd`'s algorithm is used to minimize the objective cost function, and it is widely used thanks to its simplicity and the theoretical guarantees it possesses[4]. Given an initial set S of centers, it first assigns each point in P to its closest center.

It then computes the centroid of each cluster to obtain the new set of centers. Finally, it checks if the new objective cost (obtained using the centroid as centers) is lower than the previous one. It is run until it converges.

Algorithm 2: `Lloyd`(P, S, k)

Input: Set $P \subset \mathbb{R}^d$ of n points; Set $S \subset \mathbb{R}^d$ of k centers ; $k > 1 \in \mathbb{N}$
Output: Set S $\subset \mathbb{R}^d$ of k centers with small cost(P,S)
$S = \{c_1, c_2, ..., c_k\}$; `/* set of` k `centers */`
$\Phi \leftarrow cost(P, S)$;
while *true* **do**
    for $1 \leq i \leq k$ **do**
        $C_i \leftarrow$ points of C closest to c_i;
        $c_i \leftarrow \frac{1}{|C_i|} \sum_{x \in C_i} x$; `/* centroid of` C_i `*/`
    Let $S = \{c_1, c_2, ..., c_k\}$;
    if $cost(P, S) < \Phi$ **then**
        $\Phi \leftarrow cost(P, S)$;
    else
        return S;

Generally, these two algorithms are run in sequence and for a number r of times, and in the end the best solution is taken. For this reason, for the rest of this thesis, we are going to refer to these algorithms as if they were one by the name of `Km++L`. Also, if not differently specified, from now on we will always assume to run `Km++L` 20 times, and in the end to take the best solution among all of the 20.

1.3 The predictor

In the case of this thesis, we assume to have an initial set P^0 on which we apply `Km++L`, from which we obtain a set of centers S^0. The set P^0 is then discarded and a new set of points, P^1, is given, drawn from the same distribution of P^0. Because of this, it could be reasonable to apply set S^0 to P^1 to solve the k-means problem on P^1, although the effectiveness of this decision depends on sets P^0 and P^1. The

reason is that, since the points in these sets come from the same distribution $\mathcal{D}$, then set S^0 could be a good solution to P^1 as it would be an hypothetical set S^1 computed from P^1 using `Km++L`. The same reasoning can be applied to sets P^2, P^3 and so on. This is just an example that can be useful to start understanding how it is possible to use past informations (i.e. previous sets) to create predictions.

At this point, the predictor Π must be introduced. The predictor can be used in different ways, hence its definition is not unique. For example, a simple predictor could be the following one: given a set P^i, from the same distribution $\mathcal{D}$ of P^0, and the set S^0 described above, we could apply S^0 to P^i and compute $cost(P^i, S^0)$, also normalizing it by dividing it for the cardinality of P^i; we could call this measure nc_i. Then, indicating with nc_0 the normalized value of $cost(P^0, S^0)$, we could compare the two normalized costs to see if S^0 is a good solution for P^i.

Pseudocode example is provided below:

Algorithm 3: Predictor example 1

Input: Set $P^i \subset \mathbb{R}^d$ of n points; $k > 1 \in \mathbb{N}$; nc_0 normalized cost of a starting set P^0 from the same distribution of P^i
Output: Set $S^i \subset \mathbb{R}^d$ of k centers
$nc_i \leftarrow \frac{cost(P^i, S^0)}{|P^i|}$;
if $nc_i \approx nc_0$ **then**
    return S^0;
else
    $S^i \leftarrow$ `kmeans++(`P^i`,` k`)` ;
    return `Lloyd(`P^i`,` S^i`,` k`)`;

Another way to use the predictor is the following one: given a set P and a partial set of centers S' of cardinality $m < k$, we want Π to predict the next center. Pseudocode example is provided below. Also, in this second example, the matter of training the predictor Π is intentionally not dealt with, since it mostly depends

on the actual predictor.

Algorithm 4: Predictor example 2

Input: Set $P^i \subset \mathbb{R}^d$ of n points; $k > 1 \in \mathbb{N}$; Set $S' \subset \mathbb{R}^d$ of centers of cardinality $m < k$; Π predictor

Output: Set $S^i \subset \mathbb{R}^d$ of k centers

$c_{m+1} \leftarrow \Pi(P^{\mathrm{i}}, S^{'})$;

return $S' \cup \{c_{m+1}\}$;

1.4 Related work

Before illustrating the works that inspired the creation of this thesis it may be appropriate to first talk about the field of "algorithms with predictions".

The first time this name has been officially used was in the paper "Algorithms with predictions"[5] written by M. Mitzenmacher and S. Vassilvitskii in mid-2020. The main objective of their paper was to illustrate how combining the predicitions of a machine learning model with an algorithm may help to "circumvent worst-case analysis"[5]. In order to do that, they picked as examples some algorithms or class of problems and provided a general way to train and use a predictor, together with the algorithm, to make it more robust to worst-case scenarios. Their theoretical paper inspired researchers to actually experiment on real case scenarios and with many different algorithms in various settings, with the most common ones being offline, online and streaming. The interested reader can find a regularly updated collection of papers and works in the framework of "algorithms with predictions " at the following website: https://algorithms-with-predictions.github.io/.

Among these works, the closest one to our thesis is "Learning-Augmented k-means Clustering"[6] by Jon C. Ergun et al, even if there are some differences worth discussing. First of all, they start by assuming to have a good clustering, meaning that it has an associated relatively low cost of the objective function, and their objective is to improve that cost using a machine learning predictor. Moreover,

they train and use a predictor model to obtain the clustering labels of the elements of a given dataset. Also, the algorithm they used together with the predictor was developed around the error rate λ of the predictor itself, and the baselines they used were represented by `kmeans++` and a random sampling of the predictor labels. Finally, they used as predictors `Km++L` initialized once or a corrupted version of it or a neural network, depending on the dataset they worked on.

In the case of our thesis, instead, we assume to have subsets coming from a common distribution $\mathcal{D}$, and our goal is to exploit this information to help the clustering algorithm finding good centers for the subsequent subsets, while also improving the performance in terms of costs and of time with respect to a given baseline. Hence, our algorithms don't revolve around error parameters. Moreover, our baselines are represented by `Km++L` with two different initializations for `kmeans++`.

Chapter 2

Algorithms

In this section we are reporting the algorithms we designed along with a detailed description.

2.1 Algorithm 1

The main idea behind algorithm 1 is the following: given a set $\mathcal{P}$ of subsets P^i, defined as $\mathcal{P} = \{P^0, P^1, ..., P^l\}$, all coming from the same distribution $\mathcal{D}$, we initially perform `Km++L` on the first subset P^0, thus obtaining the set S^0 of k centers. Then, for each of the remaining $l-1$ subsets, we apply just one iteration of `Lloyd` using as initial set of centers the set S^0. The main reason behind this algorithm lies in the possibility of finding good centers for subsets $P^1, ..., P^l$ using only informations coming from the first subset, that is set S^0, while also limiting `Lloyd` to one iteration so to reduce computing time. It goes without saying that letting `Lloyd` converge would let us obtain better results in terms of cost of the objective function, however the analysis of the trade off between computing time and quality of the clustering is exactly what this algorithm was built for; in this case we chose to give more priority to the running time and slightly give up on the quality of the solution. However, for this algorithm we used a non-optimized version of `Lloyd`'s algorithm, hence the results that we will show later are relatively far from the best one can get.

2.2 Algorithm 2

Suppose to have a set $\mathcal{P}$ of subsets P^i, defined as $\mathcal{P} = \{P^0, P^1, ..., P^l\}$, all coming from the same distribution $\mathcal{D}$. The idea behind algorithm 2 is to apply `Km++L`

on the first subset, thus obtaining a set of centers referred to as S^0. Then S^0 is used as the starting set of centers for `Lloyd` applied on the other subsets. As a consequence, `Lloyd` is run only once. It is worth nothing to point out that the quality of the sets S^1, S^2, ... , S^l, where l is the number of subset coming from the same distribution that are being studied, depends on the set S^0 and on the compatibility between sets P^0 and P^i; this concept will be studied in (insert ref to analisi aggiuntiva). The reason behind this experiment lies in the assumption that the subsets all come from the same distribution, hence the distribution of the points in the different subsets may allow to use a single set of centers such as S^0 for all the subsets and obtain a good solution (i.e. a solution with a low cost). Of course, in order to understand whether or not the solutions obtained in this way are good we compared them to their baselines, as explained in the introduction of this section.

2.2.1 Variant

We also considered a variant of 2.2 consisting in using, at each iteration, the set of centers found by the predictor in the previous iteration. This means that instead of always using the set of centers S^0 as in 2.2 we use, in the i-th iteration, the set of centers found by `Lloyd` in the $(i-1)$-th iteration.

2.3 Algorithm 3

Assume to have a subset P, coming from a certain dataset, on which we want to perform the k-means analysis. We divide the subset P into two halves, P_1 and P_2, and apply `Km++L` on P_1, thus obtaining the set of centers S_1. We then use S_1 as the starting set of centers for `Lloyd` applied on P. This sequence of actions describes what our algorithm 2 does, which serves to study whether only part of a subset can be enough to find "good" centers to clusterize the entire subset; "good" means that the cost of the objective function is not too far from the cost one would obtain using `Km++L` instead of the algorithm in question. Although this experiment

may look similar to 2.2, the key difference here is that we run `Km++L` only on half subset, hence the computing time is shorter (even if for a relatively small amount) but the quality of the solution relatively to the entire subset may still be good. This idea can be explored even further, for example dividing P in more than two parts, hence reducing the computing time even more.

2.4 Algorithm 4

The idea is the following: given a set $\mathcal{P}$ of subsets P^i defined as $\mathcal{P} = \{P^0, P^1, ..., P^l\}$, all coming from the same distribution $\mathcal{D}$, we start by applying `Km++L` on the first of them, thus obtaining the set of centers S^0. Then, starting from the second subset, we run `Lloyd's algorithm` on the current subset i using as the initial set of centers a set of k centers obtained by randomly keeping k centers from the $i-1$ sets S of centers obtained in the previous iterations. To better explain this idea, assume to take subset P^0 from $\mathcal{P}$: we apply `Km++L` on it, thus obtaining the set of centers S^0. We then use S^0 as the starting set of centers for `Lloyd` on the next set, P^1, hence obtaining the set of centers S^1. At this point we have two set of centers and a total of $2 * k$ centers, from which we randomly choose k centers to be used for `Lloyd` applied on the third set, P^2. We then repeat the same actions performed before, meaning that from the $3 * k$ centers we only keep k of them, chosen randomly. This entire procedure is repeated on all the remaining subsets.

2.4.1 Variant

We also considered a variant of this idea which consists in performing, in each iteration i with $i \geq 2$, the `kmeans++` algorithm on the set of $i * k$ centers candidates and using the set of k centers returned by it. This variant could potentially yield a lower cost thanks to the fact that each of the $i * k$ centers candidate is assigned a probability to be chosen which depends on their distance from the set S of centers that `kmeans++` is creating, as seen in 1. In other words, this means that the probability of having in S two candidate centers near to each other is lower

with respect to the base algorithm described above.

2.5 Algorithm 5

With this algorithm we want to study another way of using informations of subsets coming from the same distribution $\mathcal{D}$, in particular we are now interested in the entire subsets as opposed to just the clustering centers. For this reason, given a set $\mathcal{P}$ of subsets P^i, defined as $\mathcal{P} = \{P^0, P^1, ..., P^l\}$, all coming from the same distribution $\mathcal{D}$, we initially perform `Km++L` on the first subset P^0, thus obtaining the set S^0 of k centers. We then apply `Lloyd` to subset P^1 using S^0 as initial set of centers, obtaining set S^1. Now, for subsets P^i, where $3 \leq i \leq l$, what we do is merge all the previous subsets P^j together, where $0 \leq j < i$, thus obtaining set P^i_{merged}. At this point we apply `Km++L` on P^i_{merged}, thus obtaining the set S^i_{merged} of k centers, which will be used as initial set of centers for `Lloyd` applied to P^i. This procedure is repeated for every i, so for example in iteration $i + 1$ we will have set P^{i+1}_{merged}, created by merging together subsets P^j, but where now it is $0 \leq j < i + 1$. The reason why previous subsets may help finding good centers for the next subsets lies in the fact that a single subset is just one sample of the underlying distribution $\mathcal{D}$, thus merging more samples may lead to a better sample of the distribution.

2.6 Algorithm 6

The idea behind algorithm 6 is the following: assume to have a set $\mathcal{P}$ of subsets P^i, defined as $\mathcal{P} = \{P^0, P^1, ..., P^l\}$, all coming from the same distribution $\mathcal{D}$. We first perform `Km++L` on subset P^0, thus obtaining set S^0 of k centers. Then we perform `kmeans++` on the next subset, hence obtaining set S^i_{++} of k centers. We then merge sets S^0 and S^i_{++}, and at this point we perform `kmeans++` on the merged set, obtaining set $S^i_{merged++}$. Now we use `Lloyd` on set P^i using $S^i_{merged++}$ as the initial set of centers, thus obtaining set S^i. We repeat these steps for all the other subsets, but each time substituting S^0 with the set S^i obtained in the

previous iteration. As it can be seen, at each iteration we are trying to use both the informations coming from the current subset, by appyling `kmeans++` on it and obtaining a first set of candidate centers, and the informations of the previous subset, by using its centers. This potentially allows the predictor to yield better results on the current subset, because with some probability the previous subset is more similar to the current one with respect to all the other subsets before it. Also, by applying `kmeans++` on the merged set what we are doing is essentially trying to get centers as sparse and fit as possible, with respect to the subset in question. The reason why we should get centers with these characteristics lies in the `kmeans++` algorithm, and in particular in its way of picking centers, that is using a probability distribution that depends on the distance of the set candidate centers to the set S of chosen centers; also remember that this probability is higher for candidate centers that are further from S.

Chapter 3

Experimental setup and implementation

In this section we are going to report the datasets on which we tested our algorithms and analyze the results we obtained, while also making a comparison with the `Km++L` algorithm, which is used as a baseline. Moreover, we tested the datasets with four values of k, specifically for $k = \{10, 15, 20, 25\}$. This gives us an idea on how the costs and the computation times vary as the hyperparameter k changes.

3.1 Baselines

For each dataset we used two baselines for the cost of the objective function and three for the computation time. Regarding the costs, the first baseline is represented by `Km++L` run 20 times, while the second one is represented by `Km++L` run only once. The reason behind these choices is to see how our algorithms performed against the two baselines, each representing a different quality of the objective function cost.

Two of the three baselines used for the computation time are represented by `Km++L` run 20 times and `Km++L` run once, while the third one is represented by an algorithm, which we will refer to as `Ideal`, which makes use of an ideal and unfeasible oracle. Given as input a set P, the oracle returns in time $\mathcal{O}(1)$ the set S of centers that one would obtain running `Km++L` 20 times on P. The algorithm then uses S as the starting set of centers for `Lloyd` applied on P. We chose to use this

baseline because it allows us to compare the running times of our algorithms with the quickest possible algorithm which makes use of an oracle.

3.2 Datasets

We tested our ideas using seven different datasets, one taken from the *SNAP library*[7], three from *kaggle.com*, while the remaining three have been generated using *scikit-learn*. Informations regarding the datasets are reported below:

3.2.1 Synthetic datasets

We created the synthetic datasets using the `make_blobs` function from the *scikit-learn* package with the following shared parameters: "centers" = None, "random_state"= 0. The values of other two remaining parameters are reported in the table below, along with other specific informations regarding the datasets.

	Synth20K	Synth100K	Synth2M
Cardinality	$2 \cdot 10^4$	$1 \cdot 10^5$	$2 \cdot 10^6$
Number of features	5	5	5
Number of subsets	2	2	2
"cluster_std"	7	7	7
"n_samples"	$(3.5 \cdot 10^3, 6 \cdot 10^3, 2.5 \cdot 10^3, 8 \cdot 10^4)$	$(1.75 \cdot 10^4, 3 \cdot 10^4, 1.25 \cdot 10^4, 4 \cdot 10^4)$	$(3.5 \cdot 10^5, 6 \cdot 10^5, 2.5 \cdot 10^5, 8 \cdot 10^5)$
"center_box"	$(-35.0, 35.0)$	$(-35.0, 35.0)$	$(-35.0, 35.0)$

Table 1: Synthetic datasets informations

3.2.2 Real datasets

Informations regarding the four real datasets are reported below.

1. **Oregon**[1]: Dataset of 9 graph snapshots sampled across 3 months from an internet router communication network[8][7]. As done in previous work[6], we used the top two eigenvectors of the normalized Laplacian matrix to obtain node embeddings in $\mathbb{R}^2$ for each graph, which gives us 9 subsets. Each subset has $n \sim 10^4$ points.

2. **Bank**[2]: Dataset of roughly 78 thousand elements and 21 attributes; 2 out of 21 attributes were removed since they contain text informations. The dataset was shuffled and divided into 7 subsets of equal size. In this way we were able to make the original dataset fit the setting of our work.

3. **Power**[3]: Dataset of roughly 53 thousand elements and 9 attributes; 1 out of 9 attributes were removed since they contain text informations. The dataset was shuffled and divided into 7 subsets of equal size. In this way we were able to make the original dataset fit the setting of our work.

4. **GPS-ephemeris**[4]: Dataset of roughly 308 thousand elements and 15 attributes, where 2 of them were removed since they contained text informations. The dataset was divided into 8 subsets of equal length. In this way we were able to make the original dataset fit the setting of our work.

3.3 Implementation

In this subsection we report the informations required to replicate the tests we made, that is what functions and packages we used as well as the parameters we chose. All the code has been written in Python. Informations regarding specifics of the machine on which tests were run can be found here as well. Functions written

[1] https://snap.stanford.edu/data/Oregon-2.html
[2] https://www.kaggle.com/datasets/utkarshx27/american-companies-bankruptcy-prediction-dataset
[3] https://www.kaggle.com/datasets/fedesoriano/electric-power-consumption
[4] https://www.kaggle.com/datasets/georgyzubkov/gps-ephemeristemporal-information

	Oregon	Bank	Power	GPS-ephemeris
Source	*SNAP library*[7]	*kaggle.com*	*kaggle.com*	*kaggle.com*
Cardinality	$\sim 9 \cdot 10^4$	$\sim 8 \cdot 10^4$	$\sim 5 \cdot 10^4$	$\sim 3 \cdot 10^5$
Number of features	2	19	8	13
Number of subsets	9	7	7	8

Table 2: Real datasets informations

by us will be reported in the Appendix section at the end of the thesis.

3.3.1 Machine and specifics

The code as well as the tests has been written and run on the default configuration of Google Colab. This configuration provides an Intel Xeon CPU with 2 vCPUs and 13GB of RAM, and a Tesla K80 GPU with 12GB of RAM.

3.3.2 Functions and packages

Datasets were read using *pandas* with the `read_csv` method, since they mostly were .csv files, and any row with missing values was dropped with `dropna`. The only exception was Oregon: this dataset was originally divided into 9 distinct .txt files, each of them containing graph informations that were retrieved using *networkx* with the `read_edgelist` method.

The shuffling of the datasets has been achieved using the `train_test_split` method from *sklearn*; regarding the parameters, we used "test_size"= 0.5 and "random_state"= 1699. Moreover, from this package we also used the `KMeans` and `kmeans_plusplus` functions, which respectively represent `Km++L` and `kmeans++`. For both of them the parameter "random_state" was set to 0.

Chapter 4

Results on synthetic datasets

This subsection contains the experimental results we obtained from testing our algorithms on the synthetic datasets described in the "Datasets" section. We tested the synthetic datasets with one value of k, specifically for $k = 10$, and we chose to focus on trying to understand how the performance of our algorithms change when the cardinality of the dataset, and consequently of the subsets, increases. It is worth noting that only the second subset has been reported, and not the first one: the reason is that the first subset is often only used to obtain an initial set of centers by means of `Km++L`. Actually, algorithm 2 is the only exception to this rule, since by design the first subset is treated as any other subset coming from the same dataset. Nonetheless the first subset and its results were not reported in order to make the results of algorithm 2 comparable with those of the other algorithms.

	Synth20K	
	cost	time
`Km++L` 20 init	$1.935 \cdot 10^6$	2.2971s
`Km++L` 1 init	$1.957 \cdot 10^6$	0.1458s
`Algorithm 1`	$1.952 \cdot 10^6$	1.5630s
`Algorithm 2`	$1.924 \cdot 10^6$	0.0910s
`Algorithm 2v`	$1.924 \cdot 10^6$	0.0275s
`Algorithm 3`	$1.929 \cdot 10^6$	2.1030s
`Algorithm 4`	$1.924 \cdot 10^6$	0.0353s
`Algorithm 4v`	$1.924 \cdot 10^6$	0.0275s
`Algorithm 5`	$1.924 \cdot 10^6$	0.0293s
`Algorithm 6`	$1.951 \cdot 10^6$	0.1304s

Table 3: Synth20K, $k = 10$, all algorithms

	Synth100K	
	cost	time
`Km++L` 20 init	$9.676 \cdot 10^6$	2.0481s
`Km++L` 1 init	$9.979 \cdot 10^6$	0.1872s
`Algorithm 1`	$9.730 \cdot 10^6$	8.6073s
`Algorithm 2`	$9.680 \cdot 10^6$	0.1261s
`Algorithm 2v`	$9.680 \cdot 10^6$	0.1280s
`Algorithm 3`	$9.689 \cdot 10^6$	2.9602s
`Algorithm 4`	$9.680 \cdot 10^6$	0.1098s
`Algorithm 4v`	$9.680 \cdot 10^6$	0.1098s
`Algorithm 5`	$9.680 \cdot 10^6$	0.1149s
`Algorithm 6`	$9.991 \cdot 10^6$	0.2656s

Table 4: Synth100K, $k = 10$, all algorithms

	Synth2M	
	cost	time
`Km++L` 20 init	$1.951 \cdot 10^8$	34.091s
`Km++L` 1 init	$1.951 \cdot 10^8$	2.7447s
`Algorithm 1`	$1.951 \cdot 10^8$	135.03s
`Algorithm 2`	$1.951 \cdot 10^8$	0.3452s
`Algorithm 2v`	$1.951 \cdot 10^8$	0.3476s
`Algorithm 3`	$1.951 \cdot 10^8$	15.056s
`Algorithm 4`	$1.951 \cdot 10^8$	0.3367s
`Algorithm 4v`	$1.951 \cdot 10^8$	0.3430s
`Algorithm 5`	$1.951 \cdot 10^8$	0.3411s
`Algorithm 6`	$1.980 \cdot 10^8$	1.8176s

Table 5: Synth2M, $k = 10$, all algorithms

4.1 Analysis

We gathered and plotted here the results reported in the tables, hence obtaining a better understanding on how each algorithm performed. As written previously, tests on the synthetic datasets have been done with only one value of k, specifically for $k = 10$.

In the legend present in every plot we referred to the variants of algorithms 2 and 4 as "2v" and "4v" for brevity, and for the same reason we referred to the two baselines as "Km++L 20 init" and "Km++L 1 init", meaning that they have been initialized 20 times and 1 time respectively.

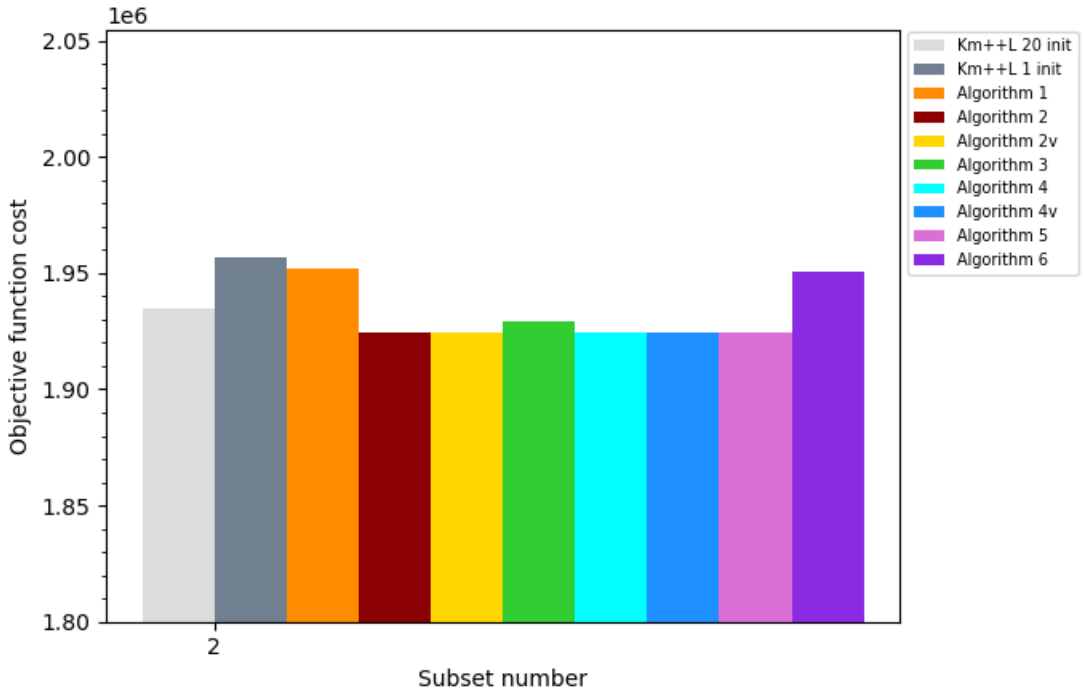

Figure 1: Synth20K, $k = 10$, objective function cost

Figure 1 shows the objective function cost obtained by each algorithm on Synth20K for $k = 10$. Except for algorithms 1 and 6, all the others obtained lower costs than those of the baselines. This result has a great importance as it tells us that for subsets actually coming from the same distribution our algorithms are very competitive compared to `Km++L`.

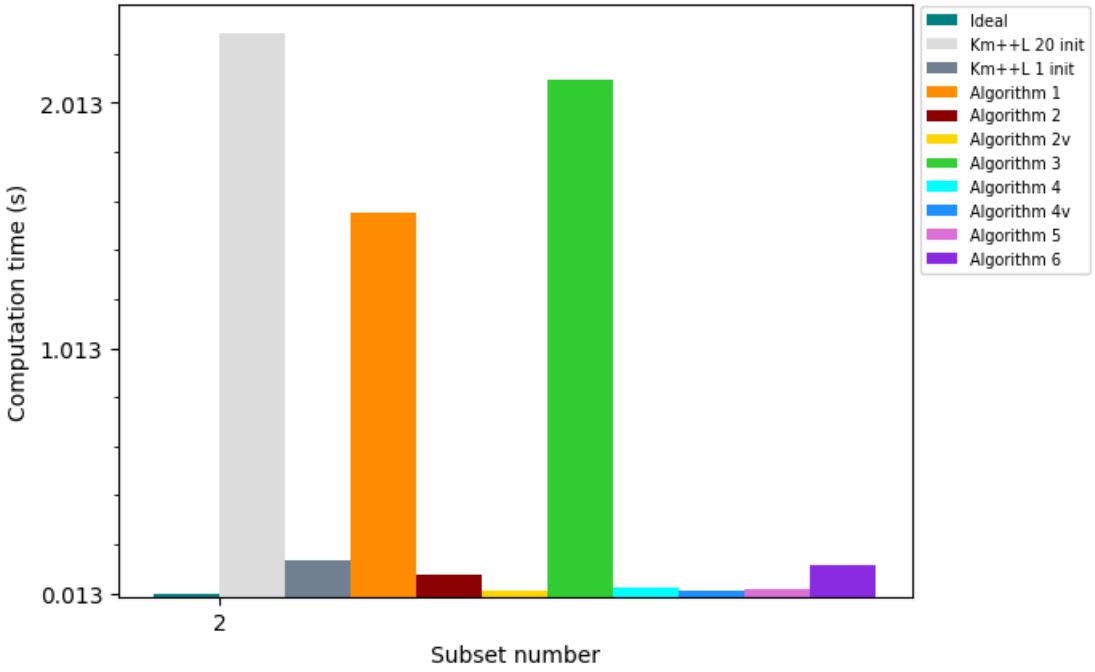

Figure 2: Synth20K, $k = 10$, computation time

Figure 2 shows the running time of each algorithm with respect to the analyzed subset of Synth20K, for $k = 10$. Except for algorithms 1 and 3, the others are faster than the `Km++L` 1 init baseline, and some are even near the `Ideal` one, in particular algorithms 2v and 4v. Apart from this remarkable result, what is interesting is that the majority of these faster algorithms also obtained better results in terms of cost compared to the `Km++L` 20 init baseline.

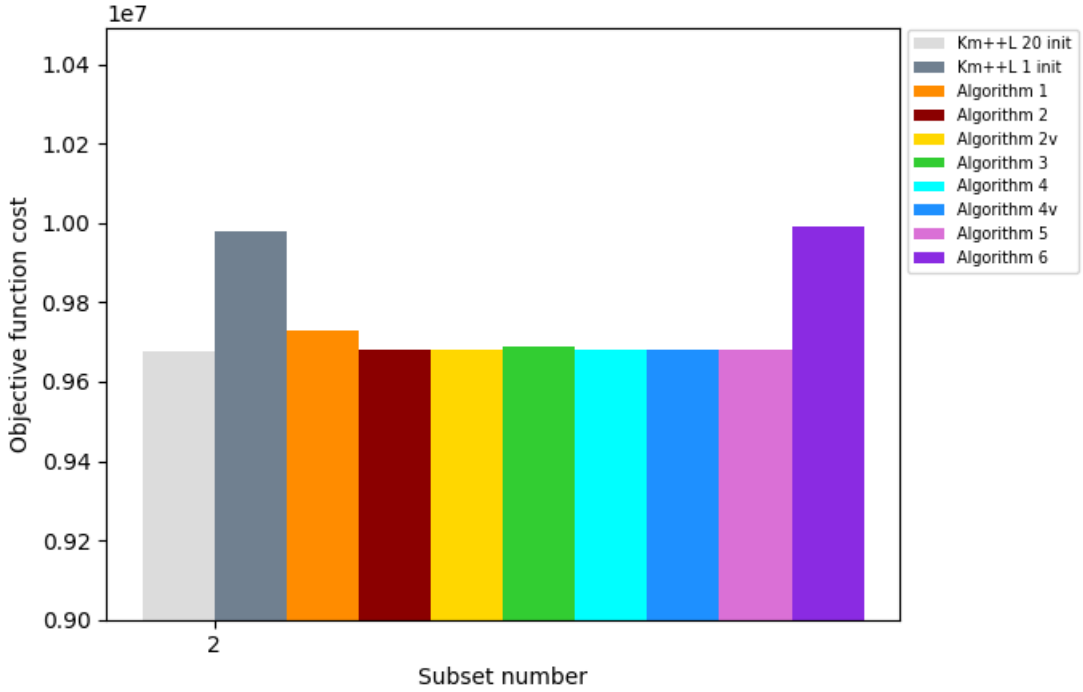

Figure 3: Synth100K, $k = 10$, objective function cost

Figure 3 shows the objective function cost obtained by each algorithm on Synth100K for $k = 10$. Unlike what we have seen in Synth20K, depicted in figure 1, here the majority of our algorithms obtained costs equal to the `Km++L` 20 init baseline.

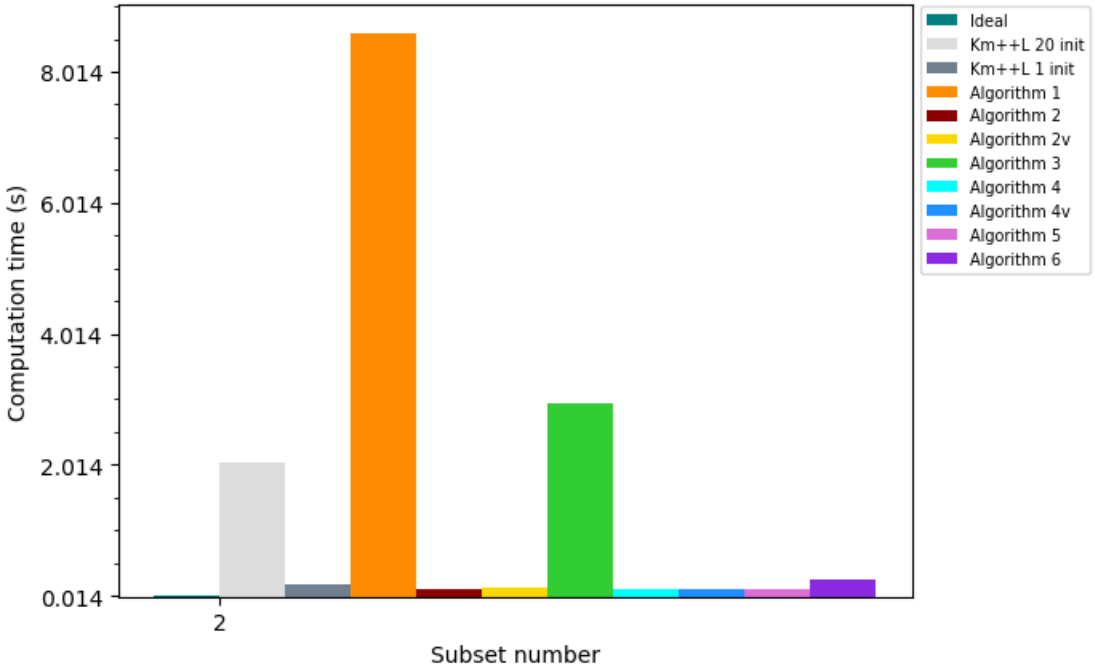

Figure 4: Synth100K, $k = 10$, computation time

Figure 4 shows the running time of each algorithm with respect to the analyzed subset of Synth100K, for $k = 10$. The results are quite similar to the ones found for Synth20k, depicted in figure 2, with the main difference that here algorithms 1 and 3 are slower than the `Km++L` 20 init baseline.

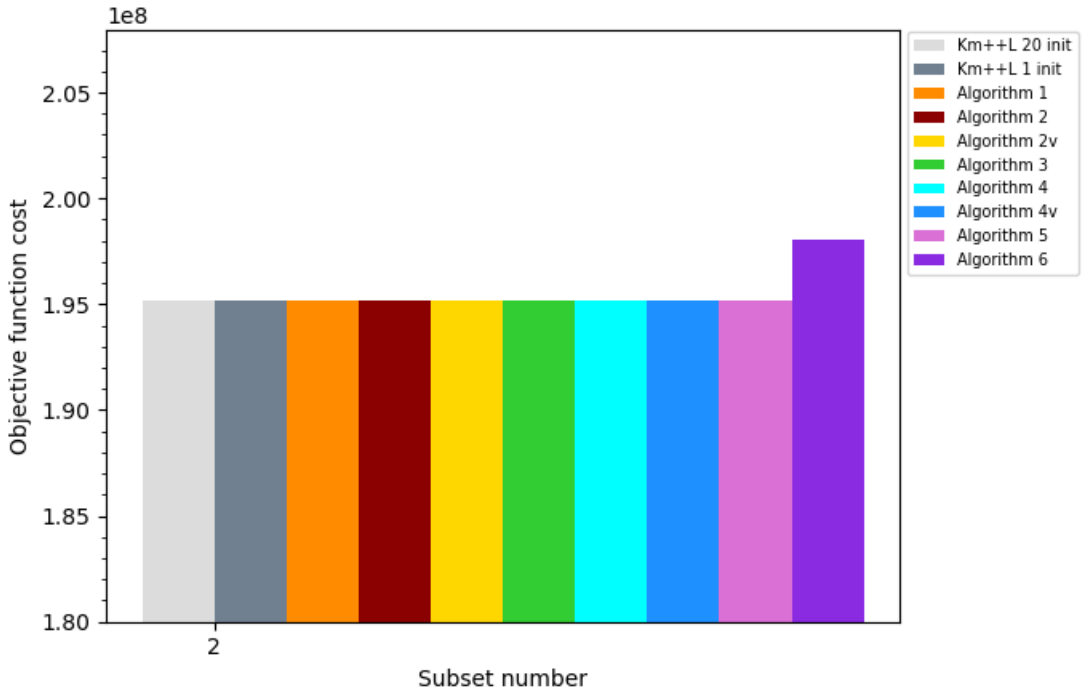

Figure 5: Synth2M, $k = 10$, objective function cost

Figure 5 shows the objective function cost obtained by each algorithm on Synth2M for $k = 10$. In this case the cost found from every algorithm, except for algorithm 6, is very close to those found from the others, as it can be easily observed both from the figure.

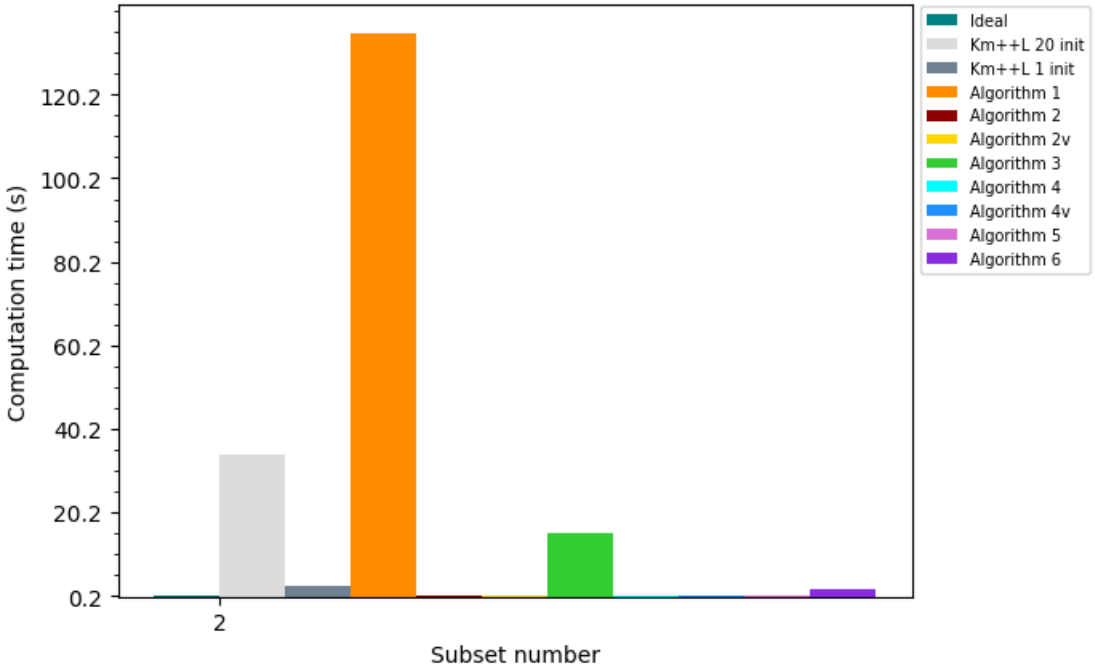

Figure 6: Synth2M, $k = 10$, computing time

Figure 6 shows the running time of each algorithm with respect to the analyzed subset of Synth2M, for $k = 10$. Results are very similar to the ones found for Synth100K, depicted in figure 4, with the only difference that here algorithm 3 is faster than the `Km++L` 20 init baseline.

Chapter 5

Results on real datasets

This subsection contains the experimental results we obtained from testing our algorithms on the real datasets described in the "Datasets" section. The results are divided by dataset. For each algorithm we compiled a table reporting the results obtained for the four values of the hyperparameter k, that is for $k = 10, 15, 20, 25$. In each table both the cost of the objective function and the computing time were reported.

For each value of k there are four associated columns. One column, called "Sub", contains the subsets, expressed as numbers, of a given dataset. As specified for the synthetic datasets, the results associated with the first subset of each dataset are not reported since it is used only to obtain an initial set S of centers by means of `Km++L`. The remaining three columns are called respectively "Algorithm #", "`Km++L` 20 init" and "`Km++L` 1 init, and each one of them is split in two parts, one reporting the cost of the objective function and the other one the computing time; the symbol # is a placeholder for the number of the algorithm of which the table is reporting the results. The last two columns just mentioned refer to the baselines of `Km++L` run 20 times and 1 time respectively.

5.1 Oregon

We report here the results obtained on the Oregon dataset.

k	Sub	Algorithm 1		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.609 \cdot 10^{-1}$	1.7537s	$6.373 \cdot 10^{-1}$	1.1084s	$6.475 \cdot 10^{-1}$	0.1278s
	3	$5.961 \cdot 10^{-1}$	3.0108s	$3.945 \cdot 10^{-1}$	1.3932s	$4.107 \cdot 10^{-1}$	0.0105s
	4	1.064	1.6165s	$6.149 \cdot 10^{-1}$	1.2551s	$6.259 \cdot 10^{-1}$	0.0567s
	5	$2.304 \cdot 10^{-1}$	2.2785s	$1.835 \cdot 10^{-1}$	2.1173s	$1.890 \cdot 10^{-1}$	0.0112s
	6	2.528	1.4503s	$8.515 \cdot 10^{-1}$	0.8624s	$9.861 \cdot 10^{-1}$	0.0139s
	7	$4.647 \cdot 10^{-1}$	1.3639s	$2.794 \cdot 10^{-1}$	1.6159s	$2.972 \cdot 10^{-1}$	0.0139s
	8	$7.152 \cdot 10^{-1}$	1.4735s	$4.079 \cdot 10^{-1}$	1.1135s	$4.207 \cdot 10^{-1}$	0.0137s
	9	1.127	1.4458s	$5.912 \cdot 10^{-1}$	1.6629s	$6.370 \cdot 10^{-1}$	0.1272s
15	2	$5.365 \cdot 10^{-1}$	2.2150s	$3.983 \cdot 10^{-1}$	1.7552s	$4.436 \cdot 10^{-1}$	0.2565s
	3	$3.564 \cdot 10^{-1}$	1.8377s	$2.246 \cdot 10^{-1}$	1.1674s	$2.532 \cdot 10^{-1}$	0.3123s
	4	$5.826 \cdot 10^{-1}$	2.7093s	$3.784 \cdot 10^{-1}$	1.2347s	$3.800 \cdot 10^{-1}$	0.1194s
	5	$1.379 \cdot 10^{-1}$	1.9821s	$1.116 \cdot 10^{-1}$	1.3484s	$1.251 \cdot 10^{-1}$	0.0996s
	6	1.208	1.9497s	$5.390 \cdot 10^{-1}$	1.1953s	$5.557 \cdot 10^{-1}$	0.3869s
	7	$3.654 \cdot 10^{-1}$	1.9627s	$1.835 \cdot 10^{-1}$	1.4577s	$1.863 \cdot 10^{-1}$	0.0968s
	8	$4.552 \cdot 10^{-1}$	1.8327s	$2.634 \cdot 10^{-1}$	1.1931s	$2.773 \cdot 10^{-1}$	0.0955s
	9	$6.791 \cdot 10^{-1}$	2.3958s	$3.976 \cdot 10^{-1}$	1.5465s	$4.305 \cdot 10^{-1}$	0.0621s
20	2	$4.579 \cdot 10^{-1}$	3.0655s	$2.728 \cdot 10^{-1}$	1.2052s	$3.039 \cdot 10^{-1}$	0.0194s
	3	$2.553 \cdot 10^{-1}$	2.6742s	$1.558 \cdot 10^{-1}$	1.6074s	$1.684 \cdot 10^{-1}$	0.0177s
	4	$5.193 \cdot 10^{-1}$	2.6529s	$2.371 \cdot 10^{-1}$	1.8631s	$2.722 \cdot 10^{-1}$	0.0947s
	5	$1.054 \cdot 10^{-1}$	3.6979s	$7.898 \cdot 10^{-2}$	1.7600s	$8.107 \cdot 10^{-2}$	0.1368s
	6	1.083	2.8244s	$3.486 \cdot 10^{-1}$	1.3124s	$3.676 \cdot 10^{-1}$	0.1194s
	7	$2.315 \cdot 10^{-1}$	2.6637s	$1.330 \cdot 10^{-1}$	2.3199s	$1.342 \cdot 10^{-1}$	0.1314s
	8	$3.186 \cdot 10^{-1}$	2.7061s	$1.792 \cdot 10^{-1}$	1.4710s	$1.884 \cdot 10^{-1}$	0.0605s
	9	$5.702 \cdot 10^{-1}$	3.7026s	$2.811 \cdot 10^{-1}$	1.3550s	$3.204 \cdot 10^{-1}$	0.0211s
25	2	$4.294 \cdot 10^{-1}$	4.6736s	$1.907 \cdot 10^{-1}$	1.2929s	$2.133 \cdot 10^{-1}$	0.0254s
	3	$2.012 \cdot 10^{-1}$	3.1121s	$1.159 \cdot 10^{-1}$	1.0632s	$1.264 \cdot 10^{-1}$	0.0319s
	4	$4.563 \cdot 10^{-1}$	3.2326s	$1.853 \cdot 10^{-1}$	0.6241s	$1.859 \cdot 10^{-1}$	0.0231s
	5	$7.700 \cdot 10^{-2}$	4.1719s	$5.740 \cdot 10^{-2}$	2.0785s	$5.874 \cdot 10^{-2}$	0.1239s
	6	$7.473 \cdot 10^{-1}$	3.5863s	$2.331 \cdot 10^{-1}$	1.4850s	$2.728 \cdot 10^{-1}$	0.0305s
	7	$1.812 \cdot 10^{-1}$	3.2358s	$9.522 \cdot 10^{-2}$	2.5558s	$9.930 \cdot 10^{-2}$	0.0281s
	8	$2.484 \cdot 10^{-1}$	4.0138s	$1.379 \cdot 10^{-1}$	1.3089s	$1.434 \cdot 10^{-1}$	0.0320s
	9	$4.857 \cdot 10^{-1}$	3.5378s	$2.045 \cdot 10^{-1}$	1.5602s	$2.045 \cdot 10^{-1}$	0.0520s

Table 6: Oregon, algorithm 1

k	Sub	Algorithm 2		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$8.510 \cdot 10^{-1}$	0.0124s	$6.373 \cdot 10^{-1}$	1.3253s	$6.475 \cdot 10^{-1}$	0.0665s
	3	$3.910 \cdot 10^{-1}$	0.3387s	$3.945 \cdot 10^{-1}$	1.1368s	$4.107 \cdot 10^{-1}$	0.0388s
	4	$7.799 \cdot 10^{-1}$	0.2016s	$6.149 \cdot 10^{-1}$	1.4302s	$6.259 \cdot 10^{-1}$	0.0599s
	5	$1.888 \cdot 10^{-1}$	0.2794s	$1.835 \cdot 10^{-1}$	1.1800s	$1.890 \cdot 10^{-1}$	0.0677s
	6	1.301	0.2140s	$8.515 \cdot 10^{-1}$	1.2604s	$9.861 \cdot 10^{-1}$	0.0161s
	7	$3.043 \cdot 10^{-1}$	0.0117s	$2.794 \cdot 10^{-1}$	1.5849s	$2.972 \cdot 10^{-1}$	0.0129s
	8	$5.063 \cdot 10^{-1}$	0.0107s	$4.079 \cdot 10^{-1}$	1.3192s	$4.207 \cdot 10^{-1}$	0.0143s
	9	$7.428 \cdot 10^{-1}$	0.0235s	$5.912 \cdot 10^{-1}$	2.0727s	$6.370 \cdot 10^{-1}$	0.0130s
15	2	$4.379 \cdot 10^{-1}$	0.0266s	$3.983 \cdot 10^{-1}$	1.2724s	$4.436 \cdot 10^{-1}$	0.0212s
	3	$2.845 \cdot 10^{-1}$	0.0091s	$2.246 \cdot 10^{-1}$	1.6342s	$2.532 \cdot 10^{-1}$	0.1236s
	4	$4.443 \cdot 10^{-1}$	0.0089s	$3.784 \cdot 10^{-1}$	2.5654s	$3.800 \cdot 10^{-1}$	0.0169s
	5	$1.136 \cdot 10^{-1}$	0.0086s	$1.116 \cdot 10^{-1}$	1.6397s	$1.251 \cdot 10^{-1}$	0.0152s
	6	$9.256 \cdot 10^{-1}$	0.0078s	$5.390 \cdot 10^{-1}$	1.3937s	$5.557 \cdot 10^{-1}$	0.0236s
	7	$2.363 \cdot 10^{-1}$	0.0081s	$1.835 \cdot 10^{-1}$	1.5191s	$1.863 \cdot 10^{-1}$	0.0545s
	8	$3.163 \cdot 10^{-1}$	0.0112s	$2.634 \cdot 10^{-1}$	1.4302s	$2.773 \cdot 10^{-1}$	0.0556s
	9	$5.024 \cdot 10^{-1}$	0.0088s	$3.976 \cdot 10^{-1}$	1.1470s	$4.305 \cdot 10^{-1}$	0.0611s
20	2	$3.500 \cdot 10^{-1}$	0.0124s	$2.728 \cdot 10^{-1}$	1.6616s	$3.039 \cdot 10^{-1}$	0.1297s
	3	$1.619 \cdot 10^{-1}$	0.0310s	$1.558 \cdot 10^{-1}$	1.3606s	$1.684 \cdot 10^{-1}$	0.1253s
	4	$2.787 \cdot 10^{-1}$	0.0128s	$2.371 \cdot 10^{-1}$	1.6995s	$2.722 \cdot 10^{-1}$	0.1114s
	5	$8.573 \cdot 10^{-2}$	0.0108s	$7.898 \cdot 10^{-2}$	1.7768s	$8.107 \cdot 10^{-2}$	0.1323s
	6	$5.663 \cdot 10^{-1}$	0.0196s	$3.486 \cdot 10^{-1}$	1.4974s	$3.676 \cdot 10^{-1}$	0.1137s
	7	$1.350 \cdot 10^{-1}$	0.0126s	$1.330 \cdot 10^{-1}$	2.5613s	$1.342 \cdot 10^{-1}$	0.1243s
	8	$2.149 \cdot 10^{-1}$	0.0134s	$1.792 \cdot 10^{-1}$	1.7372s	$1.884 \cdot 10^{-1}$	0.1249s
	9	$3.517 \cdot 10^{-1}$	0.0109s	$2.811 \cdot 10^{-1}$	0.6214s	$3.204 \cdot 10^{-1}$	0.0216s
25	2	$2.347 \cdot 10^{-1}$	0.0243s	$1.907 \cdot 10^{-1}$	1.3191s	$2.133 \cdot 10^{-1}$	0.1237s
	3	$1.271 \cdot 10^{-1}$	0.0089s	$1.159 \cdot 10^{-1}$	1.0428s	$1.264 \cdot 10^{-1}$	0.0235s
	4	$2.245 \cdot 10^{-1}$	0.0193s	$1.853 \cdot 10^{-1}$	1.4846s	$1.859 \cdot 10^{-1}$	0.0252s
	5	$6.114 \cdot 10^{-2}$	0.0094s	$5.740 \cdot 10^{-2}$	2.3709s	$5.874 \cdot 10^{-2}$	0.1320s
	6	$4.102 \cdot 10^{-1}$	0.0160s	$2.331 \cdot 10^{-1}$	0.5797s	$2.728 \cdot 10^{-1}$	0.0220s
	7	$1.077 \cdot 10^{-1}$	0.0188s	$9.522 \cdot 10^{-2}$	1.3127s	$9.930 \cdot 10^{-2}$	0.0264s
	8	$1.682 \cdot 10^{-1}$	0.0163s	$1.379 \cdot 10^{-1}$	1.2492s	$1.434 \cdot 10^{-1}$	0.1036s
	9	$2.533 \cdot 10^{-1}$	0.0112s	$2.045 \cdot 10^{-1}$	0.8875s	$2.045 \cdot 10^{-1}$	0.1279s

Table 7: Oregon, algorithm 2

k	Sub	Algorithm 2v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$8.510 \cdot 10^{-1}$	0.0074s	$6.373 \cdot 10^{-1}$	1.4931s	$6.475 \cdot 10^{-1}$	0.0493s
	3	$6.909 \cdot 10^{-1}$	0.0106s	$3.945 \cdot 10^{-1}$	1.5416s	$4.107 \cdot 10^{-1}$	0.0909s
	4	$7.799 \cdot 10^{-1}$	0.0069s	$6.149 \cdot 10^{-1}$	1.4871s	$6.259 \cdot 10^{-1}$	0.0631s
	5	$2.066 \cdot 10^{-1}$	0.0068s	$1.835 \cdot 10^{-1}$	1.2149s	$1.890 \cdot 10^{-1}$	0.0155s
	6	1.062	0.0072s	$8.515 \cdot 10^{-1}$	1.3653s	$9.861 \cdot 10^{-1}$	0.0150s
	7	$3.396 \cdot 10^{-1}$	0.0144s	$2.794 \cdot 10^{-1}$	1.5522s	$2.972 \cdot 10^{-1}$	0.0131s
	8	$4.546 \cdot 10^{-1}$	0.0103s	$4.079 \cdot 10^{-1}$	1.5860s	$4.207 \cdot 10^{-1}$	0.1136s
	9	$6.867 \cdot 10^{-1}$	0.0063s	$5.912 \cdot 10^{-1}$	1.5404s	$6.370 \cdot 10^{-1}$	0.0654s
15	2	$4.379 \cdot 10^{-1}$	0.1385s	$3.983 \cdot 10^{-1}$	1.3591s	$4.436 \cdot 10^{-1}$	0.0166s
	3	$2.785 \cdot 10^{-1}$	0.2396s	$2.246 \cdot 10^{-1}$	1.0938s	$2.532 \cdot 10^{-1}$	0.0203s
	4	$4.817 \cdot 10^{-1}$	0.4095s	$3.784 \cdot 10^{-1}$	1.6421s	$3.800 \cdot 10^{-1}$	0.0214s
	5	$1.252 \cdot 10^{-1}$	0.3860s	$1.116 \cdot 10^{-1}$	1.3186s	$1.251 \cdot 10^{-1}$	0.0163s
	6	$7.769 \cdot 10^{-1}$	0.3033s	$5.390 \cdot 10^{-1}$	1.2997s	$5.557 \cdot 10^{-1}$	0.0255s
	7	$2.432 \cdot 10^{-1}$	0.0123s	$1.835 \cdot 10^{-1}$	1.4732s	$1.863 \cdot 10^{-1}$	0.1206s
	8	$3.084 \cdot 10^{-1}$	0.0092s	$2.634 \cdot 10^{-1}$	1.4300s	$2.773 \cdot 10^{-1}$	0.0685s
	9	$4.432 \cdot 10^{-1}$	0.0081s	$3.976 \cdot 10^{-1}$	1.9207s	$4.305 \cdot 10^{-1}$	0.1253s
20	2	$3.500 \cdot 10^{-1}$	0.0361s	$2.728 \cdot 10^{-1}$	1.3548s	$3.039 \cdot 10^{-1}$	0.0209s
	3	$2.027 \cdot 10^{-1}$	0.1041s	$1.558 \cdot 10^{-1}$	1.2772s	$1.684 \cdot 10^{-1}$	0.0219s
	4	$3.198 \cdot 10^{-1}$	0.0105s	$2.371 \cdot 10^{-1}$	1.4819s	$2.722 \cdot 10^{-1}$	0.0212s
	5	$8.715 \cdot 10^{-2}$	0.0180s	$7.898 \cdot 10^{-2}$	2.7196s	$8.107 \cdot 10^{-2}$	0.0324s
	6	$6.513 \cdot 10^{-1}$	0.0090s	$3.486 \cdot 10^{-1}$	1.5382s	$3.676 \cdot 10^{-1}$	0.0197s
	7	$1.466 \cdot 10^{-1}$	0.0206s	$1.330 \cdot 10^{-1}$	1.4577s	$1.342 \cdot 10^{-1}$	0.1243s
	8	$2.371 \cdot 10^{-1}$	0.0118s	$1.792 \cdot 10^{-1}$	0.9577s	$1.884 \cdot 10^{-1}$	0.1302s
	9	$3.378 \cdot 10^{-1}$	0.0089s	$2.811 \cdot 10^{-1}$	1.4191s	$3.204 \cdot 10^{-1}$	0.0192s
25	2	$2.347 \cdot 10^{-1}$	0.0138s	$1.907 \cdot 10^{-1}$	1.6061s	$2.133 \cdot 10^{-1}$	0.1139s
	3	$1.423 \cdot 10^{-1}$	0.0886s	$1.159 \cdot 10^{-1}$	2.1408s	$1.264 \cdot 10^{-1}$	0.1271s
	4	$2.099 \cdot 10^{-1}$	0.0190s	$1.853 \cdot 10^{-1}$	1.1257s	$1.859 \cdot 10^{-1}$	0.1266s
	5	$6.594 \cdot 10^{-2}$	0.0184s	$5.740 \cdot 10^{-2}$	1.6618s	$5.874 \cdot 10^{-2}$	0.1174s
	6	$3.798 \cdot 10^{-1}$	0.0130s	$2.331 \cdot 10^{-1}$	1.3438s	$2.728 \cdot 10^{-1}$	0.1300s
	7	$1.323 \cdot 10^{-1}$	0.0150s	$9.522 \cdot 10^{-2}$	1.5691s	$9.930 \cdot 10^{-2}$	0.1322s
	8	$1.731 \cdot 10^{-1}$	0.0121s	$1.379 \cdot 10^{-1}$	0.8342s	$1.434 \cdot 10^{-1}$	0.1246s
	9	$2.682 \cdot 10^{-1}$	0.0095s	$2.045 \cdot 10^{-1}$	1.5110s	$2.045 \cdot 10^{-1}$	0.1235s

Table 8: Oregon, variant algorithm 2

k	Sub	Algorithm 3		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$6.361 \cdot 10^{-1}$	1.2983s	$6.373 \cdot 10^{-1}$	1.2626s	$6.475 \cdot 10^{-1}$	0.1256s
	3	$4.101 \cdot 10^{-1}$	1.2696s	$3.945 \cdot 10^{-1}$	0.7504s	$4.107 \cdot 10^{-1}$	0.0932s
	4	$6.436 \cdot 10^{-1}$	0.7226s	$6.149 \cdot 10^{-1}$	1.3706s	$6.259 \cdot 10^{-1}$	0.1260s
	5	$1.823 \cdot 10^{-1}$	1.0179s	$1.835 \cdot 10^{-1}$	1.2867s	$1.890 \cdot 10^{-1}$	0.1241s
	6	$8.638 \cdot 10^{-1}$	1.3038s	$8.515 \cdot 10^{-1}$	1.4793s	$9.861 \cdot 10^{-1}$	0.1113s
	7	$2.793 \cdot 10^{-1}$	1.0555s	$2.794 \cdot 10^{-1}$	1.1700s	$2.972 \cdot 10^{-1}$	0.1243s
	8	$4.065 \cdot 10^{-1}$	1.1746s	$4.079 \cdot 10^{-1}$	1.6288s	$4.207 \cdot 10^{-1}$	0.0154s
	9	$5.884 \cdot 10^{-1}$	1.9977s	$5.912 \cdot 10^{-1}$	2.2449s	$6.370 \cdot 10^{-1}$	0.0174s
15	2	$4.232 \cdot 10^{-1}$	1.5184s	$3.983 \cdot 10^{-1}$	1.3988s	$4.436 \cdot 10^{-1}$	0.2464s
	3	$2.536 \cdot 10^{-1}$	1.3319s	$2.246 \cdot 10^{-1}$	1.3421s	$2.532 \cdot 10^{-1}$	0.2645s
	4	$3.884 \cdot 10^{-1}$	1.0189s	$3.784 \cdot 10^{-1}$	1.1875s	$3.800 \cdot 10^{-1}$	0.2461s
	5	$1.114 \cdot 10^{-1}$	1.3412s	$1.116 \cdot 10^{-1}$	1.5515s	$1.251 \cdot 10^{-1}$	0.0724s
	6	$5.339 \cdot 10^{-1}$	1.4318s	$5.390 \cdot 10^{-1}$	1.2702s	$5.557 \cdot 10^{-1}$	0.1187s
	7	$1.817 \cdot 10^{-1}$	1.6682s	$1.835 \cdot 10^{-1}$	1.1640s	$1.863 \cdot 10^{-1}$	0.1317s
	8	$2.635 \cdot 10^{-1}$	1.1486s	$2.634 \cdot 10^{-1}$	1.1158s	$2.773 \cdot 10^{-1}$	0.1280s
	9	$4.002 \cdot 10^{-1}$	1.4166s	$3.976 \cdot 10^{-1}$	1.6409s	$4.305 \cdot 10^{-1}$	0.0941s
20	2	$2.810 \cdot 10^{-1}$	1.4795s	$2.728 \cdot 10^{-1}$	1.3377s	$3.039 \cdot 10^{-1}$	0.0240s
	3	$1.625 \cdot 10^{-1}$	1.3817s	$1.558 \cdot 10^{-1}$	1.2294s	$1.684 \cdot 10^{-1}$	0.0186s
	4	$2.626 \cdot 10^{-1}$	1.6044s	$2.371 \cdot 10^{-1}$	1.1419s	$2.722 \cdot 10^{-1}$	0.1133s
	5	$8.011 \cdot 10^{-2}$	1.9453s	$7.898 \cdot 10^{-2}$	2.6930s	$8.107 \cdot 10^{-2}$	0.1330s
	6	$3.561 \cdot 10^{-1}$	1.2885s	$3.486 \cdot 10^{-1}$	1.2688s	$3.676 \cdot 10^{-1}$	0.0175s
	7	$1.345 \cdot 10^{-1}$	1.2318s	$1.330 \cdot 10^{-1}$	1.4949s	$1.342 \cdot 10^{-1}$	0.1302s
	8	$1.920 \cdot 10^{-1}$	1.1142s	$1.792 \cdot 10^{-1}$	1.5520s	$1.884 \cdot 10^{-1}$	0.0213s
	9	$2.878 \cdot 10^{-1}$	0.8725s	$2.811 \cdot 10^{-1}$	1.5791s	$3.204 \cdot 10^{-1}$	0.0352s
25	2	$1.971 \cdot 10^{-1}$	2.2545s	$1.907 \cdot 10^{-1}$	1.8226s	$2.133 \cdot 10^{-1}$	0.0269s
	3	$1.247 \cdot 10^{-1}$	1.6683s	$1.159 \cdot 10^{-1}$	2.2576s	$1.264 \cdot 10^{-1}$	0.0265s
	4	$1.848 \cdot 10^{-1}$	1.5967s	$1.853 \cdot 10^{-1}$	0.8102s	$1.859 \cdot 10^{-1}$	0.0387s
	5	$6.245 \cdot 10^{-2}$	1.0402s	$5.740 \cdot 10^{-2}$	1.4918s	$5.874 \cdot 10^{-2}$	0.0271s
	6	$2.331 \cdot 10^{-1}$	1.5086s	$2.331 \cdot 10^{-1}$	1.0876s	$2.728 \cdot 10^{-1}$	0.0286s
	7	$1.070 \cdot 10^{-1}$	0.8185s	$9.522 \cdot 10^{-2}$	1.6200s	$9.930 \cdot 10^{-2}$	0.0304s
	8	$1.342 \cdot 10^{-1}$	0.8210s	$1.379 \cdot 10^{-1}$	1.0859s	$1.434 \cdot 10^{-1}$	0.0273s
	9	$2.088 \cdot 10^{-1}$	1.3330s	$2.045 \cdot 10^{-1}$	0.9393s	$2.045 \cdot 10^{-1}$	0.0276s

Table 9: Oregon, algorithm 3

k	Sub	Algorithm 4		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$8.510 \cdot 10^{-1}$	0.0114s	$6.373 \cdot 10^{-1}$	1.5112s	$6.475 \cdot 10^{-1}$	0.1253s
	3	$4.133 \cdot 10^{-1}$	0.0101s	$3.945 \cdot 10^{-1}$	1.3757s	$4.107 \cdot 10^{-1}$	0.1232s
	4	1.195	0.0083s	$6.149 \cdot 10^{-1}$	1.4077s	$6.259 \cdot 10^{-1}$	0.0170s
	5	$1.888 \cdot 10^{-1}$	0.0093s	$1.835 \cdot 10^{-1}$	1.6330s	$1.890 \cdot 10^{-1}$	0.1245s
	6	1.118	0.0113s	$8.515 \cdot 10^{-1}$	1.4236s	$9.861 \cdot 10^{-1}$	0.0216s
	7	$3.556 \cdot 10^{-1}$	0.0082s	$2.794 \cdot 10^{-1}$	2.5596s	$2.972 \cdot 10^{-1}$	0.1226s
	8	$4.670 \cdot 10^{-1}$	0.0114s	$4.079 \cdot 10^{-1}$	1.3383s	$4.207 \cdot 10^{-1}$	0.0159s
	9	$6.428 \cdot 10^{-1}$	0.0176s	$5.912 \cdot 10^{-1}$	1.0034s	$6.370 \cdot 10^{-1}$	0.1232s
15	2	$4.379 \cdot 10^{-1}$	0.0186s	$3.983 \cdot 10^{-1}$	1.5813s	$4.436 \cdot 10^{-1}$	0.0611s
	3	$3.236 \cdot 10^{-1}$	0.0186s	$2.246 \cdot 10^{-1}$	1.2899s	$2.532 \cdot 10^{-1}$	0.0186s
	4	$4.943 \cdot 10^{-1}$	0.0092s	$3.784 \cdot 10^{-1}$	1.3770s	$3.800 \cdot 10^{-1}$	0.0195s
	5	$1.436 \cdot 10^{-1}$	0.0081s	$1.116 \cdot 10^{-1}$	1.4911s	$1.251 \cdot 10^{-1}$	0.0964s
	6	1.071	0.0089s	$5.390 \cdot 10^{-1}$	1.2522s	$5.557 \cdot 10^{-1}$	0.0625s
	7	$2.143 \cdot 10^{-1}$	0.0207s	$1.835 \cdot 10^{-1}$	2.3024s	$1.863 \cdot 10^{-1}$	0.1261s
	8	$3.711 \cdot 10^{-1}$	0.0206s	$2.634 \cdot 10^{-1}$	1.1885s	$2.773 \cdot 10^{-1}$	0.0136s
	9	$5.702 \cdot 10^{-1}$	0.0099s	$3.976 \cdot 10^{-1}$	1.2163s	$4.305 \cdot 10^{-1}$	0.0289s
20	2	$3.500 \cdot 10^{-1}$	0.3109s	$2.728 \cdot 10^{-1}$	1.2277s	$3.039 \cdot 10^{-1}$	0.0186s
	3	$2.205 \cdot 10^{-1}$	0.0524s	$1.558 \cdot 10^{-1}$	2.2401s	$1.684 \cdot 10^{-1}$	0.0239s
	4	$2.713 \cdot 10^{-1}$	0.0109s	$2.371 \cdot 10^{-1}$	1.0508s	$2.722 \cdot 10^{-1}$	0.0209s
	5	$8.874 \cdot 10^{-2}$	0.0123s	$7.898 \cdot 10^{-2}$	1.7188s	$8.107 \cdot 10^{-2}$	0.1164s
	6	$7.965 \cdot 10^{-1}$	0.0118s	$3.486 \cdot 10^{-1}$	1.0923s	$3.676 \cdot 10^{-1}$	0.1032s
	7	$2.377 \cdot 10^{-1}$	0.0098s	$1.330 \cdot 10^{-1}$	1.5592s	$1.342 \cdot 10^{-1}$	0.1308s
	8	$2.750 \cdot 10^{-1}$	0.0268s	$1.792 \cdot 10^{-1}$	1.4430s	$1.884 \cdot 10^{-1}$	0.0212s
	9	$4.437 \cdot 10^{-1}$	0.0134s	$2.811 \cdot 10^{-1}$	1.4726s	$3.204 \cdot 10^{-1}$	0.0718s
25	2	$2.347 \cdot 10^{-1}$	0.0339s	$1.907 \cdot 10^{-1}$	2.0133s	$2.133 \cdot 10^{-1}$	0.0605s
	3	$1.406 \cdot 10^{-1}$	0.0102s	$1.159 \cdot 10^{-1}$	1.3949s	$1.264 \cdot 10^{-1}$	0.0660s
	4	$2.903 \cdot 10^{-1}$	0.0140s	$1.853 \cdot 10^{-1}$	0.5253s	$1.859 \cdot 10^{-1}$	0.0226s
	5	$7.796 \cdot 10^{-2}$	0.0177s	$5.740 \cdot 10^{-2}$	2.0081s	$5.874 \cdot 10^{-2}$	0.0323s
	6	$4.985 \cdot 10^{-1}$	0.0140s	$2.331 \cdot 10^{-1}$	0.8518s	$2.728 \cdot 10^{-1}$	0.0226s
	7	$1.155 \cdot 10^{-1}$	0.0300s	$9.522 \cdot 10^{-2}$	1.4330s	$9.930 \cdot 10^{-2}$	0.0406s
	8	$2.300 \cdot 10^{-1}$	0.0162s	$1.379 \cdot 10^{-1}$	1.0439s	$1.434 \cdot 10^{-1}$	0.0239s
	9	$3.547 \cdot 10^{-1}$	0.0210s	$2.045 \cdot 10^{-1}$	1.5785s	$2.045 \cdot 10^{-1}$	0.0262s

Table 10: Oregon, algorithm 4

k	Sub	Algorithm 4v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$8.510 \cdot 10^{-1}$	0.0076s	$6.373 \cdot 10^{-1}$	1.2536s	$6.475 \cdot 10^{-1}$	0.0181s
	3	$5.567 \cdot 10^{-1}$	0.0121s	$3.945 \cdot 10^{-1}$	1.3395s	$4.107 \cdot 10^{-1}$	0.0133s
	4	$8.245 \cdot 10^{-1}$	0.0094s	$6.149 \cdot 10^{-1}$	1.4244s	$6.259 \cdot 10^{-1}$	0.0152s
	5	$2.167 \cdot 10^{-1}$	0.0201s	$1.835 \cdot 10^{-1}$	1.3931s	$1.890 \cdot 10^{-1}$	0.1231s
	6	1.385	0.0097s	$8.515 \cdot 10^{-1}$	1.6441s	$9.861 \cdot 10^{-1}$	0.0942s
	7	$3.510 \cdot 10^{-1}$	0.0170s	$2.794 \cdot 10^{-1}$	1.2055s	$2.972 \cdot 10^{-1}$	0.0142s
	8	$4.538 \cdot 10^{-1}$	0.0128s	$4.079 \cdot 10^{-1}$	1.0800s	$4.207 \cdot 10^{-1}$	0.1286s
	9	$7.332 \cdot 10^{-1}$	0.0126s	$5.912 \cdot 10^{-1}$	1.1476s	$6.370 \cdot 10^{-1}$	0.1230s
15	2	$4.379 \cdot 10^{-1}$	0.0230s	$3.983 \cdot 10^{-1}$	1.6336s	$4.436 \cdot 10^{-1}$	0.0218s
	3	$3.146 \cdot 10^{-1}$	0.0137s	$2.246 \cdot 10^{-1}$	1.3758s	$2.532 \cdot 10^{-1}$	0.0712s
	4	$6.898 \cdot 10^{-1}$	0.0099s	$3.784 \cdot 10^{-1}$	1.1290s	$3.800 \cdot 10^{-1}$	0.0671s
	5	$1.332 \cdot 10^{-1}$	0.0183s	$1.116 \cdot 10^{-1}$	1.5793s	$1.251 \cdot 10^{-1}$	0.0175s
	6	1.153	0.0119s	$5.390 \cdot 10^{-1}$	2.0519s	$5.557 \cdot 10^{-1}$	0.1279s
	7	$3.213 \cdot 10^{-1}$	0.0150s	$1.835 \cdot 10^{-1}$	1.5596s	$1.863 \cdot 10^{-1}$	0.1292s
	8	$3.423 \cdot 10^{-1}$	0.0150s	$2.634 \cdot 10^{-1}$	1.0773s	$2.773 \cdot 10^{-1}$	0.1021s
	9	$6.096 \cdot 10^{-1}$	0.0122s	$3.976 \cdot 10^{-1}$	0.9250s	$4.305 \cdot 10^{-1}$	0.1264s
20	2	$3.500 \cdot 10^{-1}$	0.0124s	$2.728 \cdot 10^{-1}$	1.1912s	$3.039 \cdot 10^{-1}$	0.2325s
	3	$2.757 \cdot 10^{-1}$	0.6226s	$1.558 \cdot 10^{-1}$	1.3484s	$1.684 \cdot 10^{-1}$	0.1781s
	4	$4.472 \cdot 10^{-1}$	0.1153s	$2.371 \cdot 10^{-1}$	1.1726s	$2.722 \cdot 10^{-1}$	0.2005s
	5	$1.111 \cdot 10^{-1}$	0.2680s	$7.898 \cdot 10^{-2}$	1.7233s	$8.107 \cdot 10^{-2}$	0.4618s
	6	$5.726 \cdot 10^{-1}$	0.0199s	$3.486 \cdot 10^{-1}$	0.7419s	$3.676 \cdot 10^{-1}$	0.0484s
	7	$1.898 \cdot 10^{-1}$	0.0252s	$1.330 \cdot 10^{-1}$	1.7141s	$1.342 \cdot 10^{-1}$	0.0651s
	8	$2.634 \cdot 10^{-1}$	0.0221s	$1.792 \cdot 10^{-1}$	0.9403s	$1.884 \cdot 10^{-1}$	0.0255s
	9	$3.710 \cdot 10^{-1}$	0.0209s	$2.811 \cdot 10^{-1}$	1.2845s	$3.204 \cdot 10^{-1}$	0.1245s
25	2	$2.347 \cdot 10^{-1}$	0.0635s	$1.907 \cdot 10^{-1}$	1.8991s	$2.133 \cdot 10^{-1}$	0.0244s
	3	$1.999 \cdot 10^{-1}$	0.0454s	$1.159 \cdot 10^{-1}$	0.8588s	$1.264 \cdot 10^{-1}$	0.0257s
	4	$3.121 \cdot 10^{-1}$	0.0205s	$1.853 \cdot 10^{-1}$	0.4874s	$1.859 \cdot 10^{-1}$	0.1319s
	5	$9.654 \cdot 10^{-2}$	0.0316s	$5.740 \cdot 10^{-2}$	1.0336s	$5.874 \cdot 10^{-2}$	0.1236s
	6	$4.776 \cdot 10^{-1}$	0.0274s	$2.331 \cdot 10^{-1}$	0.8411s	$2.728 \cdot 10^{-1}$	0.0243s
	7	$1.961 \cdot 10^{-1}$	0.0190s	$9.522 \cdot 10^{-2}$	1.5809s	$9.930 \cdot 10^{-2}$	0.1282s
	8	$2.470 \cdot 10^{-1}$	0.0290s	$1.379 \cdot 10^{-1}$	1.6585s	$1.434 \cdot 10^{-1}$	0.1171s
	9	$3.404 \cdot 10^{-1}$	0.0245s	$2.045 \cdot 10^{-1}$	1.7565s	$2.045 \cdot 10^{-1}$	0.0253s

Table 11: Oregon, variant algorithm 4

k	Sub	Algorithm 5		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$8.510 \cdot 10^{-1}$	0.0163	$6.373 \cdot 10^{-1}$	4.9200s	$6.475 \cdot 10^{-1}$	0.1233s
	3	$4.096 \cdot 10^{-1}$	1.3378s	$3.945 \cdot 10^{-1}$	0.9978s	$4.107 \cdot 10^{-1}$	0.0132s
	4	$7.170 \cdot 10^{-1}$	1.6021s	$6.149 \cdot 10^{-1}$	1.3965s	$6.259 \cdot 10^{-1}$	0.0134s
	5	$1.983 \cdot 10^{-1}$	2.4848s	$1.835 \cdot 10^{-1}$	0.8627s	$1.890 \cdot 10^{-1}$	0.0122s
	6	1.081	2.1289s	$8.515 \cdot 10^{-1}$	1.3196s	$9.861 \cdot 10^{-1}$	0.1286s
	7	$3.377 \cdot 10^{-1}$	2.1713s	$2.794 \cdot 10^{-1}$	2.0470s	$2.972 \cdot 10^{-1}$	0.1267s
	8	$4.514 \cdot 10^{-1}$	2.8752s	$4.079 \cdot 10^{-1}$	1.3572s	$4.207 \cdot 10^{-1}$	0.1050s
	9	$6.338 \cdot 10^{-1}$	4.3805s	$5.912 \cdot 10^{-1}$	1.2610s	$6.370 \cdot 10^{-1}$	0.0159s
15	2	$4.379 \cdot 10^{-1}$	0.0113s	$3.983 \cdot 10^{-1}$	5.1807s	$4.436 \cdot 10^{-1}$	0.0159s
	3	$2.560 \cdot 10^{-1}$	1.9580s	$2.246 \cdot 10^{-1}$	1.3755s	$2.532 \cdot 10^{-1}$	0.0218s
	4	$4.483 \cdot 10^{-1}$	1.6626s	$3.784 \cdot 10^{-1}$	1.4867s	$3.800 \cdot 10^{-1}$	0.0197s
	5	$1.286 \cdot 10^{-1}$	2.4200s	$1.116 \cdot 10^{-1}$	1.3027s	$1.251 \cdot 10^{-1}$	0.1228s
	6	$7.145 \cdot 10^{-1}$	3.7768s	$5.390 \cdot 10^{-1}$	1.7950s	$5.557 \cdot 10^{-1}$	0.1286s
	7	$2.456 \cdot 10^{-1}$	3.1186s	$1.835 \cdot 10^{-1}$	1.3019s	$1.863 \cdot 10^{-1}$	0.1264s
	8	$2.883 \cdot 10^{-1}$	3.5988s	$2.634 \cdot 10^{-1}$	1.5303s	$2.773 \cdot 10^{-1}$	0.1265s
	9	$4.449 \cdot 10^{-1}$	4.8061s	$3.976 \cdot 10^{-1}$	1.6773s	$4.305 \cdot 10^{-1}$	0.1268s
20	2	$3.500 \cdot 10^{-1}$	0.0182s	$2.728 \cdot 10^{-1}$	7.4553s	$3.039 \cdot 10^{-1}$	0.0981s
	3	$1.770 \cdot 10^{-1}$	1.9258s	$1.558 \cdot 10^{-1}$	0.8835s	$1.684 \cdot 10^{-1}$	0.0186s
	4	$2.719 \cdot 10^{-1}$	2.4742s	$2.371 \cdot 10^{-1}$	0.9204s	$2.722 \cdot 10^{-1}$	0.0206s
	5	$7.936 \cdot 10^{-2}$	3.9520s	$7.898 \cdot 10^{-2}$	1.5416s	$8.107 \cdot 10^{-2}$	0.0714s
	6	$5.131 \cdot 10^{-1}$	3.0751s	$3.486 \cdot 10^{-1}$	1.5213s	$3.676 \cdot 10^{-1}$	0.1267s
	7	$1.704 \cdot 10^{-1}$	4.0074s	$1.330 \cdot 10^{-1}$	1.2714s	$1.342 \cdot 10^{-1}$	0.1297s
	8	$2.106 \cdot 10^{-1}$	5.1989s	$1.792 \cdot 10^{-1}$	1.6123s	$1.884 \cdot 10^{-1}$	0.1083s
	9	$3.136 \cdot 10^{-1}$	4.6288s	$2.811 \cdot 10^{-1}$	1.9894s	$3.204 \cdot 10^{-1}$	0.0181s
25	2	$2.347 \cdot 10^{-1}$	0.0189s	$1.907 \cdot 10^{-1}$	7.9624s	$2.133 \cdot 10^{-1}$	0.0228s
	3	$1.382 \cdot 10^{-1}$	1.8886s	$1.159 \cdot 10^{-1}$	1.0798s	$1.264 \cdot 10^{-1}$	0.0250s
	4	$2.135 \cdot 10^{-1}$	3.1943s	$1.853 \cdot 10^{-1}$	1.9308s	$1.859 \cdot 10^{-1}$	0.0240s
	5	$6.895 \cdot 10^{-2}$	4.5371s	$5.740 \cdot 10^{-2}$	1.8448s	$5.874 \cdot 10^{-2}$	0.0273s
	6	$3.804 \cdot 10^{-1}$	4.2062s	$2.331 \cdot 10^{-1}$	1.3910s	$2.728 \cdot 10^{-1}$	0.0243s
	7	$1.143 \cdot 10^{-1}$	5.1707s	$9.522 \cdot 10^{-2}$	1.7130s	$9.930 \cdot 10^{-2}$	0.0319s
	8	$1.697 \cdot 10^{-1}$	5.0434s	$1.379 \cdot 10^{-1}$	0.9108s	$1.434 \cdot 10^{-1}$	0.0711s
	9	$2.436 \cdot 10^{-1}$	6.8343s	$2.045 \cdot 10^{-1}$	1.2462s	$2.045 \cdot 10^{-1}$	0.1256s

Table 12: Oregon, algorithm 5

k	Sub	Algorithm 6		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.629 \cdot 10^{-1}$	0.0926s	$6.373 \cdot 10^{-1}$	1.3608s	$6.475 \cdot 10^{-1}$	0.0253s
	3	$5.748 \cdot 10^{-1}$	0.0300s	$3.945 \cdot 10^{-1}$	1.1528s	$4.107 \cdot 10^{-1}$	0.0270s
	4	$7.688 \cdot 10^{-1}$	0.1268s	$6.149 \cdot 10^{-1}$	1.2206s	$6.259 \cdot 10^{-1}$	0.0245s
	5	$1.927 \cdot 10^{-1}$	0.1358s	$1.835 \cdot 10^{-1}$	1.5944s	$1.890 \cdot 10^{-1}$	0.1282s
	6	1.107	0.0468s	$8.515 \cdot 10^{-1}$	1.1818s	$9.861 \cdot 10^{-1}$	0.0949s
	7	$3.520 \cdot 10^{-1}$	0.1345s	$2.794 \cdot 10^{-1}$	1.3602s	$2.972 \cdot 10^{-1}$	0.0869s
	8	$4.552 \cdot 10^{-1}$	0.0894s	$4.079 \cdot 10^{-1}$	2.5123s	$4.207 \cdot 10^{-1}$	0.1327s
	9	$7.435 \cdot 10^{-1}$	0.0637s	$5.912 \cdot 10^{-1}$	1.6970s	$6.370 \cdot 10^{-1}$	0.0229s
15	2	$5.121 \cdot 10^{-1}$	0.0418s	$3.983 \cdot 10^{-1}$	1.1794s	$4.436 \cdot 10^{-1}$	0.1117s
	3	$4.124 \cdot 10^{-1}$	0.1281s	$2.246 \cdot 10^{-1}$	1.7307s	$2.532 \cdot 10^{-1}$	0.1330s
	4	$6.046 \cdot 10^{-1}$	0.0851s	$3.784 \cdot 10^{-1}$	1.5090s	$3.800 \cdot 10^{-1}$	0.1312s
	5	$1.320 \cdot 10^{-1}$	0.1322s	$1.116 \cdot 10^{-1}$	2.2422s	$1.251 \cdot 10^{-1}$	0.0281s
	6	$8.401 \cdot 10^{-1}$	0.1320s	$5.390 \cdot 10^{-1}$	2.0080s	$5.557 \cdot 10^{-1}$	0.0379s
	7	$2.472 \cdot 10^{-1}$	0.1367s	$1.835 \cdot 10^{-1}$	1.7290s	$1.863 \cdot 10^{-1}$	0.1275s
	8	$3.254 \cdot 10^{-1}$	0.0583s	$2.634 \cdot 10^{-1}$	1.2109s	$2.773 \cdot 10^{-1}$	0.0280s
	9	$5.382 \cdot 10^{-1}$	0.0385s	$3.976 \cdot 10^{-1}$	1.3039s	$4.305 \cdot 10^{-1}$	0.0292s
20	2	$3.972 \cdot 10^{-1}$	0.1358s	$2.728 \cdot 10^{-1}$	1.7720s	$3.039 \cdot 10^{-1}$	0.3122s
	3	$2.737 \cdot 10^{-1}$	0.1263s	$1.558 \cdot 10^{-1}$	1.5706s	$1.684 \cdot 10^{-1}$	0.2209s
	4	$3.827 \cdot 10^{-1}$	0.1306s	$2.371 \cdot 10^{-1}$	2.8839s	$2.722 \cdot 10^{-1}$	0.2755s
	5	$9.290 \cdot 10^{-2}$	0.1244s	$7.898 \cdot 10^{-2}$	2.0933s	$8.107 \cdot 10^{-2}$	0.7184s
	6	$5.769 \cdot 10^{-1}$	0.1261s	$3.486 \cdot 10^{-1}$	1.9582s	$3.676 \cdot 10^{-1}$	0.1027s
	7	$2.047 \cdot 10^{-1}$	0.1395s	$1.330 \cdot 10^{-1}$	2.2355s	$1.342 \cdot 10^{-1}$	0.3548s
	8	$2.544 \cdot 10^{-1}$	0.0478s	$1.792 \cdot 10^{-1}$	1.3499s	$1.884 \cdot 10^{-1}$	0.0654s
	9	$3.897 \cdot 10^{-1}$	0.1305s	$2.811 \cdot 10^{-1}$	1.9465s	$3.204 \cdot 10^{-1}$	0.1344s
25	2	$2.516 \cdot 10^{-1}$	0.1245s	$1.907 \cdot 10^{-1}$	2.0291s	$2.133 \cdot 10^{-1}$	0.0372s
	3	$2.470 \cdot 10^{-1}$	0.1299s	$1.159 \cdot 10^{-1}$	1.9384s	$1.264 \cdot 10^{-1}$	0.0826s
	4	$2.956 \cdot 10^{-1}$	0.1345s	$1.853 \cdot 10^{-1}$	1.7898s	$1.859 \cdot 10^{-1}$	0.0384s
	5	$8.549 \cdot 10^{-2}$	0.1722s	$5.740 \cdot 10^{-2}$	3.6517s	$5.874 \cdot 10^{-2}$	0.0498s
	6	$5.320 \cdot 10^{-1}$	0.0697s	$2.331 \cdot 10^{-1}$	1.8793s	$2.728 \cdot 10^{-1}$	0.0374s
	7	$1.923 \cdot 10^{-1}$	0.1503s	$9.522 \cdot 10^{-2}$	2.1395s	$9.930 \cdot 10^{-2}$	0.1310s
	8	$1.992 \cdot 10^{-1}$	0.0653s	$1.379 \cdot 10^{-1}$	1.9048s	$1.434 \cdot 10^{-1}$	0.0553s
	9	$2.826 \cdot 10^{-1}$	0.1297s	$2.045 \cdot 10^{-1}$	1.9391s	$2.045 \cdot 10^{-1}$	0.0403s

Table 13: Oregon, algorithm 6

5.1.1 Analysis

We gathered and plotted the results reported in the tables in the plots below, hence obtaining a better understanding on how each algorithm performed. We also measured the computation times each algorithm took for each subset, plotting them in a separated file. On the x-axis the number of the subset is written, while the y-axis represents the cost of the objective function or the computation time in seconds. Only the values 10 and 25 of k have been used because, being the two extremes, they are the most significant values. Moreover, it is reasonable to expect that the plot for $k = 15$ would look similar to the one for $k = 10$, since the difference in the number of clusters is only 5; a similar reasoning can be applied for $k = 20$ with respect to $k = 25$.

As a last note, in the legend present in every plot we referred to the variants of algorithms 2 and 4 as "2v" and "4v" for brevity, and for the same reason we referred to the two baselines as "Km++L 20 init" and "Km++L 1 init", meaning that they have been initialized 20 times and 1 time respectively. All this will hold also for the analysis of the other datasets.

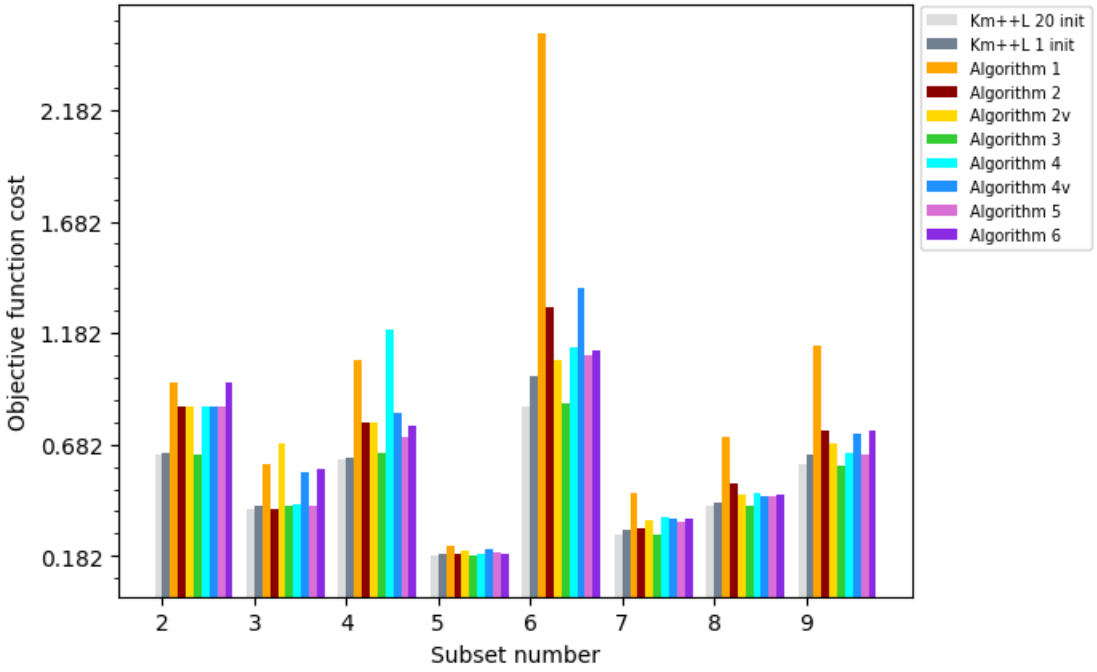

Figure 7: Oregon, $k = 10$, objective function cost

Figure 7 shows the objective function cost obtained by each algorithm on each subset of Oregon for $k = 10$. We can immediately see that algorithm 3 is the one with the overall best performance for each subset. One explanation may be that just half of a subset is enough to find good centers for the entire subset, even if this depends on the half-subset that is being analyzed. In fact, if the majority of the elements in the half-subset are outliers or elements near them, then it will not be a good sample of the original subset and the centers that can be found on it will not work well on the entire subset.

Algorithm 1, instead, has the highest costs in the majority of the subsets. This fact is not entirely surprising since it is the most naive algorithm among all.

As for the other algorithms, their performance varies depending on the subset we consider. For example, algorithm 2 seems to work well for subsets 3, 5 and 7, but in other subsets, especially in 6, the cost it produces is higher than the one obtained by the baselines. A similar reasoning can be made for the remaining algorithms.

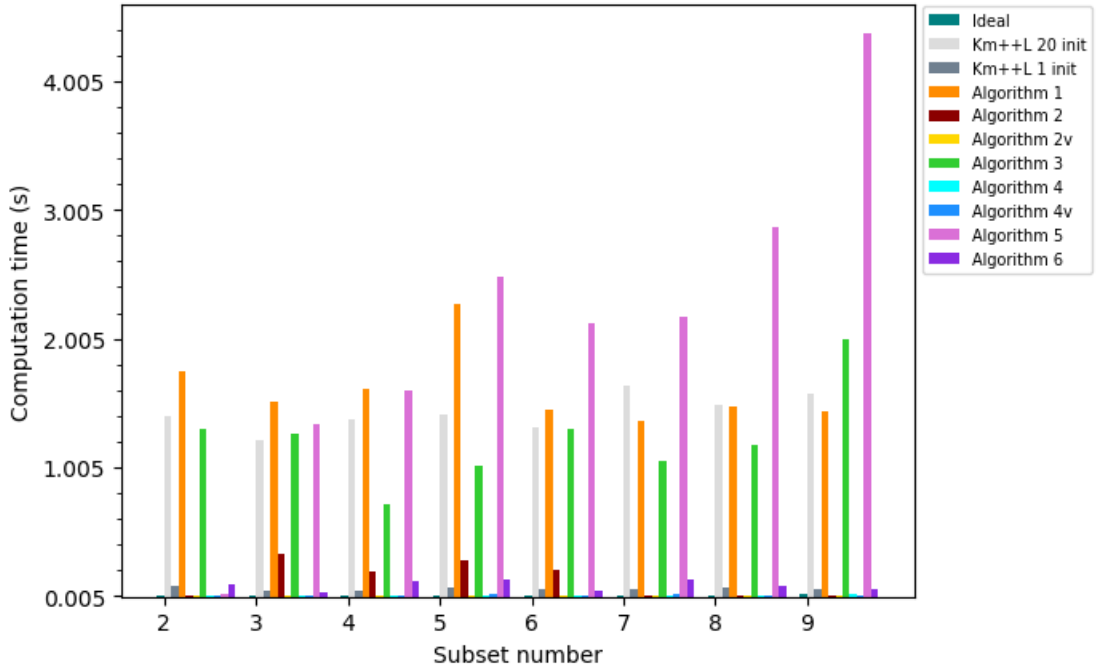

Figure 8: Oregon, $k = 10$, computation time

Figure 8 shows the running time of each algorithm with respect to the analyzed subsets of Oregon, for $k = 10$. The first thing that is worth noting is how the computation time of algorithm 5 more or less increases at each subset, except for subset 6, reaching its peak in the last one. This behaviour has to be expected due to the fact that this algorithm progressively merge all the previous subsets and performs, at each iteration, `Km++L` on the merged set, hence it progressively takes more time.

Shifting the focus on the other algorithms, it is clear that some of them are very efficient, even when compared to the baseline with 1 initialization. In fact algorithms 2v, 4, 4v and occasionally also 2 and 6 all took less computation time than Km++L 1 init, and the first three of them are also competitive with the Ideal baseline. Unfortunately the costs they produce are not smaller as well, as we have seen in the plot above. However this may still be a good trade-off, in particular if we are interested in a quick initial analysis of a very big dataset.

The last interesting result to comment is the one of algorithm 3, the best performer

in terms of cost of the objective function: it is clear that its computation time is on average close to that of the baseline initialized 20 times, and since sometimes it also produces better costs than this baseline we can argue that it is a competitive algorithm.

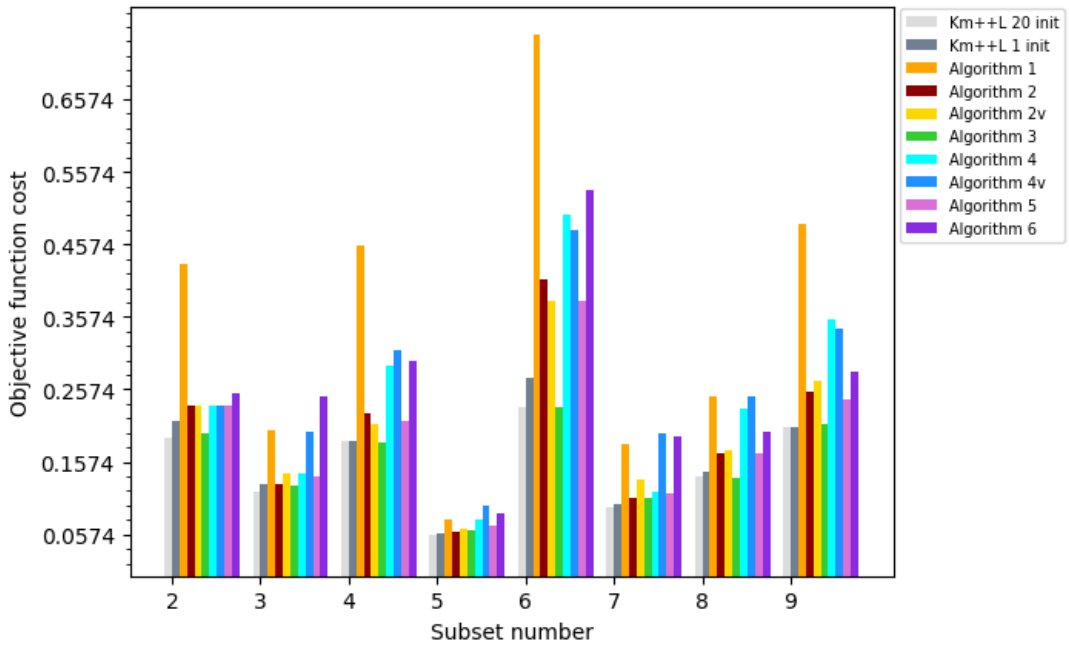

Figure 9: Oregon, $k = 25$, objective function cost

Figure 9 shows the objective function cost obtained by each algorithm on each subset of Oregon for $k = 25$. With respect to figure 7 we can see that the performance of algorithms 1, 4, 4v and 6 has generally got worse. To be more precise, the costs they obtained are generally higher than those of the other algorithms, as well as than those of the baselines. If we focus on algorithm 3, instead, we can see that it remains competitive in most of the subsets. Lastly, it is interesting to notice that algorithm 5 seems to be consistent in terms of performance as k varies.

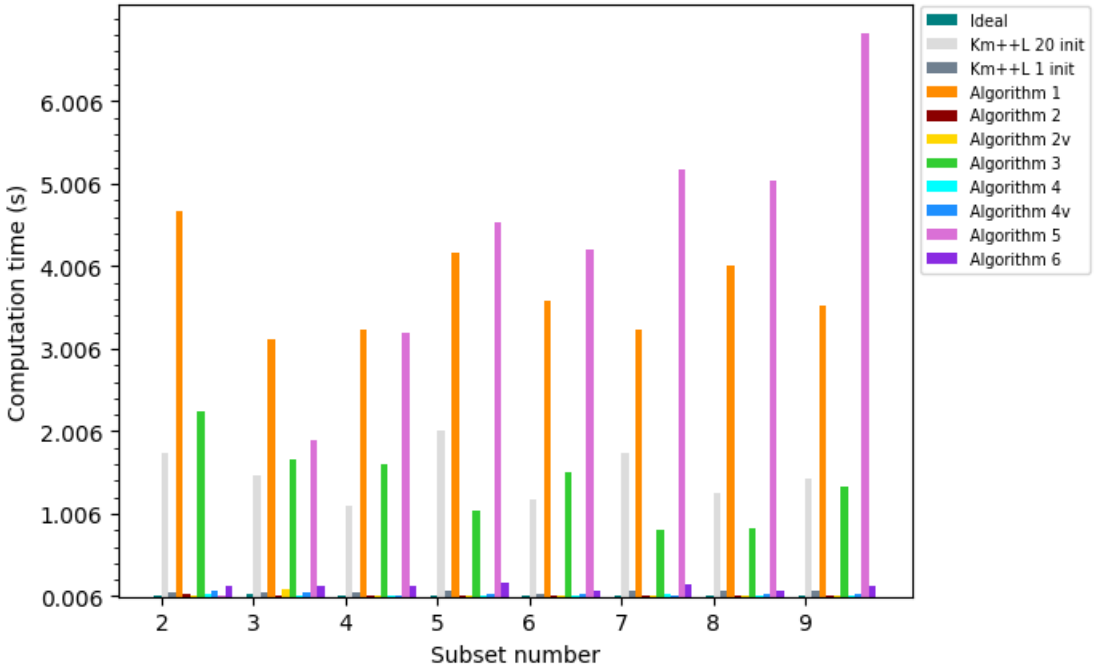

Figure 10: Oregon, $k = 25$, computation time

Figure 10 shows the running time of each algorithm with respect to the analyzed subsets of Oregon for $k = 25$. Results are very similar to the ones found for $k = 10$, depicted in figure 8, so the same considerations apply.

5.2 Bank

We report here the results obtained on the Bank dataset.

k	Sub	Algorithm 1		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.750 \cdot 10^{12}$	2.3233s	$1.662 \cdot 10^{12}$	2.4255s	$1.671 \cdot 10^{12}$	0.1518s
	3	$2.325 \cdot 10^{12}$	2.0739s	$1.872 \cdot 10^{12}$	1.9903s	$2.126 \cdot 10^{12}$	0.1526s
	4	$2.026 \cdot 10^{12}$	2.3361s	$1.823 \cdot 10^{12}$	2.0988s	$1.949 \cdot 10^{12}$	0.1654s
	5	$1.371 \cdot 10^{12}$	2.1092s	$1.241 \cdot 10^{12}$	4.1199s	$1.399 \cdot 10^{12}$	0.1644s
	6	$1.993 \cdot 10^{12}$	2.9887s	$1.637 \cdot 10^{12}$	2.4960s	$1.673 \cdot 10^{12}$	0.1610s
	7	$1.463 \cdot 10^{12}$	2.3052s	$1.271 \cdot 10^{12}$	2.2706s	$1.299 \cdot 10^{12}$	0.1565s
15	2	$1.320 \cdot 10^{12}$	2.5304s	$1.175 \cdot 10^{12}$	3.0871s	$1.201 \cdot 10^{12}$	0.1757s
	3	$1.702 \cdot 10^{12}$	3.2297s	$1.266 \cdot 10^{12}$	2.8770s	$1.339 \cdot 10^{12}$	0.1725s
	4	$1.602 \cdot 10^{12}$	3.2268s	$1.278 \cdot 10^{12}$	4.1790s	$1.353 \cdot 10^{12}$	0.0949s
	5	$1.037 \cdot 10^{12}$	3.1901s	$8.405 \cdot 10^{11}$	2.9523s	$9.259 \cdot 10^{11}$	0.1410s
	6	$1.602 \cdot 10^{12}$	4.3533s	$1.149 \cdot 10^{12}$	2.7854s	$1.297 \cdot 10^{12}$	0.1933s
	7	$9.942 \cdot 10^{11}$	2.8607s	$8.792 \cdot 10^{11}$	4.2868s	$9.071 \cdot 10^{11}$	0.1574s
20	2	$1.020 \cdot 10^{12}$	3.1177s	$8.979 \cdot 10^{11}$	3.1511s	$9.401 \cdot 10^{11}$	0.6172s
	3	$1.507 \cdot 10^{12}$	4.4850s	$9.670 \cdot 10^{11}$	3.2010s	$9.865 \cdot 10^{11}$	0.6781s
	4	$1.166 \cdot 10^{12}$	4.5803s	$9.969 \cdot 10^{11}$	4.0882s	$1.069 \cdot 10^{12}$	0.6246s
	5	$7.968 \cdot 10^{11}$	4.2207s	$6.418 \cdot 10^{11}$	3.4476s	$6.538 \cdot 10^{11}$	0.2173s
	6	$1.096 \cdot 10^{12}$	4.5612s	$9.147 \cdot 10^{11}$	2.9364s	$9.704 \cdot 10^{11}$	0.1854s
	7	$8.602 \cdot 10^{11}$	4.0872s	$6.909 \cdot 10^{11}$	2.9805s	$7.377 \cdot 10^{11}$	0.1729s
25	2	$9.265 \cdot 10^{11}$	3.5851s	$7.160 \cdot 10^{11}$	4.7898s	$7.392 \cdot 10^{11}$	0.1692s
	3	$1.344 \cdot 10^{12}$	6.0896s	$7.792 \cdot 10^{11}$	3.5966s	$7.926 \cdot 10^{11}$	0.1942s
	4	$1.080 \cdot 10^{12}$	5.0119s	$8.189 \cdot 10^{11}$	3.3324s	$8.298 \cdot 10^{11}$	0.1825s
	5	$6.314 \cdot 10^{11}$	5.9078s	$5.154 \cdot 10^{11}$	4.4978s	$5.247 \cdot 10^{11}$	0.0938s
	6	$9.435 \cdot 10^{11}$	4.7721s	$7.495 \cdot 10^{11}$	3.5322s	$7.952 \cdot 10^{11}$	0.1983s
	7	$7.064 \cdot 10^{11}$	4.9476s	$5.629 \cdot 10^{11}$	3.2786s	$6.142 \cdot 10^{11}$	0.1758s

Table 14: Bank, algorithm 1

k	Sub	Algorithm 2		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.662 \cdot 10^{12}$	0.0321s	$1.662 \cdot 10^{12}$	1.6906s	$1.671 \cdot 10^{12}$	0.1625s
	3	$1.999 \cdot 10^{12}$	0.0410s	$1.872 \cdot 10^{12}$	3.3698s	$2.126 \cdot 10^{12}$	0.0818s
	4	$1.921 \cdot 10^{12}$	0.0274s	$1.823 \cdot 10^{12}$	2.1631s	$1.949 \cdot 10^{12}$	0.1698s
	5	$1.337 \cdot 10^{12}$	0.0280s	$1.241 \cdot 10^{12}$	2.2970s	$1.399 \cdot 10^{12}$	0.1662s
	6	$1.957 \cdot 10^{12}$	0.0415s	$1.637 \cdot 10^{12}$	2.1790s	$1.673 \cdot 10^{12}$	0.1272s
	7	$1.302 \cdot 10^{12}$	0.0283s	$1.271 \cdot 10^{12}$	2.0531s	$1.299 \cdot 10^{12}$	0.6063s
15	2	$1.227 \cdot 10^{12}$	0.0278s	$1.175 \cdot 10^{12}$	4.1048s	$1.201 \cdot 10^{12}$	0.0568s
	3	$1.408 \cdot 10^{12}$	0.0404s	$1.266 \cdot 10^{12}$	2.0093s	$1.339 \cdot 10^{12}$	0.1657s
	4	$1.387 \cdot 10^{12}$	0.0319s	$1.278 \cdot 10^{12}$	2.4505s	$1.353 \cdot 10^{12}$	0.1695s
	5	$9.702 \cdot 10^{11}$	0.0372s	$8.405 \cdot 10^{11}$	2.6075s	$9.259 \cdot 10^{11}$	0.0454s
	6	$1.583 \cdot 10^{12}$	0.0397s	$1.149 \cdot 10^{12}$	3.2106s	$1.297 \cdot 10^{12}$	0.1849s
	7	$9.239 \cdot 10^{11}$	0.0254s	$8.792 \cdot 10^{11}$	3.2607s	$9.071 \cdot 10^{11}$	0.1807s
20	2	$9.702 \cdot 10^{11}$	0.0425s	$8.979 \cdot 10^{11}$	2.5873s	$9.401 \cdot 10^{11}$	0.1727s
	3	$1.003 \cdot 10^{12}$	0.0411s	$9.670 \cdot 10^{11}$	4.0372s	$9.865 \cdot 10^{11}$	0.1868s
	4	$1.099 \cdot 10^{12}$	0.0515s	$9.969 \cdot 10^{11}$	3.4999s	$1.069 \cdot 10^{12}$	0.2018s
	5	$6.332 \cdot 10^{11}$	0.0525s	$6.418 \cdot 10^{11}$	3.8448s	$6.538 \cdot 10^{11}$	0.2145s
	6	$1.022 \cdot 10^{12}$	0.0606s	$9.147 \cdot 10^{11}$	2.5830s	$9.704 \cdot 10^{11}$	0.0561s
	7	$7.190 \cdot 10^{11}$	0.0436s	$6.909 \cdot 10^{11}$	4.4949s	$7.377 \cdot 10^{11}$	0.1571s
25	2	$8.453 \cdot 10^{11}$	0.0412s	$7.160 \cdot 10^{11}$	3.1548s	$7.392 \cdot 10^{11}$	0.0806s
	3	$8.791 \cdot 10^{11}$	0.0559s	$7.792 \cdot 10^{11}$	4.6227s	$7.926 \cdot 10^{11}$	0.0962s
	4	$9.438 \cdot 10^{11}$	0.0330s	$8.189 \cdot 10^{11}$	2.5400s	$8.298 \cdot 10^{11}$	0.0961s
	5	$5.654 \cdot 10^{11}$	0.0508s	$5.154 \cdot 10^{11}$	2.6711s	$5.247 \cdot 10^{11}$	0.1438s
	6	$8.528 \cdot 10^{11}$	0.0517s	$7.495 \cdot 10^{11}$	3.0872s	$7.952 \cdot 10^{11}$	0.1214s
	7	$6.172 \cdot 10^{11}$	0.0520s	$5.629 \cdot 10^{11}$	4.2804s	$6.142 \cdot 10^{11}$	0.0763s

Table 15: Bank, algorithm 2

k	Sub	Algorithm 2v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.662 \cdot 10^{12}$	0.0440s	$1.662 \cdot 10^{12}$	2.1032s	$1.671 \cdot 10^{12}$	0.1486s
	3	$2.155 \cdot 10^{12}$	0.0438s	$1.872 \cdot 10^{12}$	2.0378s	$2.126 \cdot 10^{12}$	0.1165s
	4	$1.899 \cdot 10^{12}$	0.0362s	$1.823 \cdot 10^{12}$	2.2032s	$1.949 \cdot 10^{12}$	0.1748s
	5	$1.242 \cdot 10^{12}$	0.0258s	$1.241 \cdot 10^{12}$	1.8622s	$1.399 \cdot 10^{12}$	0.0406s
	6	$1.726 \cdot 10^{12}$	0.0234s	$1.637 \cdot 10^{12}$	3.6489s	$1.673 \cdot 10^{12}$	0.1643s
	7	$1.302 \cdot 10^{12}$	0.0309s	$1.271 \cdot 10^{12}$	2.4014s	$1.299 \cdot 10^{12}$	0.1611s
15	2	$1.227 \cdot 10^{12}$	0.0270s	$1.175 \cdot 10^{12}$	2.2570s	$1.201 \cdot 10^{12}$	0.0576s
	3	$1.573 \cdot 10^{12}$	0.0264s	$1.266 \cdot 10^{12}$	2.5255s	$1.339 \cdot 10^{12}$	0.1649s
	4	$1.419 \cdot 10^{12}$	0.0304s	$1.278 \cdot 10^{12}$	3.5542s	$1.353 \cdot 10^{12}$	0.0657s
	5	$8.590 \cdot 10^{11}$	0.0324s	$8.405 \cdot 10^{11}$	2.3803s	$9.259 \cdot 10^{11}$	0.0581s
	6	$1.153 \cdot 10^{12}$	0.0785s	$1.149 \cdot 10^{12}$	2.5961s	$1.297 \cdot 10^{12}$	0.1209s
	7	$9.303 \cdot 10^{11}$	0.0367s	$8.792 \cdot 10^{11}$	2.7269s	$9.071 \cdot 10^{11}$	0.1972s
20	2	$9.702 \cdot 10^{11}$	0.1189s	$8.979 \cdot 10^{11}$	2.8458s	$9.401 \cdot 10^{11}$	0.0817s
	3	$1.125 \cdot 10^{12}$	0.0343s	$9.670 \cdot 10^{11}$	4.2435s	$9.865 \cdot 10^{11}$	0.1036s
	4	$1.102 \cdot 10^{12}$	0.0316s	$9.969 \cdot 10^{11}$	2.5973s	$1.069 \cdot 10^{12}$	0.1867s
	5	$6.434 \cdot 10^{11}$	0.0319s	$6.418 \cdot 10^{11}$	2.7076s	$6.538 \cdot 10^{11}$	0.1493s
	6	$9.717 \cdot 10^{11}$	0.0242s	$9.147 \cdot 10^{11}$	2.3870s	$9.704 \cdot 10^{11}$	0.0611s
	7	$7.097 \cdot 10^{11}$	0.0701s	$6.909 \cdot 10^{11}$	4.0600s	$7.377 \cdot 10^{11}$	0.0731s
25	2	$8.453 \cdot 10^{11}$	0.0399s	$7.160 \cdot 10^{11}$	2.8464s	$7.392 \cdot 10^{11}$	0.1220s
	3	$9.966 \cdot 10^{11}$	0.0627s	$7.792 \cdot 10^{11}$	4.7102s	$7.926 \cdot 10^{11}$	0.1327s
	4	$9.658 \cdot 10^{11}$	0.0384s	$8.189 \cdot 10^{11}$	2.8489s	$8.298 \cdot 10^{11}$	0.0717s
	5	$5.643 \cdot 10^{11}$	0.0418s	$5.154 \cdot 10^{11}$	2.8177s	$5.247 \cdot 10^{11}$	0.0742s
	6	$7.781 \cdot 10^{11}$	0.0540s	$7.495 \cdot 10^{11}$	3.1727s	$7.952 \cdot 10^{11}$	0.1093s
	7	$5.816 \cdot 10^{11}$	0.0733s	$5.629 \cdot 10^{11}$	4.6354s	$6.142 \cdot 10^{11}$	0.0752s

Table 16: Bank, variant algorithm 2

k	Sub	Algorithm 3		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.804 \cdot 10^{12}$	1.6089s	$1.662 \cdot 10^{12}$	1.7967s	$1.671 \cdot 10^{12}$	0.0573s
	3	$2.503 \cdot 10^{12}$	1.2280s	$1.872 \cdot 10^{12}$	1.7352s	$2.126 \cdot 10^{12}$	0.0558s
	4	$1.821 \cdot 10^{12}$	1.5021s	$1.823 \cdot 10^{12}$	1.9733s	$1.949 \cdot 10^{12}$	0.0690s
	5	$1.404 \cdot 10^{12}$	1.4241s	$1.241 \cdot 10^{12}$	1.9693s	$1.399 \cdot 10^{12}$	0.0973s
	6	$1.667 \cdot 10^{12}$	2.5542s	$1.637 \cdot 10^{12}$	3.5132s	$1.673 \cdot 10^{12}$	0.0756s
	7	$1.313 \cdot 10^{12}$	1.6029s	$1.271 \cdot 10^{12}$	2.2458s	$1.299 \cdot 10^{12}$	0.0739s
15	2	$1.197 \cdot 10^{12}$	2.9662s	$1.175 \cdot 10^{12}$	3.8648s	$1.201 \cdot 10^{12}$	0.0617s
	3	$1.898 \cdot 10^{12}$	1.3635s	$1.266 \cdot 10^{12}$	2.3286s	$1.339 \cdot 10^{12}$	0.0798s
	4	$1.324 \cdot 10^{12}$	1.8637s	$1.278 \cdot 10^{12}$	2.1982s	$1.353 \cdot 10^{12}$	0.0773s
	5	$1.107 \cdot 10^{12}$	1.9456s	$8.405 \cdot 10^{11}$	2.6478s	$9.259 \cdot 10^{11}$	0.1665s
	6	$1.242 \cdot 10^{12}$	1.3787s	$1.149 \cdot 10^{12}$	3.7292s	$1.297 \cdot 10^{12}$	0.1613s
	7	$9.061 \cdot 10^{11}$	1.9360s	$8.792 \cdot 10^{11}$	2.6425s	$9.071 \cdot 10^{11}$	0.1786s
20	2	$9.451 \cdot 10^{11}$	2.0114s	$8.979 \cdot 10^{11}$	2.5855s	$9.401 \cdot 10^{11}$	0.1831s
	3	$1.695 \cdot 10^{12}$	1.4484s	$9.670 \cdot 10^{11}$	2.4125s	$9.865 \cdot 10^{11}$	0.1759s
	4	$1.063 \cdot 10^{12}$	1.8690s	$9.969 \cdot 10^{11}$	4.0990s	$1.069 \cdot 10^{12}$	0.1972s
	5	$6.846 \cdot 10^{11}$	2.0765s	$6.418 \cdot 10^{11}$	2.9016s	$6.538 \cdot 10^{11}$	0.1938s
	6	$9.861 \cdot 10^{11}$	3.6528s	$9.147 \cdot 10^{11}$	2.4286s	$9.704 \cdot 10^{11}$	0.1630s
	7	$6.965 \cdot 10^{11}$	2.2619s	$6.909 \cdot 10^{11}$	2.3915s	$7.377 \cdot 10^{11}$	0.1231s
25	2	$7.497 \cdot 10^{11}$	2.1349s	$7.160 \cdot 10^{11}$	2.9494s	$7.392 \cdot 10^{11}$	0.0793s
	3	$1.079 \cdot 10^{12}$	2.1389s	$7.792 \cdot 10^{11}$	4.4733s	$7.926 \cdot 10^{11}$	0.1025s
	4	$9.143 \cdot 10^{11}$	2.3469s	$8.189 \cdot 10^{11}$	2.9419s	$8.298 \cdot 10^{11}$	0.0832s
	5	$5.845 \cdot 10^{11}$	3.5818s	$5.154 \cdot 10^{11}$	2.6275s	$5.247 \cdot 10^{11}$	0.0795s
	6	$8.487 \cdot 10^{11}$	2.3110s	$7.495 \cdot 10^{11}$	2.9831s	$7.952 \cdot 10^{11}$	0.1932s
	7	$6.107 \cdot 10^{11}$	2.7154s	$5.629 \cdot 10^{11}$	4.3728s	$6.142 \cdot 10^{11}$	0.0651s

Table 17: Bank, algorithm 3

k	Sub	Algorithm 4		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.662 \cdot 10^{12}$	0.0264s	$1.662 \cdot 10^{12}$	1.8876s	$1.671 \cdot 10^{12}$	0.0533s
	3	$1.935 \cdot 10^{12}$	0.0299s	$1.872 \cdot 10^{12}$	2.2563s	$2.126 \cdot 10^{12}$	0.0484s
	4	$2.040 \cdot 10^{12}$	0.0387s	$1.823 \cdot 10^{12}$	3.5076s	$1.949 \cdot 10^{12}$	0.1712s
	5	$1.294 \cdot 10^{12}$	0.0368s	$1.241 \cdot 10^{12}$	1.9543s	$1.399 \cdot 10^{12}$	0.1207s
	6	$1.869 \cdot 10^{12}$	0.0211s	$1.637 \cdot 10^{12}$	2.3007s	$1.673 \cdot 10^{12}$	0.0759s
	7	$1.342 \cdot 10^{12}$	0.0328s	$1.271 \cdot 10^{12}$	2.1488s	$1.299 \cdot 10^{12}$	0.0724s
15	2	$1.227 \cdot 10^{12}$	0.0285s	$1.175 \cdot 10^{12}$	2.6262s	$1.201 \cdot 10^{12}$	0.1634s
	3	$1.310 \cdot 10^{12}$	0.0416s	$1.266 \cdot 10^{12}$	2.9778s	$1.339 \cdot 10^{12}$	0.0695s
	4	$1.335 \cdot 10^{12}$	0.0366s	$1.278 \cdot 10^{12}$	2.6429s	$1.353 \cdot 10^{12}$	0.0775s
	5	$8.911 \cdot 10^{11}$	0.0419s	$8.405 \cdot 10^{11}$	2.7169s	$9.259 \cdot 10^{11}$	0.0526s
	6	$1.405 \cdot 10^{12}$	0.0403s	$1.149 \cdot 10^{12}$	2.5481s	$1.297 \cdot 10^{12}$	0.1937s
	7	$9.461 \cdot 10^{11}$	0.0740s	$8.792 \cdot 10^{11}$	2.4307s	$9.071 \cdot 10^{11}$	0.2066s
20	2	$9.702 \cdot 10^{11}$	0.5119s	$8.979 \cdot 10^{11}$	2.7327s	$9.401 \cdot 10^{11}$	0.1779s
	3	$1.084 \cdot 10^{12}$	0.0470s	$9.670 \cdot 10^{11}$	2.7300s	$9.865 \cdot 10^{11}$	0.1374s
	4	$1.127 \cdot 10^{12}$	0.0405s	$9.969 \cdot 10^{11}$	2.4957s	$1.069 \cdot 10^{12}$	0.1323s
	5	$7.715 \cdot 10^{11}$	0.0360s	$6.418 \cdot 10^{11}$	4.1645s	$6.538 \cdot 10^{11}$	0.1157s
	6	$1.081 \cdot 10^{12}$	0.0328s	$9.147 \cdot 10^{11}$	2.4348s	$9.704 \cdot 10^{11}$	0.1135s
	7	$7.470 \cdot 10^{11}$	0.0538s	$6.909 \cdot 10^{11}$	2.7502s	$7.377 \cdot 10^{11}$	0.1184s
25	2	$8.453 \cdot 10^{11}$	0.0391s	$7.160 \cdot 10^{11}$	2.9702s	$7.392 \cdot 10^{11}$	0.0753s
	3	$8.719 \cdot 10^{11}$	0.0519s	$7.792 \cdot 10^{11}$	4.4506s	$7.926 \cdot 10^{11}$	0.1987s
	4	$9.623 \cdot 10^{11}$	0.0896s	$8.189 \cdot 10^{11}$	2.4392s	$8.298 \cdot 10^{11}$	0.1679s
	5	$7.220 \cdot 10^{11}$	0.0488s	$5.154 \cdot 10^{11}$	2.8338s	$5.247 \cdot 10^{11}$	0.1874s
	6	$9.001 \cdot 10^{11}$	0.0478s	$7.495 \cdot 10^{11}$	2.8352s	$7.952 \cdot 10^{11}$	0.1071s
	7	$6.508 \cdot 10^{11}$	0.0492s	$5.629 \cdot 10^{11}$	4.4020s	$6.142 \cdot 10^{11}$	0.1777s

Table 18: Bank, algorithm 4

k	Sub	Algorithm 4v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.662 \cdot 10^{12}$	0.0293s	$1.662 \cdot 10^{12}$	2.1668s	$1.671 \cdot 10^{12}$	0.0671s
	3	$2.036 \cdot 10^{12}$	0.0327s	$1.872 \cdot 10^{12}$	1.6676s	$2.126 \cdot 10^{12}$	0.1334s
	4	$2.053 \cdot 10^{12}$	0.0388s	$1.823 \cdot 10^{12}$	1.9835s	$1.949 \cdot 10^{12}$	0.0711s
	5	$1.295 \cdot 10^{12}$	0.0357s	$1.241 \cdot 10^{12}$	2.2473s	$1.399 \cdot 10^{12}$	0.1182s
	6	$1.864 \cdot 10^{12}$	0.0337s	$1.637 \cdot 10^{12}$	2.9435s	$1.673 \cdot 10^{12}$	0.1734s
	7	$1.370 \cdot 10^{12}$	0.0506s	$1.271 \cdot 10^{12}$	2.3150s	$1.299 \cdot 10^{12}$	0.1441s
15	2	$1.227 \cdot 10^{12}$	0.0284s	$1.175 \cdot 10^{12}$	2.6417s	$1.201 \cdot 10^{12}$	0.0610s
	3	$1.428 \cdot 10^{12}$	0.0694s	$1.266 \cdot 10^{12}$	2.3827s	$1.339 \cdot 10^{12}$	0.1593s
	4	$1.658 \cdot 10^{12}$	0.0357s	$1.278 \cdot 10^{12}$	2.0011s	$1.353 \cdot 10^{12}$	0.1754s
	5	$9.183 \cdot 10^{11}$	0.0781s	$8.405 \cdot 10^{11}$	3.8391s	$9.259 \cdot 10^{11}$	0.1618s
	6	$1.378 \cdot 10^{12}$	0.0486s	$1.149 \cdot 10^{12}$	2.2708s	$1.297 \cdot 10^{12}$	0.2025s
	7	$1.075 \cdot 10^{12}$	0.0391s	$8.792 \cdot 10^{11}$	2.4558s	$9.071 \cdot 10^{11}$	0.1755s
20	2	$9.702 \cdot 10^{11}$	0.1135s	$8.979 \cdot 10^{11}$	2.8444s	$9.401 \cdot 10^{11}$	0.1892s
	3	$1.091 \cdot 10^{12}$	0.0897s	$9.670 \cdot 10^{11}$	2.6720s	$9.865 \cdot 10^{11}$	0.0916s
	4	$1.320 \cdot 10^{12}$	0.4213s	$9.969 \cdot 10^{11}$	2.4913s	$1.069 \cdot 10^{12}$	0.1246s
	5	$9.252 \cdot 10^{11}$	0.6681s	$6.418 \cdot 10^{11}$	4.1984s	$6.538 \cdot 10^{11}$	0.1539s
	6	$1.118 \cdot 10^{12}$	0.3421s	$9.147 \cdot 10^{11}$	2.3657s	$9.704 \cdot 10^{11}$	0.0593s
	7	$7.692 \cdot 10^{11}$	0.3468s	$6.909 \cdot 10^{11}$	2.4433s	$7.377 \cdot 10^{11}$	0.1814s
25	2	$8.453 \cdot 10^{11}$	0.0411s	$7.160 \cdot 10^{11}$	3.3865s	$7.392 \cdot 10^{11}$	0.2026s
	3	$9.007 \cdot 10^{11}$	0.0374s	$7.792 \cdot 10^{11}$	4.4917s	$7.926 \cdot 10^{11}$	0.1940s
	4	$9.874 \cdot 10^{11}$	0.0390s	$8.189 \cdot 10^{11}$	2.9654s	$8.298 \cdot 10^{11}$	0.1795s
	5	$6.807 \cdot 10^{11}$	0.0519s	$5.154 \cdot 10^{11}$	2.9158s	$5.247 \cdot 10^{11}$	0.0683s
	6	$9.296 \cdot 10^{11}$	0.0735s	$7.495 \cdot 10^{11}$	4.5999s	$7.952 \cdot 10^{11}$	0.1816s
	7	$6.525 \cdot 10^{11}$	0.0843s	$5.629 \cdot 10^{11}$	3.3045s	$6.142 \cdot 10^{11}$	0.1039s

Table 19: Bank, variant algorithm 4

k	Sub	Algorithm 5		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.662 \cdot 10^{12}$	0.0824s	$1.662 \cdot 10^{12}$	10.4023s	$1.671 \cdot 10^{12}$	0.1561s
	3	$2.156 \cdot 10^{12}$	3.2945s	$1.872 \cdot 10^{12}$	1.8928s	$2.126 \cdot 10^{12}$	0.1543s
	4	$1.921 \cdot 10^{12}$	3.8378s	$1.823 \cdot 10^{12}$	2.0928s	$1.949 \cdot 10^{12}$	0.1493s
	5	$1.338 \cdot 10^{12}$	6.0737s	$1.241 \cdot 10^{12}$	2.0247s	$1.399 \cdot 10^{12}$	0.1522s
	6	$1.677 \cdot 10^{12}$	5.0566s	$1.637 \cdot 10^{12}$	2.0199s	$1.673 \cdot 10^{12}$	0.1342s
	7	$1.298 \cdot 10^{12}$	7.0619s	$1.271 \cdot 10^{12}$	3.3739s	$1.299 \cdot 10^{12}$	0.1572s
15	2	$1.227 \cdot 10^{12}$	0.0281s	$1.175 \cdot 10^{12}$	15.1779s	$1.201 \cdot 10^{12}$	0.1726s
	3	$1.396 \cdot 10^{12}$	3.9073s	$1.266 \cdot 10^{12}$	2.1450s	$1.339 \cdot 10^{12}$	0.1744s
	4	$1.412 \cdot 10^{12}$	6.4425s	$1.278 \cdot 10^{12}$	2.1673s	$1.353 \cdot 10^{12}$	0.1708s
	5	$8.600 \cdot 10^{11}$	5.6422s	$8.405 \cdot 10^{11}$	2.5375s	$9.259 \cdot 10^{11}$	0.0451s
	6	$1.197 \cdot 10^{12}$	9.5812s	$1.149 \cdot 10^{12}$	3.6477s	$1.297 \cdot 10^{12}$	0.1688s
	7	$9.311 \cdot 10^{11}$	9.2394s	$8.792 \cdot 10^{11}$	2.4828s	$9.071 \cdot 10^{11}$	0.0817s
20	2	$9.702 \cdot 10^{11}$	0.0460s	$8.979 \cdot 10^{11}$	15.6232s	$9.401 \cdot 10^{11}$	0.0827s
	3	$1.053 \cdot 10^{12}$	4.3799s	$9.670 \cdot 10^{11}$	2.7623s	$9.865 \cdot 10^{11}$	0.1146s
	4	$1.019 \cdot 10^{12}$	6.7461s	$9.969 \cdot 10^{11}$	2.5086s	$1.069 \cdot 10^{12}$	0.2128s
	5	$6.411 \cdot 10^{11}$	8.6036s	$6.418 \cdot 10^{11}$	3.0744s	$6.538 \cdot 10^{11}$	0.1014s
	6	$9.662 \cdot 10^{11}$	8.9618s	$9.147 \cdot 10^{11}$	2.3764s	$9.704 \cdot 10^{11}$	0.0770s
	7	$7.467 \cdot 10^{11}$	11.1328s	$6.909 \cdot 10^{11}$	3.8068s	$7.377 \cdot 10^{11}$	0.0777s
25	2	$8.453 \cdot 10^{11}$	0.0405s	$7.160 \cdot 10^{11}$	18.8106s	$7.392 \cdot 10^{11}$	0.0707s
	3	$8.612 \cdot 10^{11}$	6.1339s	$7.792 \cdot 10^{11}$	2.9024s	$7.926 \cdot 10^{11}$	0.1482s
	4	$8.508 \cdot 10^{11}$	6.3158s	$8.189 \cdot 10^{11}$	2.5254s	$8.298 \cdot 10^{11}$	0.1893s
	5	$5.645 \cdot 10^{11}$	9.1494s	$5.154 \cdot 10^{11}$	2.8527s	$5.247 \cdot 10^{11}$	0.0667s
	6	$7.383 \cdot 10^{11}$	11.6670s	$7.495 \cdot 10^{11}$	4.3811s	$7.952 \cdot 10^{11}$	0.1778s
	7	$6.038 \cdot 10^{11}$	13.4492s	$5.629 \cdot 10^{11}$	2.7461s	$6.142 \cdot 10^{11}$	0.1680s

Table 20: Bank, algorithm 5

k	Sub	Algorithm 6		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$2.186 \cdot 10^{12}$	0.0662s	$1.662 \cdot 10^{12}$	1.9079s	$1.671 \cdot 10^{12}$	0.1231s
	3	$1.991 \cdot 10^{12}$	0.1542s	$1.872 \cdot 10^{12}$	1.6254s	$2.126 \cdot 10^{12}$	0.0751s
	4	$2.294 \cdot 10^{12}$	0.0821s	$1.823 \cdot 10^{12}$	2.2106s	$1.949 \cdot 10^{12}$	0.1246s
	5	$1.294 \cdot 10^{12}$	0.1553s	$1.241 \cdot 10^{12}$	3.0947s	$1.399 \cdot 10^{12}$	0.1307s
	6	$2.127 \cdot 10^{12}$	0.1597s	$1.637 \cdot 10^{12}$	1.5335s	$1.673 \cdot 10^{12}$	0.0775s
	7	$1.601 \cdot 10^{12}$	0.0799s	$1.271 \cdot 10^{12}$	2.0549s	$1.299 \cdot 10^{12}$	0.0744s
15	2	$1.504 \cdot 10^{12}$	0.1612s	$1.175 \cdot 10^{12}$	4.3544s	$1.201 \cdot 10^{12}$	0.0662s
	3	$1.581 \cdot 10^{12}$	0.1567s	$1.266 \cdot 10^{12}$	2.2708s	$1.339 \cdot 10^{12}$	0.1482s
	4	$1.538 \cdot 10^{12}$	0.1854s	$1.278 \cdot 10^{12}$	1.7953s	$1.353 \cdot 10^{12}$	0.0905s
	5	$9.396 \cdot 10^{11}$	0.1555s	$8.405 \cdot 10^{11}$	2.3572s	$9.259 \cdot 10^{11}$	0.0577s
	6	$1.390 \cdot 10^{12}$	0.0981s	$1.149 \cdot 10^{12}$	2.2543s	$1.297 \cdot 10^{12}$	0.1261s
	7	$1.059 \cdot 10^{12}$	0.1831s	$8.792 \cdot 10^{11}$	4.0958s	$9.071 \cdot 10^{11}$	0.1725s
20	2	$1.089 \cdot 10^{12}$	0.1583s	$8.979 \cdot 10^{11}$	2.9825s	$9.401 \cdot 10^{11}$	0.0787s
	3	$1.177 \cdot 10^{12}$	0.1743s	$9.670 \cdot 10^{11}$	3.9942s	$9.865 \cdot 10^{11}$	0.1364s
	4	$1.262 \cdot 10^{12}$	0.1587s	$9.969 \cdot 10^{11}$	2.2315s	$1.069 \cdot 10^{12}$	0.1260s
	5	$7.401 \cdot 10^{11}$	0.1797s	$6.418 \cdot 10^{11}$	3.0867s	$6.538 \cdot 10^{11}$	0.1652s
	6	$1.207 \cdot 10^{12}$	0.0945s	$9.147 \cdot 10^{11}$	2.4728s	$9.704 \cdot 10^{11}$	0.1691s
	7	$7.725 \cdot 10^{11}$	0.1487s	$6.909 \cdot 10^{11}$	4.1130s	$7.377 \cdot 10^{11}$	0.1689s
25	2	$8.297 \cdot 10^{11}$	0.1948s	$7.160 \cdot 10^{11}$	4.3310s	$7.392 \cdot 10^{11}$	0.0808s
	3	$9.972 \cdot 10^{11}$	0.1532s	$7.792 \cdot 10^{11}$	3.0104s	$7.926 \cdot 10^{11}$	0.1024s
	4	$1.083 \cdot 10^{12}$	0.1868s	$8.189 \cdot 10^{11}$	2.5700s	$8.298 \cdot 10^{11}$	0.0714s
	5	$6.826 \cdot 10^{11}$	0.1944s	$5.154 \cdot 10^{11}$	2.7652s	$5.247 \cdot 10^{11}$	0.0765s
	6	$1.079 \cdot 10^{12}$	0.1976s	$7.495 \cdot 10^{11}$	4.6282s	$7.952 \cdot 10^{11}$	0.2221s
	7	$7.788 \cdot 10^{11}$	0.2213s	$5.629 \cdot 10^{11}$	3.0129s	$6.142 \cdot 10^{11}$	0.0650s

Table 21: Bank, algorithm 6

5.2.1 Analysis

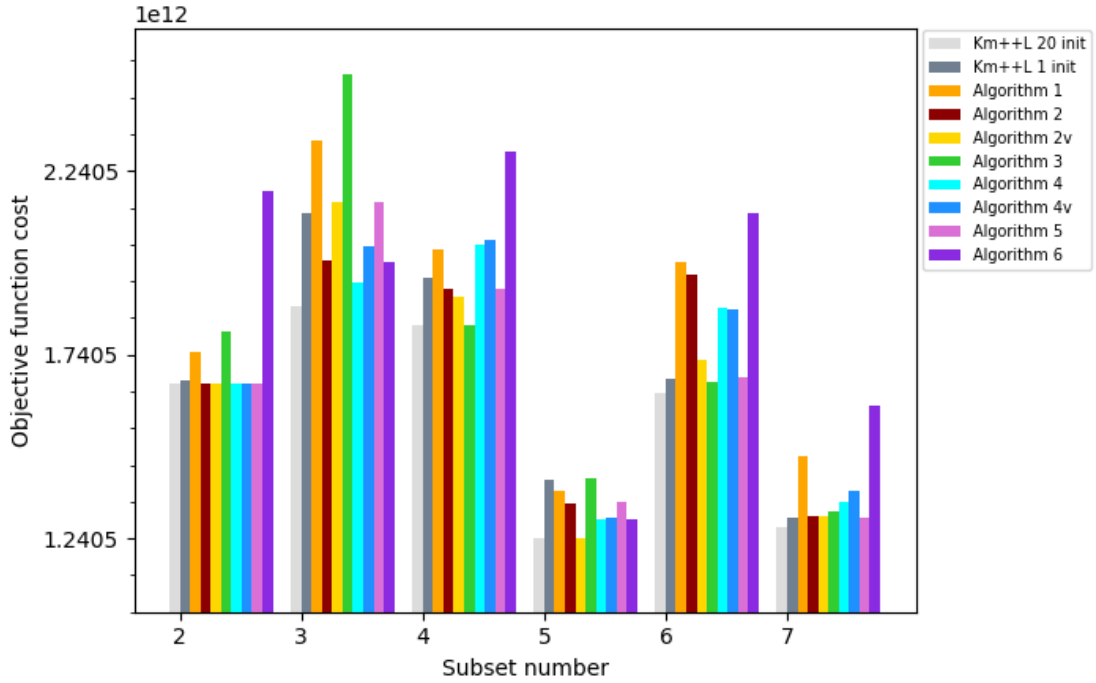

Figure 11: Bank, $k = 10$, objective function cost

Figure 11 shows the objective function cost obtained by each algorithm on each subset of the Bank dataset for $k = 10$. As opposed to the Oregon dataset, here we do not have a consistent behaviour of the algorithms throughout the subsets. For example when can look at algorithm 2v: compared to the baselines, on subsets 2, 5 and 7 it obtained relatively low costs, while on the remaining subsets the costs are higher. A similar reasoning can be applied for the other algorithms, except for algorithm 6, which is obviously the worst performer.

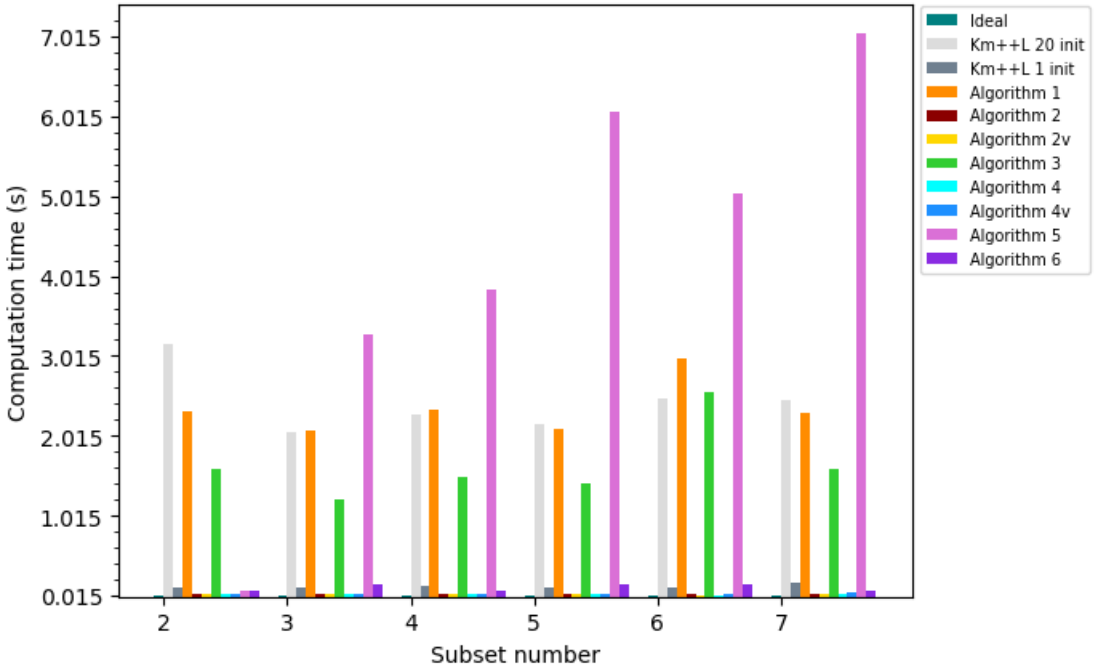

Figure 12: Bank, $k = 10$, computation time

Figure 12 shows the computing time of each algorithm on each subset of the Bank dataset for $k = 10$. Comparing figure 12 with figure 8 of the Oregon dataset makes it clear that the two plots are very similar to each other in terms of the distribution of the running times with respect to the subsets, and hence the analysis we made for the latter holds for the former too. This result seems to suggest that our algorithms are quite consistent even in datasets different from each others in terms of number of elements and of number of attributes.

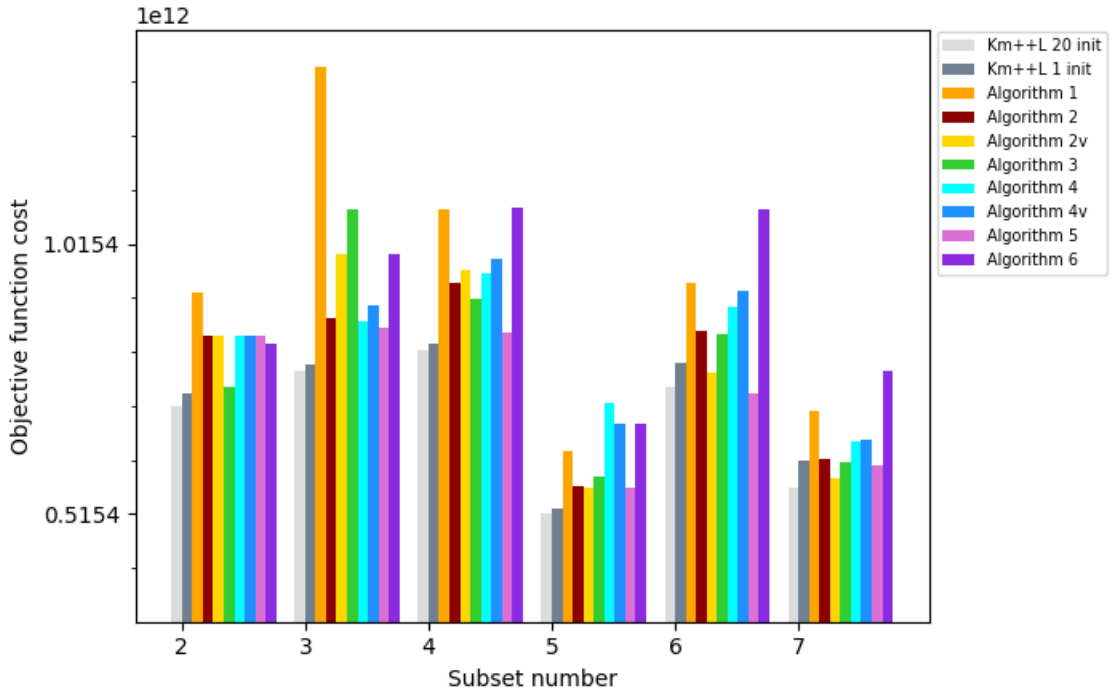

Figure 13: Bank, $k = 25$, objective function cost

Figure 13 shows the objective function cost obtained by each algorithm on each subset of the Bank dataset for $k = 25$. This higher value of k has significantly changed the results obtained by our algorithms: in general, they are more distant from the baselines compared to what we saw in figure 11. Algorithm 5 might be an exception to this, as for subsets from 3 to 7 it is the nearest to the baselines compared to the others.

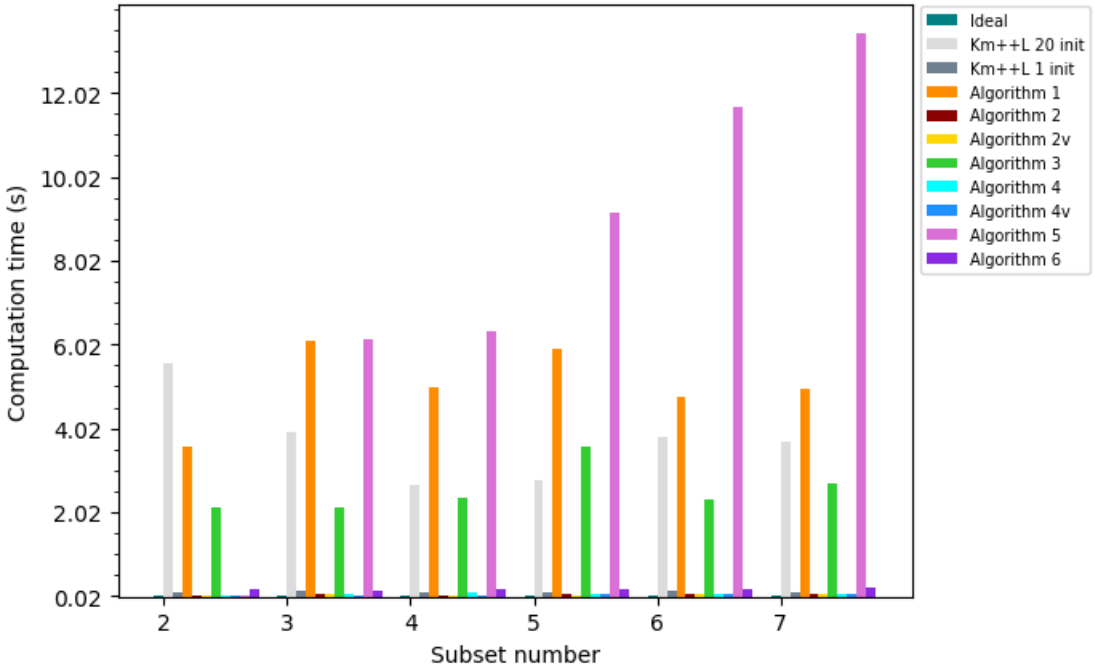

Figure 14: Bank, $k = 25$, computing time

Figure 14 shows the computing time of each algorithm on each subset of the Bank dataset for $k = 25$. Results are very similar to the ones found for $k = 10$ on the same dataset, depicted in figure 12, so the same considerations apply.

5.3 Power

We report here the results obtained on the Power dataset.

k	Sub	Algorithm 1		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.395 \cdot 10^{10}$	1.2529s	$9.390 \cdot 10^{10}$	2.5530s	$9.460 \cdot 10^{10}$	0.1798s
	3	$9.330 \cdot 10^{10}$	2.5955s	$9.303 \cdot 10^{10}$	2.1213s	$9.396 \cdot 10^{10}$	0.1466s
	4	$9.082 \cdot 10^{10}$	1.8778s	$9.074 \cdot 10^{10}$	3.4467s	$9.478 \cdot 10^{10}$	0.1576s
	5	$9.278 \cdot 10^{10}$	2.1364s	$9.267 \cdot 10^{10}$	3.0899s	$9.466 \cdot 10^{10}$	0.0347s
	6	$9.234 \cdot 10^{10}$	1.3180s	$9.228 \cdot 10^{10}$	2.0349s	$9.457 \cdot 10^{10}$	0.1403s
	7	$9.097 \cdot 10^{10}$	1.4254s	$9.068 \cdot 10^{10}$	2.2194s	$9.068 \cdot 10^{10}$	0.1456s
15	2	$6.786 \cdot 10^{10}$	1.6998s	$6.614 \cdot 10^{10}$	3.1450s	$6.752 \cdot 10^{10}$	0.1440s
	3	$6.736 \cdot 10^{10}$	2.1748s	$6.652 \cdot 10^{10}$	2.4280s	$6.901 \cdot 10^{10}$	0.0378s
	4	$6.552 \cdot 10^{10}$	2.1957s	$6.463 \cdot 10^{10}$	2.6537s	$6.711 \cdot 10^{10}$	0.1434s
	5	$6.737 \cdot 10^{10}$	2.1044s	$6.668 \cdot 10^{10}$	3.5414s	$6.759 \cdot 10^{10}$	0.0893s
	6	$6.592 \cdot 10^{10}$	2.1744s	$6.508 \cdot 10^{10}$	3.5249s	$6.548 \cdot 10^{10}$	0.0582s
	7	$6.490 \cdot 10^{10}$	2.9093s	$6.416 \cdot 10^{10}$	2.2401s	$6.460 \cdot 10^{10}$	0.0617s
20	2	$5.294 \cdot 10^{10}$	2.0556s	$5.161 \cdot 10^{10}$	2.7814s	$5.212 \cdot 10^{10}$	0.1400s
	3	$5.309 \cdot 10^{10}$	3.4309s	$5.265 \cdot 10^{10}$	2.8290s	$5.265 \cdot 10^{10}$	0.1344s
	4	$5.094 \cdot 10^{10}$	3.0807s	$5.040 \cdot 10^{10}$	3.6079s	$5.056 \cdot 10^{10}$	0.1811s
	5	$5.281 \cdot 10^{10}$	2.6970s	$5.208 \cdot 10^{10}$	3.2223s	$5.372 \cdot 10^{10}$	0.1411s
	6	$5.123 \cdot 10^{10}$	3.6288s	$5.095 \cdot 10^{10}$	2.5585s	$5.167 \cdot 10^{10}$	0.1521s
	7	$5.061 \cdot 10^{10}$	3.3541s	$5.032 \cdot 10^{10}$	2.6981s	$5.178 \cdot 10^{10}$	0.1399s
25	2	$4.311 \cdot 10^{10}$	3.5530s	$4.283 \cdot 10^{10}$	4.5835s	$4.304 \cdot 10^{10}$	0.1754s
	3	$4.405 \cdot 10^{10}$	3.5733s	$4.365 \cdot 10^{10}$	3.3224s	$4.428 \cdot 10^{10}$	0.1729s
	4	$4.272 \cdot 10^{10}$	4.4456s	$4.185 \cdot 10^{10}$	3.3593s	$4.251 \cdot 10^{10}$	0.1728s
	5	$4.405 \cdot 10^{10}$	3.0847s	$4.378 \cdot 10^{10}$	5.0905s	$4.407 \cdot 10^{10}$	0.1301s
	6	$4.245 \cdot 10^{10}$	3.6086s	$4.186 \cdot 10^{10}$	3.3323s	$4.233 \cdot 10^{10}$	0.8395s
	7	$4.240 \cdot 10^{10}$	3.6934s	$4.220 \cdot 10^{10}$	3.1681s	$4.220 \cdot 10^{10}$	0.6503s

Table 22: Power, algorithm 1

k	Sub	Algorithm 2		Km++L 20 init		Km++L 1 init	
		cost	time	cost	time	cost	time
10	2	$9.390 \cdot 10^{10}$	0.0134s	$9.390 \cdot 10^{10}$	2.6501s	$9.460 \cdot 10^{10}$	1.7297s
	3	$9.304 \cdot 10^{10}$	0.0606s	$9.303 \cdot 10^{10}$	2.1127s	$9.396 \cdot 10^{10}$	0.1416s
	4	$9.074 \cdot 10^{10}$	0.0171s	$9.074 \cdot 10^{10}$	6.4855s	$9.478 \cdot 10^{10}$	0.1561s
	5	$9.268 \cdot 10^{10}$	0.0214s	$9.267 \cdot 10^{10}$	2.4891s	$9.466 \cdot 10^{10}$	0.1390s
	6	$9.230 \cdot 10^{10}$	0.0141s	$9.228 \cdot 10^{10}$	2.5262s	$9.457 \cdot 10^{10}$	0.1206s
	7	$9.068 \cdot 10^{10}$	0.0162s	$9.068 \cdot 10^{10}$	2.1366s	$9.068 \cdot 10^{10}$	0.1580s
15	2	$6.757 \cdot 10^{10}$	0.0184s	$6.614 \cdot 10^{10}$	2.6350s	$6.752 \cdot 10^{10}$	0.3856s
	3	$6.692 \cdot 10^{10}$	0.0297s	$6.652 \cdot 10^{10}$	3.8375s	$6.901 \cdot 10^{10}$	0.1383s
	4	$6.514 \cdot 10^{10}$	0.0190s	$6.463 \cdot 10^{10}$	2.9782s	$6.711 \cdot 10^{10}$	0.1454s
	5	$6.689 \cdot 10^{10}$	0.0251s	$6.668 \cdot 10^{10}$	2.4664s	$6.759 \cdot 10^{10}$	0.1432s
	6	$6.560 \cdot 10^{10}$	0.0277s	$6.508 \cdot 10^{10}$	2.6215s	$6.548 \cdot 10^{10}$	0.0435s
	7	$6.457 \cdot 10^{10}$	0.0218s	$6.416 \cdot 10^{10}$	4.1292s	$6.460 \cdot 10^{10}$	0.0564s
20	2	$5.209 \cdot 10^{10}$	0.0267s	$5.161 \cdot 10^{10}$	2.8964s	$5.212 \cdot 10^{10}$	0.1510s
	3	$5.273 \cdot 10^{10}$	0.0243s	$5.265 \cdot 10^{10}$	4.6589s	$5.265 \cdot 10^{10}$	0.1452s
	4	$5.041 \cdot 10^{10}$	0.0365s	$5.040 \cdot 10^{10}$	2.7941s	$5.056 \cdot 10^{10}$	0.1907s
	5	$5.244 \cdot 10^{10}$	0.0278s	$5.208 \cdot 10^{10}$	2.7206s	$5.372 \cdot 10^{10}$	0.1447s
	6	$5.096 \cdot 10^{10}$	0.0217s	$5.095 \cdot 10^{10}$	4.3012s	$5.167 \cdot 10^{10}$	0.1709s
	7	$5.043 \cdot 10^{10}$	0.0189s	$5.032 \cdot 10^{10}$	2.7756s	$5.178 \cdot 10^{10}$	0.1374s
25	2	$4.277 \cdot 10^{10}$	0.0524s	$4.283 \cdot 10^{10}$	3.5297s	$4.304 \cdot 10^{10}$	0.1691s
	3	$4.384 \cdot 10^{10}$	0.3068s	$4.365 \cdot 10^{10}$	3.1415s	$4.428 \cdot 10^{10}$	0.1719s
	4	$4.250 \cdot 10^{10}$	0.1028s	$4.185 \cdot 10^{10}$	3.6562s	$4.251 \cdot 10^{10}$	0.1606s
	5	$4.376 \cdot 10^{10}$	0.0254s	$4.378 \cdot 10^{10}$	4.5426s	$4.407 \cdot 10^{10}$	0.1149s
	6	$4.193 \cdot 10^{10}$	0.0374s	$4.186 \cdot 10^{10}$	3.1605s	$4.233 \cdot 10^{10}$	0.5000s
	7	$4.213 \cdot 10^{10}$	0.0310s	$4.220 \cdot 10^{10}$	3.3054s	$4.220 \cdot 10^{10}$	0.5148s

Table 23: Power, algorithm 2

k	Sub	Algorithm 2v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.390 \cdot 10^{10}$	0.0134s	$9.390 \cdot 10^{10}$	2.4417s	$9.460 \cdot 10^{10}$	0.1884s
	3	$9.305 \cdot 10^{10}$	0.0140s	$9.303 \cdot 10^{10}$	2.3990s	$9.396 \cdot 10^{10}$	0.1393s
	4	$9.074 \cdot 10^{10}$	0.0159s	$9.074 \cdot 10^{10}$	4.2744s	$9.478 \cdot 10^{10}$	0.1568s
	5	$9.268 \cdot 10^{10}$	0.0149s	$9.267 \cdot 10^{10}$	2.5132s	$9.466 \cdot 10^{10}$	0.1570s
	6	$9.230 \cdot 10^{10}$	0.0144s	$9.228 \cdot 10^{10}$	2.3816s	$9.457 \cdot 10^{10}$	0.1404s
	7	$9.067 \cdot 10^{10}$	0.0167s	$9.068 \cdot 10^{10}$	2.4239s	$9.068 \cdot 10^{10}$	0.1537s
15	2	$6.757 \cdot 10^{10}$	0.0317s	$6.614 \cdot 10^{10}$	2.7786s	$6.752 \cdot 10^{10}$	0.1481s
	3	$6.690 \cdot 10^{10}$	0.0218s	$6.652 \cdot 10^{10}$	2.3109s	$6.901 \cdot 10^{10}$	0.0791s
	4	$6.516 \cdot 10^{10}$	0.0171s	$6.463 \cdot 10^{10}$	4.1288s	$6.711 \cdot 10^{10}$	0.1531s
	5	$6.698 \cdot 10^{10}$	0.0175s	$6.668 \cdot 10^{10}$	2.4778s	$6.759 \cdot 10^{10}$	0.1400s
	6	$6.553 \cdot 10^{10}$	0.0218s	$6.508 \cdot 10^{10}$	2.4321s	$6.548 \cdot 10^{10}$	0.0455s
	7	$6.464 \cdot 10^{10}$	0.0205s	$6.416 \cdot 10^{10}$	2.3627s	$6.460 \cdot 10^{10}$	0.1505s
20	2	$5.209 \cdot 10^{10}$	0.0696s	$5.161 \cdot 10^{10}$	2.8014s	$5.212 \cdot 10^{10}$	0.1428s
	3	$5.274 \cdot 10^{10}$	0.0249s	$5.265 \cdot 10^{10}$	4.5017s	$5.265 \cdot 10^{10}$	0.1572s
	4	$5.042 \cdot 10^{10}$	0.0191s	$5.040 \cdot 10^{10}$	2.8919s	$5.056 \cdot 10^{10}$	0.1832s
	5	$5.248 \cdot 10^{10}$	0.0214s	$5.208 \cdot 10^{10}$	2.8605s	$5.372 \cdot 10^{10}$	0.1422s
	6	$5.096 \cdot 10^{10}$	0.0227s	$5.095 \cdot 10^{10}$	4.2532s	$5.167 \cdot 10^{10}$	0.1670s
	7	$5.045 \cdot 10^{10}$	0.0180s	$5.032 \cdot 10^{10}$	2.8852s	$5.178 \cdot 10^{10}$	0.1439s
25	2	$4.277 \cdot 10^{10}$	0.0269s	$4.283 \cdot 10^{10}$	3.5253s	$4.304 \cdot 10^{10}$	0.1771s
	3	$4.381 \cdot 10^{10}$	0.0207s	$4.365 \cdot 10^{10}$	4.1825s	$4.428 \cdot 10^{10}$	0.1919s
	4	$4.235 \cdot 10^{10}$	0.0347s	$4.185 \cdot 10^{10}$	4.1718s	$4.251 \cdot 10^{10}$	0.1580s
	5	$4.392 \cdot 10^{10}$	0.0219s	$4.378 \cdot 10^{10}$	3.9675s	$4.407 \cdot 10^{10}$	0.1333s
	6	$4.226 \cdot 10^{10}$	0.0317s	$4.186 \cdot 10^{10}$	4.3113s	$4.233 \cdot 10^{10}$	0.1212s
	7	$4.223 \cdot 10^{10}$	0.0276s	$4.220 \cdot 10^{10}$	3.9428s	$4.220 \cdot 10^{10}$	0.0672s

Table 24: Power, variant algorithm 2

k	Sub	Algorithm 3		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.390 \cdot 10^{10}$	2.3334s	$9.390 \cdot 10^{10}$	2.7064s	$9.460 \cdot 10^{10}$	0.1897s
	3	$9.363 \cdot 10^{10}$	2.2499s	$9.303 \cdot 10^{10}$	4.0725s	$9.396 \cdot 10^{10}$	0.1494s
	4	$9.074 \cdot 10^{10}$	1.5375s	$9.074 \cdot 10^{10}$	2.4507s	$9.478 \cdot 10^{10}$	0.1501s
	5	$9.268 \cdot 10^{10}$	1.2136s	$9.267 \cdot 10^{10}$	2.4708s	$9.466 \cdot 10^{10}$	0.0389s
	6	$9.232 \cdot 10^{10}$	1.7729s	$9.228 \cdot 10^{10}$	2.0003s	$9.457 \cdot 10^{10}$	0.1402s
	7	$9.068 \cdot 10^{10}$	1.7552s	$9.068 \cdot 10^{10}$	3.6037s	$9.068 \cdot 10^{10}$	0.0667s
15	2	$6.653 \cdot 10^{10}$	1.7729s	$6.614 \cdot 10^{10}$	3.9845s	$6.752 \cdot 10^{10}$	0.1786s
	3	$6.709 \cdot 10^{10}$	3.7722s	$6.652 \cdot 10^{10}$	2.2118s	$6.901 \cdot 10^{10}$	0.0365s
	4	$6.465 \cdot 10^{10}$	1.9655s	$6.463 \cdot 10^{10}$	2.5197s	$6.711 \cdot 10^{10}$	0.1544s
	5	$6.775 \cdot 10^{10}$	1.9761s	$6.668 \cdot 10^{10}$	2.6940s	$6.759 \cdot 10^{10}$	0.1385s
	6	$6.521 \cdot 10^{10}$	1.8919s	$6.508 \cdot 10^{10}$	4.0065s	$6.548 \cdot 10^{10}$	0.1411s
	7	$6.416 \cdot 10^{10}$	2.0829s	$6.416 \cdot 10^{10}$	2.7557s	$6.460 \cdot 10^{10}$	0.1387s
20	2	$5.159 \cdot 10^{10}$	2.3256s	$5.161 \cdot 10^{10}$	2.7478s	$5.212 \cdot 10^{10}$	0.1382s
	3	$5.260 \cdot 10^{10}$	1.9249s	$5.265 \cdot 10^{10}$	4.3805s	$5.265 \cdot 10^{10}$	0.1518s
	4	$5.057 \cdot 10^{10}$	2.4974s	$5.040 \cdot 10^{10}$	2.8454s	$5.056 \cdot 10^{10}$	0.2074s
	5	$5.216 \cdot 10^{10}$	3.5283s	$5.208 \cdot 10^{10}$	2.7277s	$5.372 \cdot 10^{10}$	0.0941s
	6	$5.104 \cdot 10^{10}$	2.3749s	$5.095 \cdot 10^{10}$	4.3584s	$5.167 \cdot 10^{10}$	0.1637s
	7	$5.070 \cdot 10^{10}$	2.6940s	$5.032 \cdot 10^{10}$	2.8582s	$5.178 \cdot 10^{10}$	0.1383s
25	2	$4.277 \cdot 10^{10}$	2.3013s	$4.283 \cdot 10^{10}$	5.0240s	$4.304 \cdot 10^{10}$	0.1779s
	3	$4.372 \cdot 10^{10}$	2.2999s	$4.365 \cdot 10^{10}$	3.0012s	$4.428 \cdot 10^{10}$	0.1646s
	4	$4.212 \cdot 10^{10}$	3.8884s	$4.185 \cdot 10^{10}$	3.3953s	$4.251 \cdot 10^{10}$	0.1677s
	5	$4.381 \cdot 10^{10}$	2.5434s	$4.378 \cdot 10^{10}$	3.3558s	$4.407 \cdot 10^{10}$	0.1722s
	6	$4.188 \cdot 10^{10}$	2.5080s	$4.186 \cdot 10^{10}$	4.5312s	$4.233 \cdot 10^{10}$	0.1761s
	7	$4.211 \cdot 10^{10}$	2.0624s	$4.220 \cdot 10^{10}$	3.0412s	$4.220 \cdot 10^{10}$	0.1600s

Table 25: Power, algorithm 3

k	Sub	Algorithm 4		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.390 \cdot 10^{10}$	0.0134s	$9.390 \cdot 10^{10}$	2.5762s	$9.460 \cdot 10^{10}$	0.1665s
	3	$9.367 \cdot 10^{10}$	0.0337s	$9.303 \cdot 10^{10}$	2.4252s	$9.396 \cdot 10^{10}$	0.1492s
	4	$9.074 \cdot 10^{10}$	0.0319s	$9.074 \cdot 10^{10}$	3.9679s	$9.478 \cdot 10^{10}$	0.0686s
	5	$9.466 \cdot 10^{10}$	0.0241s	$9.267 \cdot 10^{10}$	2.2435s	$9.466 \cdot 10^{10}$	0.1347s
	6	$9.344 \cdot 10^{10}$	0.0222s	$9.228 \cdot 10^{10}$	2.1774s	$9.457 \cdot 10^{10}$	0.1184s
	7	$9.068 \cdot 10^{10}$	0.0243s	$9.068 \cdot 10^{10}$	2.3646s	$9.068 \cdot 10^{10}$	0.0601s
15	2	$6.757 \cdot 10^{10}$	0.0199s	$6.614 \cdot 10^{10}$	2.7758s	$6.752 \cdot 10^{10}$	0.1475s
	3	$7.004 \cdot 10^{10}$	0.0519s	$6.652 \cdot 10^{10}$	3.9013s	$6.901 \cdot 10^{10}$	0.0397s
	4	$6.776 \cdot 10^{10}$	0.0334s	$6.463 \cdot 10^{10}$	2.4782s	$6.711 \cdot 10^{10}$	0.1339s
	5	$7.078 \cdot 10^{10}$	0.0367s	$6.668 \cdot 10^{10}$	2.8291s	$6.759 \cdot 10^{10}$	0.1378s
	6	$6.745 \cdot 10^{10}$	0.0476s	$6.508 \cdot 10^{10}$	2.6502s	$6.548 \cdot 10^{10}$	0.1439s
	7	$6.890 \cdot 10^{10}$	0.0304s	$6.416 \cdot 10^{10}$	4.2212s	$6.460 \cdot 10^{10}$	0.0505s
20	2	$5.209 \cdot 10^{10}$	0.0265s	$5.161 \cdot 10^{10}$	4.9865s	$5.212 \cdot 10^{10}$	0.1483s
	3	$5.478 \cdot 10^{10}$	0.0457s	$5.265 \cdot 10^{10}$	2.6303s	$5.265 \cdot 10^{10}$	0.1624s
	4	$5.201 \cdot 10^{10}$	0.0484s	$5.040 \cdot 10^{10}$	2.5958s	$5.056 \cdot 10^{10}$	0.1901s
	5	$5.380 \cdot 10^{10}$	0.0363s	$5.208 \cdot 10^{10}$	2.7527s	$5.372 \cdot 10^{10}$	0.1424s
	6	$5.990 \cdot 10^{10}$	0.0701s	$5.095 \cdot 10^{10}$	4.3727s	$5.167 \cdot 10^{10}$	0.1462s
	7	$5.310 \cdot 10^{10}$	0.0677s	$5.032 \cdot 10^{10}$	2.6570s	$5.178 \cdot 10^{10}$	0.1176s
25	2	$4.277 \cdot 10^{10}$	0.0301s	$4.283 \cdot 10^{10}$	3.3523s	$4.304 \cdot 10^{10}$	0.1877s
	3	$4.421 \cdot 10^{10}$	0.2579s	$4.365 \cdot 10^{10}$	3.2667s	$4.428 \cdot 10^{10}$	0.1808s
	4	$4.386 \cdot 10^{10}$	0.0709s	$4.185 \cdot 10^{10}$	3.7562s	$4.251 \cdot 10^{10}$	0.1569s
	5	$4.684 \cdot 10^{10}$	0.0824s	$4.378 \cdot 10^{10}$	4.3565s	$4.407 \cdot 10^{10}$	0.0822s
	6	$4.640 \cdot 10^{10}$	0.0413s	$4.186 \cdot 10^{10}$	3.2447s	$4.233 \cdot 10^{10}$	0.3372s
	7	$4.365 \cdot 10^{10}$	0.0807s	$4.220 \cdot 10^{10}$	3.2339s	$4.220 \cdot 10^{10}$	0.7390s

Table 26: Power, algorithm 4

k	Sub	Algorithm 4v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.395 \cdot 10^{10}$	0.0223s	$9.390 \cdot 10^{10}$	2.5778s	$9.460 \cdot 10^{10}$	0.8486s
	3	$9.330 \cdot 10^{10}$	2.2983s	$9.303 \cdot 10^{10}$	3.7901s	$9.396 \cdot 10^{10}$	0.1475s
	4	$9.082 \cdot 10^{10}$	1.4744s	$9.074 \cdot 10^{10}$	2.4013s	$9.478 \cdot 10^{10}$	0.1546s
	5	$9.278 \cdot 10^{10}$	1.4917s	$9.267 \cdot 10^{10}$	2.6444s	$9.466 \cdot 10^{10}$	0.1043s
	6	$9.234 \cdot 10^{10}$	1.4419s	$9.228 \cdot 10^{10}$	2.3554s	$9.457 \cdot 10^{10}$	0.1443s
	7	$9.097 \cdot 10^{10}$	1.5495s	$9.068 \cdot 10^{10}$	3.0339s	$9.068 \cdot 10^{10}$	0.1487s
15	2	$6.786 \cdot 10^{10}$	0.0181s	$6.614 \cdot 10^{10}$	2.7893s	$6.752 \cdot 10^{10}$	0.7147s
	3	$6.736 \cdot 10^{10}$	2.0686s	$6.652 \cdot 10^{10}$	2.7108s	$6.901 \cdot 10^{10}$	0.5328s
	4	$6.552 \cdot 10^{10}$	2.1683s	$6.463 \cdot 10^{10}$	4.0654s	$6.711 \cdot 10^{10}$	0.2765s
	5	$6.737 \cdot 10^{10}$	3.3257s	$6.668 \cdot 10^{10}$	2.8106s	$6.759 \cdot 10^{10}$	0.2326s
	6	$6.592 \cdot 10^{10}$	2.2126s	$6.508 \cdot 10^{10}$	2.8476s	$6.548 \cdot 10^{10}$	0.1457s
	7	$6.490 \cdot 10^{10}$	2.1913s	$6.416 \cdot 10^{10}$	2.7309s	$6.460 \cdot 10^{10}$	0.1319s
20	2	$5.294 \cdot 10^{10}$	0.0269s	$5.161 \cdot 10^{10}$	4.2600s	$5.212 \cdot 10^{10}$	0.1521s
	3	$5.309 \cdot 10^{10}$	3.0954s	$5.265 \cdot 10^{10}$	2.8175s	$5.265 \cdot 10^{10}$	0.1419s
	4	$5.094 \cdot 10^{10}$	3.7965s	$5.040 \cdot 10^{10}$	2.8788s	$5.056 \cdot 10^{10}$	0.1959s
	5	$5.281 \cdot 10^{10}$	3.1198s	$5.208 \cdot 10^{10}$	2.8300s	$5.372 \cdot 10^{10}$	0.1468s
	6	$5.123 \cdot 10^{10}$	2.9057s	$5.095 \cdot 10^{10}$	4.3717s	$5.167 \cdot 10^{10}$	0.1586s
	7	$5.061 \cdot 10^{10}$	2.9921s	$5.032 \cdot 10^{10}$	2.9161s	$5.178 \cdot 10^{10}$	0.1405s
25	2	$4.311 \cdot 10^{10}$	0.0267s	$4.283 \cdot 10^{10}$	4.7323s	$4.304 \cdot 10^{10}$	0.1994s
	3	$4.405 \cdot 10^{10}$	3.3870s	$4.365 \cdot 10^{10}$	6.2385s	$4.428 \cdot 10^{10}$	0.1873s
	4	$4.272 \cdot 10^{10}$	3.3269s	$4.185 \cdot 10^{10}$	3.4785s	$4.251 \cdot 10^{10}$	0.1676s
	5	$4.405 \cdot 10^{10}$	4.3753s	$4.378 \cdot 10^{10}$	4.6219s	$4.407 \cdot 10^{10}$	0.1242s
	6	$4.245 \cdot 10^{10}$	3.9345s	$4.186 \cdot 10^{10}$	3.4472s	$4.233 \cdot 10^{10}$	0.1925s
	7	$4.240 \cdot 10^{10}$	3.4991s	$4.220 \cdot 10^{10}$	3.4049s	$4.220 \cdot 10^{10}$	0.1633s

Table 27: Power, variant algorithm 4

k	Sub	Algorithm 5		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.390 \cdot 10^{10}$	0.0133s	$9.390 \cdot 10^{10}$	8.9163s	$9.460 \cdot 10^{10}$	0.1733s
	3	$9.305 \cdot 10^{10}$	4.6318s	$9.303 \cdot 10^{10}$	2.3857s	$9.396 \cdot 10^{10}$	0.1492s
	4	$9.074 \cdot 10^{10}$	3.3456s	$9.074 \cdot 10^{10}$	2.2215s	$9.478 \cdot 10^{10}$	0.1557s
	5	$9.268 \cdot 10^{10}$	3.7650s	$9.267 \cdot 10^{10}$	4.2752s	$9.466 \cdot 10^{10}$	0.1390s
	6	$9.230 \cdot 10^{10}$	5.5367s	$9.228 \cdot 10^{10}$	2.2302s	$9.457 \cdot 10^{10}$	0.0444s
	7	$9.067 \cdot 10^{10}$	4.6315s	$9.068 \cdot 10^{10}$	2.1914s	$9.068 \cdot 10^{10}$	0.1419s
15	2	$6.757 \cdot 10^{10}$	0.0183s	$6.614 \cdot 10^{10}$	10.6788s	$6.752 \cdot 10^{10}$	0.0692s
	3	$6.655 \cdot 10^{10}$	4.2745s	$6.652 \cdot 10^{10}$	2.3570s	$6.901 \cdot 10^{10}$	0.1480s
	4	$6.463 \cdot 10^{10}$	4.3655s	$6.463 \cdot 10^{10}$	2.7455s	$6.711 \cdot 10^{10}$	0.1433s
	5	$6.673 \cdot 10^{10}$	5.9647s	$6.668 \cdot 10^{10}$	2.5353s	$6.759 \cdot 10^{10}$	0.1374s
	6	$6.530 \cdot 10^{10}$	5.4288s	$6.508 \cdot 10^{10}$	4.2649s	$6.548 \cdot 10^{10}$	0.1460s
	7	$6.419 \cdot 10^{10}$	7.8862s	$6.416 \cdot 10^{10}$	2.6498s	$6.460 \cdot 10^{10}$	0.1394s
20	2	$5.209 \cdot 10^{10}$	0.0271s	$5.161 \cdot 10^{10}$	11.7486s	$5.212 \cdot 10^{10}$	0.1526s
	3	$5.290 \cdot 10^{10}$	5.8043s	$5.265 \cdot 10^{10}$	2.7116s	$5.265 \cdot 10^{10}$	0.0840s
	4	$5.052 \cdot 10^{10}$	4.8023s	$5.040 \cdot 10^{10}$	2.8117s	$5.056 \cdot 10^{10}$	0.1661s
	5	$5.220 \cdot 10^{10}$	7.1202s	$5.208 \cdot 10^{10}$	4.2959s	$5.372 \cdot 10^{10}$	0.1532s
	6	$5.101 \cdot 10^{10}$	6.5507s	$5.095 \cdot 10^{10}$	2.6135s	$5.167 \cdot 10^{10}$	0.1467s
	7	$5.041 \cdot 10^{10}$	8.3832s	$5.032 \cdot 10^{10}$	2.6744s	$5.178 \cdot 10^{10}$	0.1396s
25	2	$4.277 \cdot 10^{10}$	0.0272s	$4.283 \cdot 10^{10}$	14.6878s	$4.304 \cdot 10^{10}$	0.3343s
	3	$4.368 \cdot 10^{10}$	4.2875s	$4.365 \cdot 10^{10}$	2.8346s	$4.428 \cdot 10^{10}$	0.1658s
	4	$4.196 \cdot 10^{10}$	6.8369s	$4.185 \cdot 10^{10}$	4.7210s	$4.251 \cdot 10^{10}$	0.1108s
	5	$4.365 \cdot 10^{10}$	6.2735s	$4.378 \cdot 10^{10}$	3.1973s	$4.407 \cdot 10^{10}$	0.1798s
	6	$4.185 \cdot 10^{10}$	9.0110s	$4.186 \cdot 10^{10}$	3.2850s	$4.233 \cdot 10^{10}$	0.1616s
	7	$4.194 \cdot 10^{10}$	9.8543s	$4.220 \cdot 10^{10}$	4.6555s	$4.220 \cdot 10^{10}$	0.1802s

Table 28: Power, algorithm 5

k	Sub	Algorithm 6		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$9.460 \cdot 10^{10}$	0.1642s	$9.390 \cdot 10^{10}$	3.9463s	$9.460 \cdot 10^{10}$	0.1847s
	3	$9.304 \cdot 10^{10}$	0.1336s	$9.303 \cdot 10^{10}$	2.0110s	$9.396 \cdot 10^{10}$	0.1425s
	4	$9.074 \cdot 10^{10}$	0.1310s	$9.074 \cdot 10^{10}$	2.3477s	$9.478 \cdot 10^{10}$	0.1407s
	5	$9.465 \cdot 10^{10}$	0.1348s	$9.267 \cdot 10^{10}$	2.6320s	$9.466 \cdot 10^{10}$	0.0332s
	6	$9.241 \cdot 10^{10}$	0.1342s	$9.228 \cdot 10^{10}$	2.3074s	$9.457 \cdot 10^{10}$	0.1385s
	7	$9.068 \cdot 10^{10}$	0.1318s	$9.068 \cdot 10^{10}$	3.4710s	$9.068 \cdot 10^{10}$	0.1414s
15	2	$6.855 \cdot 10^{10}$	0.0537s	$6.614 \cdot 10^{10}$	4.2893s	$6.752 \cdot 10^{10}$	0.1411s
	3	$6.693 \cdot 10^{10}$	0.1627s	$6.652 \cdot 10^{10}$	2.5453s	$6.901 \cdot 10^{10}$	0.1326s
	4	$6.501 \cdot 10^{10}$	0.0990s	$6.463 \cdot 10^{10}$	2.4823s	$6.711 \cdot 10^{10}$	0.1482s
	5	$6.704 \cdot 10^{10}$	0.1556s	$6.668 \cdot 10^{10}$	2.7211s	$6.759 \cdot 10^{10}$	0.1363s
	6	$6.598 \cdot 10^{10}$	0.1404s	$6.508 \cdot 10^{10}$	4.2151s	$6.548 \cdot 10^{10}$	0.1328s
	7	$6.481 \cdot 10^{10}$	0.1330s	$6.416 \cdot 10^{10}$	2.7773s	$6.460 \cdot 10^{10}$	0.1233s
20	2	$5.215 \cdot 10^{10}$	0.1482s	$5.161 \cdot 10^{10}$	2.6542s	$5.212 \cdot 10^{10}$	0.1616s
	3	$5.296 \cdot 10^{10}$	0.3671s	$5.265 \cdot 10^{10}$	2.7338s	$5.265 \cdot 10^{10}$	0.1599s
	4	$5.069 \cdot 10^{10}$	0.1353s	$5.040 \cdot 10^{10}$	2.6841s	$5.056 \cdot 10^{10}$	0.1708s
	5	$5.300 \cdot 10^{10}$	0.1391s	$5.208 \cdot 10^{10}$	4.1297s	$5.372 \cdot 10^{10}$	0.1539s
	6	$5.176 \cdot 10^{10}$	0.1545s	$5.095 \cdot 10^{10}$	2.3763s	$5.167 \cdot 10^{10}$	0.1459s
	7	$5.301 \cdot 10^{10}$	0.1331s	$5.032 \cdot 10^{10}$	2.8734s	$5.178 \cdot 10^{10}$	0.1501s
25	2	$4.382 \cdot 10^{10}$	0.1904s	$4.283 \cdot 10^{10}$	3.4434s	$4.304 \cdot 10^{10}$	0.1758s
	3	$4.395 \cdot 10^{10}$	0.1898s	$4.365 \cdot 10^{10}$	4.7762s	$4.428 \cdot 10^{10}$	0.0883s
	4	$4.487 \cdot 10^{10}$	0.1969s	$4.185 \cdot 10^{10}$	3.3073s	$4.251 \cdot 10^{10}$	0.1321s
	5	$4.520 \cdot 10^{10}$	0.1972s	$4.378 \cdot 10^{10}$	2.8520s	$4.407 \cdot 10^{10}$	0.0739s
	6	$4.257 \cdot 10^{10}$	0.1900s	$4.186 \cdot 10^{10}$	4.9829s	$4.233 \cdot 10^{10}$	0.0823s
	7	$4.258 \cdot 10^{10}$	0.1773s	$4.220 \cdot 10^{10}$	3.1881s	$4.220 \cdot 10^{10}$	0.1762s

Table 29: Power, algorithm 6

5.3.1 Analysis

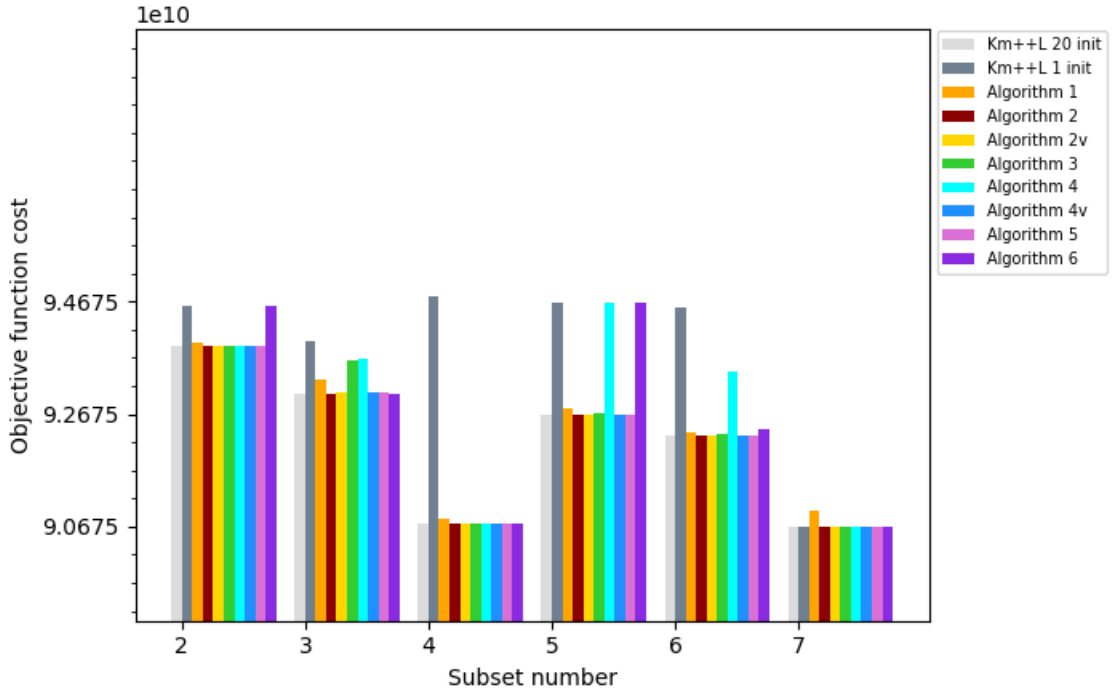

Figure 15: Power, $k = 10$, objective function cost

Figure 15 shows the objective function cost obtained by each algorithm on each subset of the Power dataset for $k = 10$. For this dataset the results obtained by our algorithms are in the most cases evidently homogeneous, and differences in costs within the same subset are almost negligible; this does not generally apply to algorithms 4 and 6, especially if we consider for example subset number 5. In any case, their performances are near those of the baselines, in particular of `Km++L` 20 init.

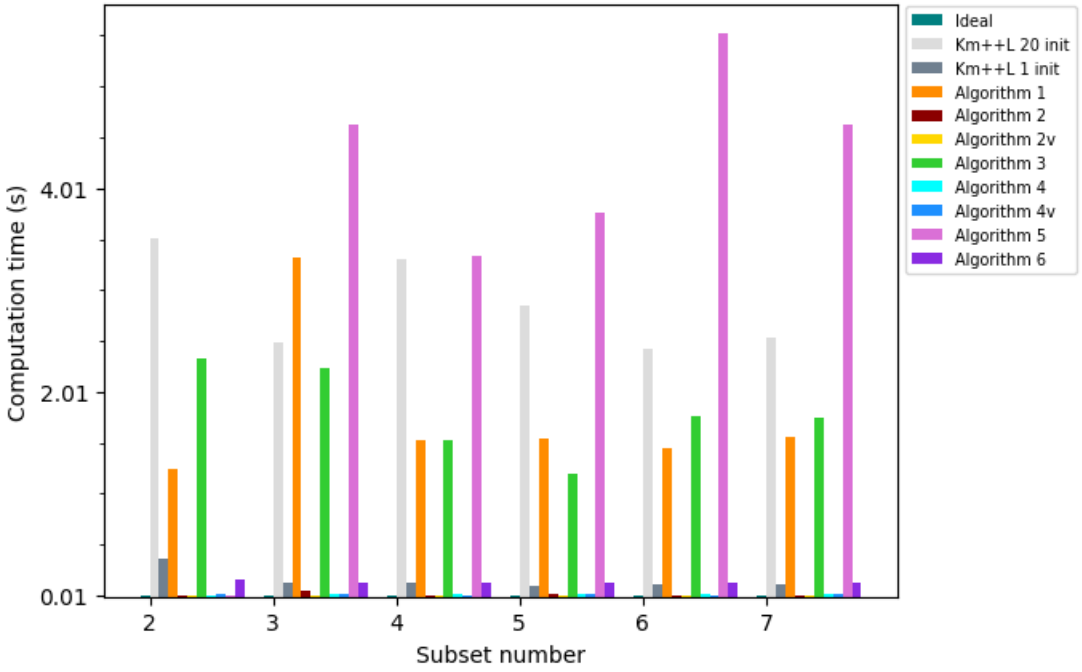

Figure 16: Power, $k = 10$, computation time

Figure 16 shows the computing time of each algorithm on each subset of the Power dataset for $k = 10$. Results are very similar to the ones found for the Oregon dataset, depicted in figure 8, so the same considerations apply.

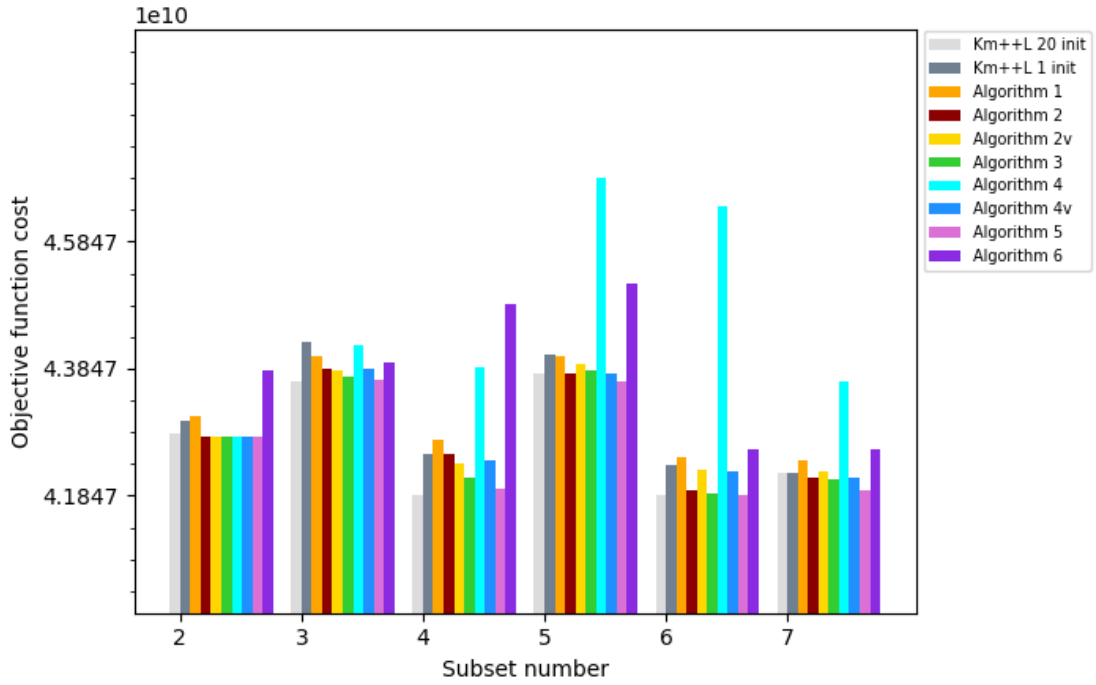

Figure 17: Power, $k = 25$, objective function cost

Figure 17 shows the objective function cost obtained by each algorithm on each subset of the Power dataset for $k = 25$. Compared to the case with $k = 10$ the situation is quite different. In almost all the 6 subsets all our algorithms, except for 1, 4 and 6, performed better than the 1 init baseline. In some cases they even outperformed the 20 init baseline, as it can be seen on subset 5 (algorithms 2 and 5) and on subset 7 (algorithms 2, 3, 4v and 5).

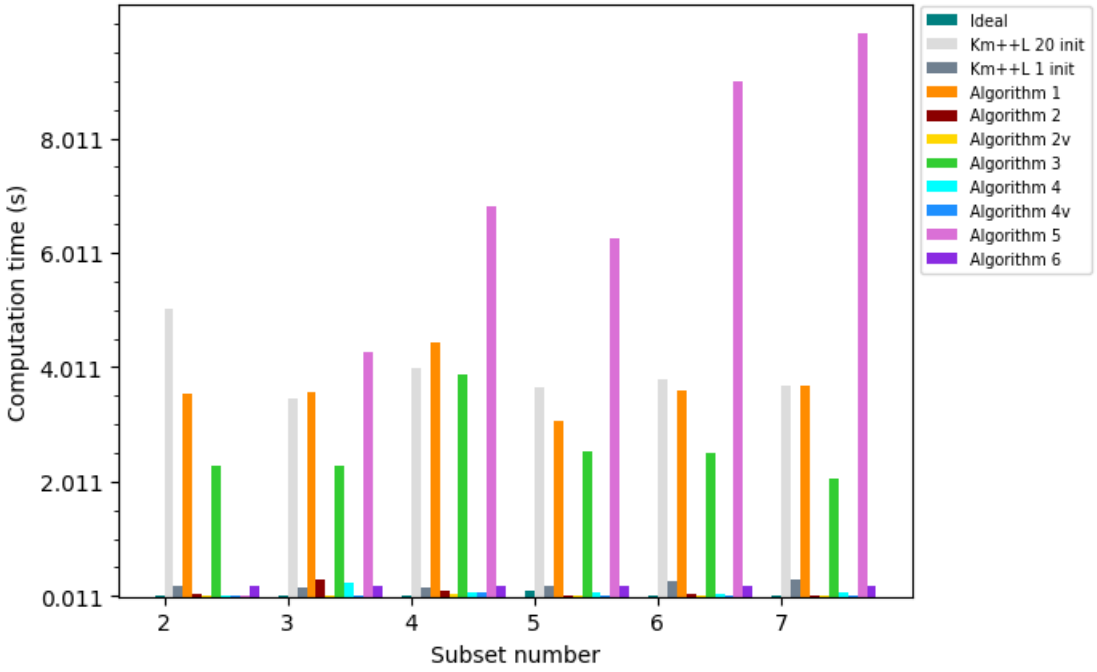

Figure 18: Power, $k = 25$, computation time

Figure 18 shows the computing time of each algorithm on each subset of the Power dataset for $k = 10$. Results are very similar to the ones found for the Oregon dataset, depicted in figure 8, so the same considerations apply.

5.4 GPS-ephemeris

We report here the results obtained on the GPS-ephemeris dataset.

k	Sub	Algorithm 1		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.918 \cdot 10^5$	6.4287s	$1.860 \cdot 10^5$	5.2266s	$1.877 \cdot 10^5$	0.3246s
	3	$3.428 \cdot 10^5$	8.1608s	$3.015 \cdot 10^5$	6.9633s	$3.015 \cdot 10^5$	0.2537s
	4	$4.662 \cdot 10^5$	8.9010s	$3.874 \cdot 10^5$	4.8792s	$3.874 \cdot 10^5$	0.3119s
	5	$2.738 \cdot 10^5$	8.5633s	$2.577 \cdot 10^5$	6.7787s	$2.584 \cdot 10^5$	0.2573s
	6	$2.867 \cdot 10^5$	8.2817s	$2.702 \cdot 10^5$	5.5232s	$2.739 \cdot 10^5$	0.3458s
	7	$2.767 \cdot 10^5$	8.6836s	$2.570 \cdot 10^5$	6.9721s	$2.570 \cdot 10^5$	0.3799s
	8	$3.185 \cdot 10^8$	8.9276s	$3.689 \cdot 10^7$	4.1672s	$3.805 \cdot 10^7$	0.2470s
15	2	$1.576 \cdot 10^5$	10.019s	$1.477 \cdot 10^5$	8.0400s	$1.479 \cdot 10^5$	0.5314s
	3	$2.670 \cdot 10^5$	13.1826s	$2.360 \cdot 10^5$	8.5980s	$2.373 \cdot 10^5$	0.2999s
	4	$3.454 \cdot 10^5$	13.1159s	$2.993 \cdot 10^5$	8.1480s	$3.083 \cdot 10^5$	0.5229s
	5	$2.154 \cdot 10^5$	12.9675s	$1.994 \cdot 10^5$	6.3608s	$2.001 \cdot 10^5$	0.2818s
	6	$2.250 \cdot 10^5$	12.7954s	$2.101 \cdot 10^5$	10.0691s	$2.101 \cdot 10^5$	1.9944s
	7	$2.209 \cdot 10^5$	12.5764s	$2.073 \cdot 10^5$	8.7152s	$2.076 \cdot 10^5$	0.7385s
	8	$2.951 \cdot 10^8$	13.0471s	$2.230 \cdot 10^7$	5.2693s	$2.363 \cdot 10^7$	0.2897s
20	2	$1.354 \cdot 10^5$	12.071s	$1.276 \cdot 10^5$	10.3371s	$1.280 \cdot 10^5$	0.3667s
	3	$2.297 \cdot 10^5$	15.5997s	$2.023 \cdot 10^5$	9.6622s	$2.024 \cdot 10^5$	0.3552s
	4	$2.923 \cdot 10^5$	15.8529s	$2.549 \cdot 10^5$	10.0579s	$2.551 \cdot 10^5$	0.3577s
	5	$1.824 \cdot 10^5$	15.1845s	$1.709 \cdot 10^5$	8.2272s	$1.716 \cdot 10^5$	0.3737s
	6	$1.949 \cdot 10^5$	16.5283s	$1.806 \cdot 10^5$	9.8800s	$1.817 \cdot 10^5$	0.5080s
	7	$1.930 \cdot 10^5$	15.8445s	$1.797 \cdot 10^5$	11.0988s	$1.799 \cdot 10^5$	0.4055s
	8	$2.882 \cdot 10^8$	16.6014s	$1.754 \cdot 10^7$	7.4771s	$1.826 \cdot 10^7$	0.2983s
25	2	$1.247 \cdot 10^5$	14.461s	$1.154 \cdot 10^5$	13.3255s	$1.159 \cdot 10^5$	0.6202s
	3	$2.117 \cdot 10^5$	20.0929s	$1.827 \cdot 10^5$	12.5683s	$1.837 \cdot 10^5$	0.4418s
	4	$2.644 \cdot 10^5$	18.9711s	$2.263 \cdot 10^5$	10.1300s	$2.274 \cdot 10^5$	0.4799s
	5	$1.638 \cdot 10^5$	19.7051s	$1.531 \cdot 10^5$	11.9094s	$1.533 \cdot 10^5$	0.4739s
	6	$1.759 \cdot 10^5$	19.2409s	$1.624 \cdot 10^5$	11.6243s	$1.628 \cdot 10^5$	0.6300s
	7	$1.734 \cdot 10^5$	20.2179s	$1.637 \cdot 10^5$	11.8710s	$1.647 \cdot 10^5$	0.3383s
	8	$2.872 \cdot 10^8$	19.2562s	$1.401 \cdot 10^7$	8.8673s	$1.461 \cdot 10^7$	0.6294s

Table 30: GPS-ephemeris, algorithm 1

k	Sub	Algorithm 2		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.862 \cdot 10^5$	0.1446s	$1.860 \cdot 10^5$	5.1850s	$1.877 \cdot 10^5$	0.3250s
	3	$3.036 \cdot 10^5$	0.1625s	$3.015 \cdot 10^5$	6.7929s	$3.015 \cdot 10^5$	0.2549s
	4	$3.874 \cdot 10^5$	0.1742s	$3.874 \cdot 10^5$	4.7970s	$3.874 \cdot 10^5$	0.2842s
	5	$2.577 \cdot 10^5$	0.1114s	$2.577 \cdot 10^5$	6.7316s	$2.584 \cdot 10^5$	0.2256s
	6	$2.704 \cdot 10^5$	0.1725s	$2.702 \cdot 10^5$	5.4825s	$2.739 \cdot 10^5$	0.3570s
	7	$2.579 \cdot 10^5$	0.2886s	$2.570 \cdot 10^5$	6.8814s	$2.570 \cdot 10^5$	0.9516s
	8	$3.724 \cdot 10^7$	0.2757s	$3.689 \cdot 10^7$	4.2325s	$3.805 \cdot 10^7$	0.2529s
15	2	$1.479 \cdot 10^5$	0.1829s	$1.477 \cdot 10^5$	9.9552s	$1.479 \cdot 10^5$	0.5299s
	3	$2.372 \cdot 10^5$	0.3712s	$2.360 \cdot 10^5$	7.0034s	$2.373 \cdot 10^5$	0.2421s
	4	$2.993 \cdot 10^5$	0.3556s	$2.993 \cdot 10^5$	8.0039s	$3.083 \cdot 10^5$	2.0840s
	5	$2.028 \cdot 10^5$	0.1968s	$1.994 \cdot 10^5$	6.3660s	$2.001 \cdot 10^5$	0.3618s
	6	$2.108 \cdot 10^5$	0.3208s	$2.101 \cdot 10^5$	10.0419s	$2.101 \cdot 10^5$	0.5892s
	7	$2.084 \cdot 10^5$	0.2638s	$2.073 \cdot 10^5$	8.7246s	$2.076 \cdot 10^5$	0.4296s
	8	$2.421 \cdot 10^7$	0.3120s	$2.230 \cdot 10^7$	5.1066s	$2.363 \cdot 10^7$	0.2594s
20	2	$1.283 \cdot 10^5$	0.3028s	$1.276 \cdot 10^5$	8.5762s	$1.280 \cdot 10^5$	0.3743s
	3	$2.023 \cdot 10^5$	0.2990s	$2.023 \cdot 10^5$	9.6261s	$2.024 \cdot 10^5$	0.3463s
	4	$2.554 \cdot 10^5$	0.2499s	$2.549 \cdot 10^5$	10.1017s	$2.551 \cdot 10^5$	0.3707s
	5	$1.716 \cdot 10^5$	0.2099s	$1.709 \cdot 10^5$	9.8148s	$1.716 \cdot 10^5$	0.3596s
	6	$1.811 \cdot 10^5$	1.4502s	$1.806 \cdot 10^5$	9.9947s	$1.817 \cdot 10^5$	0.5224s
	7	$1.797 \cdot 10^5$	1.0821s	$1.797 \cdot 10^5$	9.3499s	$1.799 \cdot 10^5$	0.4107s
	8	$1.810 \cdot 10^7$	0.6538s	$1.754 \cdot 10^7$	7.4853s	$1.826 \cdot 10^7$	0.2524s
25	2	$1.165 \cdot 10^5$	1.0305s	$1.154 \cdot 10^5$	12.5098s	$1.159 \cdot 10^5$	0.6236s
	3	$1.843 \cdot 10^5$	0.2807s	$1.827 \cdot 10^5$	12.3843s	$1.837 \cdot 10^5$	0.4192s
	4	$2.289 \cdot 10^5$	0.3032s	$2.263 \cdot 10^5$	11.5611s	$2.274 \cdot 10^5$	0.4703s
	5	$1.540 \cdot 10^5$	0.3115s	$1.531 \cdot 10^5$	12.0333s	$1.533 \cdot 10^5$	0.4755s
	6	$1.633 \cdot 10^5$	0.3471s	$1.624 \cdot 10^5$	11.5595s	$1.628 \cdot 10^5$	0.6845s
	7	$1.639 \cdot 10^5$	0.5239s	$1.637 \cdot 10^5$	11.8783s	$1.647 \cdot 10^5$	0.3063s
	8	$1.555 \cdot 10^7$	0.7486s	$1.401 \cdot 10^7$	9.0215s	$1.461 \cdot 10^7$	0.6352s

Table 31: GPS-ephemeris, algorithm 2

k	Sub	Algorithm 2v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.862 \cdot 10^5$	0.1202s	$1.860 \cdot 10^5$	6.9034s	$1.877 \cdot 10^5$	1.6439s
	3	$3.036 \cdot 10^5$	0.2673s	$3.015 \cdot 10^5$	5.2232s	$3.015 \cdot 10^5$	0.5642s
	4	$3.954 \cdot 10^5$	0.1270s	$3.874 \cdot 10^5$	6.5068s	$3.874 \cdot 10^5$	0.2938s
	5	$2.583 \cdot 10^5$	0.2090s	$2.577 \cdot 10^5$	5.2087s	$2.584 \cdot 10^5$	0.2628s
	6	$2.715 \cdot 10^5$	0.1656s	$2.702 \cdot 10^5$	7.0878s	$2.739 \cdot 10^5$	0.3702s
	7	$2.624 \cdot 10^5$	0.1671s	$2.570 \cdot 10^5$	5.5030s	$2.570 \cdot 10^5$	0.3705s
	8	$3.724 \cdot 10^7$	0.2717s	$3.689 \cdot 10^7$	4.2396s	$3.805 \cdot 10^7$	0.2359s
15	2	$1.479 \cdot 10^5$	0.1789s	$1.477 \cdot 10^5$	9.8189s	$1.479 \cdot 10^5$	0.5045s
	3	$2.372 \cdot 10^5$	0.1200s	$2.360 \cdot 10^5$	8.6291s	$2.373 \cdot 10^5$	0.2862s
	4	$3.085 \cdot 10^5$	0.1445s	$2.993 \cdot 10^5$	8.0334s	$3.083 \cdot 10^5$	0.5231s
	5	$1.994 \cdot 10^5$	0.1716s	$1.994 \cdot 10^5$	6.3897s	$2.001 \cdot 10^5$	0.2654s
	6	$2.115 \cdot 10^5$	0.1717s	$2.101 \cdot 10^5$	10.1333s	$2.101 \cdot 10^5$	0.6113s
	7	$2.084 \cdot 10^5$	0.2073s	$2.073 \cdot 10^5$	8.7724s	$2.076 \cdot 10^5$	0.4520s
	8	$2.363 \cdot 10^7$	0.4023s	$2.230 \cdot 10^7$	5.0853s	$2.363 \cdot 10^7$	0.2272s
20	2	$1.283 \cdot 10^5$	0.3006s	$1.276 \cdot 10^5$	10.2394s	$1.280 \cdot 10^5$	0.3951s
	3	$2.027 \cdot 10^5$	0.2271s	$2.023 \cdot 10^5$	9.7020s	$2.024 \cdot 10^5$	0.3528s
	4	$2.550 \cdot 10^5$	0.4038s	$2.549 \cdot 10^5$	10.1740s	$2.551 \cdot 10^5$	0.3661s
	5	$1.757 \cdot 10^5$	0.2683s	$1.709 \cdot 10^5$	8.7603s	$1.716 \cdot 10^5$	0.3553s
	6	$1.806 \cdot 10^5$	0.2245s	$1.806 \cdot 10^5$	9.4745s	$1.817 \cdot 10^5$	0.4961s
	7	$1.805 \cdot 10^5$	0.2967s	$1.797 \cdot 10^5$	10.8897s	$1.799 \cdot 10^5$	0.4096s
	8	$1.865 \cdot 10^7$	0.4354s	$1.754 \cdot 10^7$	7.4926s	$1.826 \cdot 10^7$	0.2690s
25	2	$1.165 \cdot 10^5$	0.5383s	$1.154 \cdot 10^5$	12.3062s	$1.159 \cdot 10^5$	0.5861s
	3	$1.827 \cdot 10^5$	0.5648s	$1.827 \cdot 10^5$	12.3880s	$1.837 \cdot 10^5$	0.4583s
	4	$2.281 \cdot 10^5$	0.3377s	$2.263 \cdot 10^5$	11.7223s	$2.274 \cdot 10^5$	0.4925s
	5	$1.546 \cdot 10^5$	0.3221s	$1.531 \cdot 10^5$	11.9136s	$1.533 \cdot 10^5$	0.4880s
	6	$1.628 \cdot 10^5$	0.4891s	$1.624 \cdot 10^5$	11.5882s	$1.628 \cdot 10^5$	0.6902s
	7	$1.638 \cdot 10^5$	0.4709s	$1.637 \cdot 10^5$	11.6678s	$1.647 \cdot 10^5$	0.3856s
	8	$1.556 \cdot 10^7$	0.6864s	$1.401 \cdot 10^7$	8.8546s	$1.461 \cdot 10^7$	0.6709s

Table 32: GPS-ephemeris, variant algorithm 2

k	Sub	Algorithm 3		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.861 \cdot 10^5$	4.2019s	$1.860 \cdot 10^5$	7.0203s	$1.877 \cdot 10^5$	0.3629s
	3	$3.015 \cdot 10^5$	3.8228s	$3.015 \cdot 10^5$	5.4234s	$3.015 \cdot 10^5$	0.2703s
	4	$3.874 \cdot 10^5$	5.3255s	$3.874 \cdot 10^5$	6.4364s	$3.874 \cdot 10^5$	0.2906s
	5	$2.577 \cdot 10^5$	3.7689s	$2.577 \cdot 10^5$	5.9961s	$2.584 \cdot 10^5$	0.2556s
	6	$2.702 \cdot 10^5$	5.7849s	$2.702 \cdot 10^5$	6.9112s	$2.739 \cdot 10^5$	0.3859s
	7	$2.570 \cdot 10^5$	4.0685s	$2.570 \cdot 10^5$	5.4438s	$2.570 \cdot 10^5$	0.3403s
	8	$3.689 \cdot 10^7$	3.0817s	$3.689 \cdot 10^7$	5.6573s	$3.805 \cdot 10^7$	0.2493s
15	2	$1.477 \cdot 10^5$	6.5002s	$1.477 \cdot 10^5$	8.2248s	$1.479 \cdot 10^5$	0.5135s
	3	$2.361 \cdot 10^5$	4.7285s	$2.360 \cdot 10^5$	8.5780s	$2.373 \cdot 10^5$	0.2746s
	4	$2.993 \cdot 10^5$	6.3593s	$2.993 \cdot 10^5$	7.9081s	$3.083 \cdot 10^5$	0.5081s
	5	$1.994 \cdot 10^5$	4.9077s	$1.994 \cdot 10^5$	6.3229s	$2.001 \cdot 10^5$	0.2802s
	6	$2.101 \cdot 10^5$	6.6061s	$2.101 \cdot 10^5$	10.1534s	$2.101 \cdot 10^5$	0.6448s
	7	$2.073 \cdot 10^5$	5.0797s	$2.073 \cdot 10^5$	8.8505s	$2.076 \cdot 10^5$	0.4548s
	8	$2.230 \cdot 10^7$	5.0234s	$2.230 \cdot 10^7$	5.1878s	$2.363 \cdot 10^7$	0.2447s
20	2	$1.276 \cdot 10^5$	5.5517s	$1.276 \cdot 10^5$	10.2781s	$1.280 \cdot 10^5$	0.3867s
	3	$2.024 \cdot 10^5$	6.7694s	$2.023 \cdot 10^5$	9.6853s	$2.024 \cdot 10^5$	0.3987s
	4	$2.549 \cdot 10^5$	5.2175s	$2.549 \cdot 10^5$	10.2517s	$2.551 \cdot 10^5$	0.3648s
	5	$1.709 \cdot 10^5$	6.8959s	$1.709 \cdot 10^5$	9.9033s	$1.716 \cdot 10^5$	0.3535s
	6	$1.805 \cdot 10^5$	7.3502s	$1.806 \cdot 10^5$	8.4047s	$1.817 \cdot 10^5$	0.5023s
	7	$1.797 \cdot 10^5$	5.3200s	$1.797 \cdot 10^5$	11.1475s	$1.799 \cdot 10^5$	0.4153s
	8	$1.776 \cdot 10^7$	4.0825s	$1.754 \cdot 10^7$	7.3449s	$1.826 \cdot 10^7$	0.2718s
25	2	$1.154 \cdot 10^5$	7.8389s	$1.154 \cdot 10^5$	12.4010s	$1.159 \cdot 10^5$	0.6102s
	3	$1.827 \cdot 10^5$	6.2638s	$1.827 \cdot 10^5$	12.4040s	$1.837 \cdot 10^5$	0.3997s
	4	$2.272 \cdot 10^5$	7.8127s	$2.263 \cdot 10^5$	11.7005s	$2.274 \cdot 10^5$	0.4759s
	5	$1.534 \cdot 10^5$	7.5501s	$1.531 \cdot 10^5$	11.9777s	$1.533 \cdot 10^5$	0.4971s
	6	$1.623 \cdot 10^5$	6.4182s	$1.624 \cdot 10^5$	11.6182s	$1.628 \cdot 10^5$	0.6805s
	7	$1.639 \cdot 10^5$	8.0789s	$1.637 \cdot 10^5$	11.8848s	$1.647 \cdot 10^5$	0.3377s
	8	$1.421 \cdot 10^7$	4.3976s	$1.401 \cdot 10^7$	9.0185s	$1.461 \cdot 10^7$	0.6582s

Table 33: GPS-ephemeris, algorithm 3

k	Sub	Algorithm 4		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.862 \cdot 10^5$	0.1245s	$1.860 \cdot 10^5$	5.3380s	$1.877 \cdot 10^5$	0.3391s
	3	$3.255 \cdot 10^5$	0.5452s	$3.015 \cdot 10^5$	7.0450s	$3.015 \cdot 10^5$	0.2913s
	4	$3.974 \cdot 10^5$	0.1502s	$3.874 \cdot 10^5$	4.8554s	$3.874 \cdot 10^5$	0.3282s
	5	$2.583 \cdot 10^5$	0.3726s	$2.577 \cdot 10^5$	6.8069s	$2.584 \cdot 10^5$	0.2527s
	6	$2.738 \cdot 10^5$	0.2664s	$2.702 \cdot 10^5$	5.4718s	$2.739 \cdot 10^5$	0.3726s
	7	$2.617 \cdot 10^5$	0.2512s	$2.570 \cdot 10^5$	6.8474s	$2.570 \cdot 10^5$	0.3756s
	8	$3.724 \cdot 10^7$	0.2793s	$3.689 \cdot 10^7$	4.2063s	$3.805 \cdot 10^7$	0.2501s
15	2	$1.479 \cdot 10^5$	0.1829s	$1.477 \cdot 10^5$	9.6103s	$1.479 \cdot 10^5$	0.5083s
	3	$2.476 \cdot 10^5$	0.1584s	$2.360 \cdot 10^5$	8.6052s	$2.373 \cdot 10^5$	0.2641s
	4	$3.094 \cdot 10^5$	0.4179s	$2.993 \cdot 10^5$	6.4180s	$3.083 \cdot 10^5$	0.4624s
	5	$2.126 \cdot 10^5$	0.2431s	$1.994 \cdot 10^5$	7.9412s	$2.001 \cdot 10^5$	0.2446s
	6	$2.101 \cdot 10^5$	0.4452s	$2.101 \cdot 10^5$	10.0331s	$2.101 \cdot 10^5$	0.6137s
	7	$2.077 \cdot 10^5$	0.2635s	$2.073 \cdot 10^5$	7.3243s	$2.076 \cdot 10^5$	0.4202s
	8	$2.509 \cdot 10^7$	0.2463s	$2.230 \cdot 10^7$	6.4002s	$2.363 \cdot 10^7$	0.3040s
20	2	$1.283 \cdot 10^5$	0.2949s	$1.276 \cdot 10^5$	10.4622s	$1.280 \cdot 10^5$	0.3725s
	3	$2.083 \cdot 10^5$	0.3203s	$2.023 \cdot 10^5$	9.8095s	$2.024 \cdot 10^5$	0.3857s
	4	$2.576 \cdot 10^5$	0.3757s	$2.549 \cdot 10^5$	10.1133s	$2.551 \cdot 10^5$	0.3320s
	5	$1.758 \cdot 10^5$	0.5534s	$1.709 \cdot 10^5$	8.0443s	$1.716 \cdot 10^5$	0.3465s
	6	$1.811 \cdot 10^5$	0.5557s	$1.806 \cdot 10^5$	9.7707s	$1.817 \cdot 10^5$	0.5210s
	7	$1.797 \cdot 10^5$	0.1994s	$1.797 \cdot 10^5$	11.0086s	$1.799 \cdot 10^5$	0.4193s
	8	$2.390 \cdot 10^7$	0.2411s	$1.754 \cdot 10^7$	7.3082s	$1.826 \cdot 10^7$	0.2787s
25	2	$1.165 \cdot 10^5$	0.5333s	$1.154 \cdot 10^5$	12.3794s	$1.159 \cdot 10^5$	0.6040s
	3	$1.862 \cdot 10^5$	0.3007s	$1.827 \cdot 10^5$	12.3420s	$1.837 \cdot 10^5$	0.4683s
	4	$2.407 \cdot 10^5$	0.5369s	$2.263 \cdot 10^5$	11.6328s	$2.274 \cdot 10^5$	0.4566s
	5	$1.600 \cdot 10^5$	0.6523s	$1.531 \cdot 10^5$	12.1166s	$1.533 \cdot 10^5$	0.4309s
	6	$1.630 \cdot 10^5$	3.0725s	$1.624 \cdot 10^5$	11.6296s	$1.628 \cdot 10^5$	0.6756s
	7	$1.641 \cdot 10^5$	0.5354s	$1.637 \cdot 10^5$	11.4101s	$1.647 \cdot 10^5$	0.9488s
	8	$1.704 \cdot 10^7$	0.5698s	$1.401 \cdot 10^7$	7.7406s	$1.461 \cdot 10^7$	1.6837s

Table 34: GPS-ephemeris, algorithm 4

k	Sub	Algorithm 4v		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.862 \cdot 10^5$	0.1369s	$1.860 \cdot 10^5$	5.2679s	$1.877 \cdot 10^5$	0.3160s
	3	$3.036 \cdot 10^5$	0.1418s	$3.015 \cdot 10^5$	6.9665s	$3.015 \cdot 10^5$	0.2564s
	4	$3.874 \cdot 10^5$	0.1915s	$3.874 \cdot 10^5$	4.8130s	$3.874 \cdot 10^5$	1.1032s
	5	$2.584 \cdot 10^5$	0.1290s	$2.577 \cdot 10^5$	7.5784s	$2.584 \cdot 10^5$	0.7798s
	6	$2.740 \cdot 10^5$	0.1292s	$2.702 \cdot 10^5$	6.5507s	$2.739 \cdot 10^5$	0.5124s
	7	$2.579 \cdot 10^5$	0.2771s	$2.570 \cdot 10^5$	5.7212s	$2.570 \cdot 10^5$	0.3294s
	8	$3.724 \cdot 10^7$	0.2650s	$3.689 \cdot 10^7$	4.2089s	$3.805 \cdot 10^7$	0.2579s
15	2	$1.479 \cdot 10^5$	0.1937s	$1.477 \cdot 10^5$	9.7760s	$1.479 \cdot 10^5$	0.4797s
	3	$2.372 \cdot 10^5$	0.4669s	$2.360 \cdot 10^5$	8.3853s	$2.373 \cdot 10^5$	0.2817s
	4	$3.015 \cdot 10^5$	0.3559s	$2.993 \cdot 10^5$	6.5176s	$3.083 \cdot 10^5$	0.4913s
	5	$2.018 \cdot 10^5$	0.2048s	$1.994 \cdot 10^5$	7.9986s	$2.001 \cdot 10^5$	0.2708s
	6	$2.101 \cdot 10^5$	0.2718s	$2.101 \cdot 10^5$	10.2183s	$2.101 \cdot 10^5$	0.5864s
	7	$2.091 \cdot 10^5$	0.2301s	$2.073 \cdot 10^5$	7.0401s	$2.076 \cdot 10^5$	0.4305s
	8	$2.421 \cdot 10^7$	0.2996s	$2.230 \cdot 10^7$	6.4632s	$2.363 \cdot 10^7$	0.2971s
20	2	$1.283 \cdot 10^5$	0.3103s	$1.276 \cdot 10^5$	10.2271s	$1.280 \cdot 10^5$	0.3879s
	3	$2.024 \cdot 10^5$	0.3291s	$2.023 \cdot 10^5$	9.7127s	$2.024 \cdot 10^5$	0.3791s
	4	$2.560 \cdot 10^5$	0.4310s	$2.549 \cdot 10^5$	8.4006s	$2.551 \cdot 10^5$	1.2943s
	5	$1.758 \cdot 10^5$	0.2429s	$1.709 \cdot 10^5$	9.9955s	$1.716 \cdot 10^5$	1.0829s
	6	$1.810 \cdot 10^5$	0.1863s	$1.806 \cdot 10^5$	9.9203s	$1.817 \cdot 10^5$	0.5011s
	7	$1.802 \cdot 10^5$	0.2561s	$1.797 \cdot 10^5$	10.9187s	$1.799 \cdot 10^5$	0.3943s
	8	$1.811 \cdot 10^7$	0.5958s	$1.754 \cdot 10^7$	5.8345s	$1.826 \cdot 10^7$	0.2735s
25	2	$1.165 \cdot 10^5$	0.5564s	$1.154 \cdot 10^5$	12.2926s	$1.159 \cdot 10^5$	0.6338s
	3	$1.842 \cdot 10^5$	0.3478s	$1.827 \cdot 10^5$	12.2190s	$1.837 \cdot 10^5$	1.6448s
	4	$2.275 \cdot 10^5$	0.4925s	$2.263 \cdot 10^5$	11.4409s	$2.274 \cdot 10^5$	0.8214s
	5	$1.582 \cdot 10^5$	0.3518s	$1.531 \cdot 10^5$	12.1233s	$1.533 \cdot 10^5$	0.4277s
	6	$1.631 \cdot 10^5$	0.6463s	$1.624 \cdot 10^5$	11.7307s	$1.628 \cdot 10^5$	0.6603s
	7	$1.643 \cdot 10^5$	0.3121s	$1.637 \cdot 10^5$	11.7891s	$1.647 \cdot 10^5$	0.3637s
	8	$1.559 \cdot 10^7$	0.7356s	$1.401 \cdot 10^7$	7.3822s	$1.461 \cdot 10^7$	0.6657s

Table 35: GPS-ephemeris, variant algorithm 4

k	Sub	Algorithm 5		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.862 \cdot 10^5$	0.1191s	$1.860 \cdot 10^5$	26.5743s	$1.877 \cdot 10^5$	0.3309s
	3	$3.036 \cdot 10^5$	10.4178s	$3.015 \cdot 10^5$	6.8687s	$3.015 \cdot 10^5$	0.2552s
	4	$3.907 \cdot 10^5$	14.8484s	$3.874 \cdot 10^5$	4.7734s	$3.874 \cdot 10^5$	0.3254s
	5	$2.577 \cdot 10^5$	19.5819s	$2.577 \cdot 10^5$	6.8425s	$2.584 \cdot 10^5$	0.2408s
	6	$2.743 \cdot 10^5$	22.6122s	$2.702 \cdot 10^5$	5.5413s	$2.739 \cdot 10^5$	0.3728s
	7	$2.570 \cdot 10^5$	29.6159s	$2.570 \cdot 10^5$	6.9120s	$2.570 \cdot 10^5$	0.9449s
	8	$3.724 \cdot 10^7$	30.9081s	$3.689 \cdot 10^7$	4.1255s	$3.805 \cdot 10^7$	0.9200s
15	2	$1.479 \cdot 10^5$	0.1852s	$1.477 \cdot 10^5$	43.2570s	$1.479 \cdot 10^5$	0.5146s
	3	$2.382 \cdot 10^5$	13.9461s	$2.360 \cdot 10^5$	6.9062s	$2.373 \cdot 10^5$	0.2840s
	4	$3.021 \cdot 10^5$	22.0020s	$2.993 \cdot 10^5$	8.0591s	$3.083 \cdot 10^5$	0.5294s
	5	$2.036 \cdot 10^5$	30.7774s	$1.994 \cdot 10^5$	7.8602s	$2.001 \cdot 10^5$	0.2852s
	6	$2.101 \cdot 10^5$	39.2655s	$2.101 \cdot 10^5$	8.5631s	$2.101 \cdot 10^5$	0.6504s
	7	$2.077 \cdot 10^5$	45.8106s	$2.073 \cdot 10^5$	9.1802s	$2.076 \cdot 10^5$	0.4815s
	8	$2.338 \cdot 10^7$	49.7727s	$2.230 \cdot 10^7$	6.7887s	$2.363 \cdot 10^7$	0.3030s
20	2	$1.283 \cdot 10^5$	0.3194s	$1.276 \cdot 10^5$	49.7657s	$1.280 \cdot 10^5$	0.4069s
	3	$2.024 \cdot 10^5$	17.1891s	$2.023 \cdot 10^5$	9.4674s	$2.024 \cdot 10^5$	0.3686s
	4	$2.550 \cdot 10^5$	24.1476s	$2.549 \cdot 10^5$	10.1776s	$2.551 \cdot 10^5$	0.3642s
	5	$1.716 \cdot 10^5$	35.2049s	$1.709 \cdot 10^5$	9.7998s	$1.716 \cdot 10^5$	0.3630s
	6	$1.810 \cdot 10^5$	43.4947s	$1.806 \cdot 10^5$	8.2787s	$1.817 \cdot 10^5$	0.5194s
	7	$1.804 \cdot 10^5$	51.9037s	$1.797 \cdot 10^5$	11.0037s	$1.799 \cdot 10^5$	0.3617s
	8	$1.810 \cdot 10^7$	59.5579s	$1.754 \cdot 10^7$	7.5501s	$1.826 \cdot 10^7$	0.2963s
25	2	$1.165 \cdot 10^5$	0.5581s	$1.154 \cdot 10^5$	61.3122s	$1.159 \cdot 10^5$	0.5757s
	3	$1.851 \cdot 10^5$	22.5314s	$1.827 \cdot 10^5$	12.5137s	$1.837 \cdot 10^5$	0.4537s
	4	$2.278 \cdot 10^5$	31.6216s	$2.263 \cdot 10^5$	11.9425s	$2.274 \cdot 10^5$	0.4950s
	5	$1.550 \cdot 10^5$	40.3630s	$1.531 \cdot 10^5$	12.3011s	$1.533 \cdot 10^5$	0.4472s
	6	$1.638 \cdot 10^5$	58.8680s	$1.624 \cdot 10^5$	10.8695s	$1.628 \cdot 10^5$	0.6873s
	7	$1.641 \cdot 10^5$	69.6068s	$1.637 \cdot 10^5$	10.9394s	$1.647 \cdot 10^5$	0.3752s
	8	$1.556 \cdot 10^7$	84.1518s	$1.401 \cdot 10^7$	9.1427s	$1.461 \cdot 10^7$	2.3060s

Table 36: GPS-ephemeris, algorithm 5

k	Sub	Algorithm 6		`Km++L` 20 init		`Km++L` 1 init	
		cost	time	cost	time	cost	time
10	2	$1.861 \cdot 10^5$	0.2270s	$1.860 \cdot 10^5$	7.0020s	$1.877 \cdot 10^5$	0.3327s
	3	$3.058 \cdot 10^5$	0.2481s	$3.015 \cdot 10^5$	5.3275s	$3.015 \cdot 10^5$	0.2322s
	4	$3.874 \cdot 10^5$	0.2729s	$3.874 \cdot 10^5$	6.5208s	$3.874 \cdot 10^5$	0.3352s
	5	$2.584 \cdot 10^5$	0.5121s	$2.577 \cdot 10^5$	5.2701s	$2.584 \cdot 10^5$	0.2525s
	6	$2.704 \cdot 10^5$	0.2857s	$2.702 \cdot 10^5$	7.0835s	$2.739 \cdot 10^5$	0.3512s
	7	$2.585 \cdot 10^5$	0.3811s	$2.570 \cdot 10^5$	5.3918s	$2.570 \cdot 10^5$	0.3955s
	8	$3.805 \cdot 10^7$	0.2328s	$3.689 \cdot 10^7$	5.5351s	$3.805 \cdot 10^7$	0.2143s
15	2	$1.493 \cdot 10^5$	0.3426s	$1.477 \cdot 10^5$	9.7248s	$1.479 \cdot 10^5$	1.9685s
	3	$2.385 \cdot 10^5$	0.3071s	$2.360 \cdot 10^5$	7.0158s	$2.373 \cdot 10^5$	0.2222s
	4	$3.102 \cdot 10^5$	0.4061s	$2.993 \cdot 10^5$	8.1744s	$3.083 \cdot 10^5$	0.5321s
	5	$2.031 \cdot 10^5$	0.7557s	$1.994 \cdot 10^5$	6.5542s	$2.001 \cdot 10^5$	0.2745s
	6	$2.114 \cdot 10^5$	0.4406s	$2.101 \cdot 10^5$	9.8317s	$2.101 \cdot 10^5$	0.6013s
	7	$2.077 \cdot 10^5$	0.3052s	$2.073 \cdot 10^5$	8.9584s	$2.076 \cdot 10^5$	0.4441s
	8	$2.229 \cdot 10^7$	0.3351s	$2.230 \cdot 10^7$	5.3215s	$2.363 \cdot 10^7$	0.2648s
20	2	$1.288 \cdot 10^5$	0.3140s	$1.276 \cdot 10^5$	10.2315s	$1.280 \cdot 10^5$	0.3792s
	3	$2.032 \cdot 10^5$	0.5122s	$2.023 \cdot 10^5$	9.6255s	$2.024 \cdot 10^5$	0.3684s
	4	$2.570 \cdot 10^5$	0.5708s	$2.549 \cdot 10^5$	10.0085s	$2.551 \cdot 10^5$	0.3405s
	5	$1.772 \cdot 10^5$	0.4247s	$1.709 \cdot 10^5$	8.0760s	$1.716 \cdot 10^5$	0.3520s
	6	$1.812 \cdot 10^5$	0.5520s	$1.806 \cdot 10^5$	9.7926s	$1.817 \cdot 10^5$	0.5107s
	7	$1.804 \cdot 10^5$	0.7404s	$1.797 \cdot 10^5$	10.9720s	$1.799 \cdot 10^5$	0.3917s
	8	$1.826 \cdot 10^7$	0.2834s	$1.754 \cdot 10^7$	7.1731s	$1.826 \cdot 10^7$	0.2835s
25	2	$1.159 \cdot 10^5$	0.6427s	$1.154 \cdot 10^5$	12.3160s	$1.159 \cdot 10^5$	1.6948s
	3	$1.844 \cdot 10^5$	0.4019s	$1.827 \cdot 10^5$	12.1663s	$1.837 \cdot 10^5$	0.4613s
	4	$2.288 \cdot 10^5$	0.5869s	$2.263 \cdot 10^5$	11.5381s	$2.274 \cdot 10^5$	0.4848s
	5	$1.579 \cdot 10^5$	0.4057s	$1.531 \cdot 10^5$	12.0611s	$1.533 \cdot 10^5$	0.4790s
	6	$1.627 \cdot 10^5$	1.4379s	$1.624 \cdot 10^5$	11.5218s	$1.628 \cdot 10^5$	0.6760s
	7	$1.642 \cdot 10^5$	0.4715s	$1.637 \cdot 10^5$	11.9799s	$1.647 \cdot 10^5$	0.3839s
	8	$1.461 \cdot 10^7$	0.6227s	$1.401 \cdot 10^7$	8.0151s	$1.461 \cdot 10^7$	0.6531s

Table 37: GPS-ephemeris, algorithm 6

5.4.1 Analysis

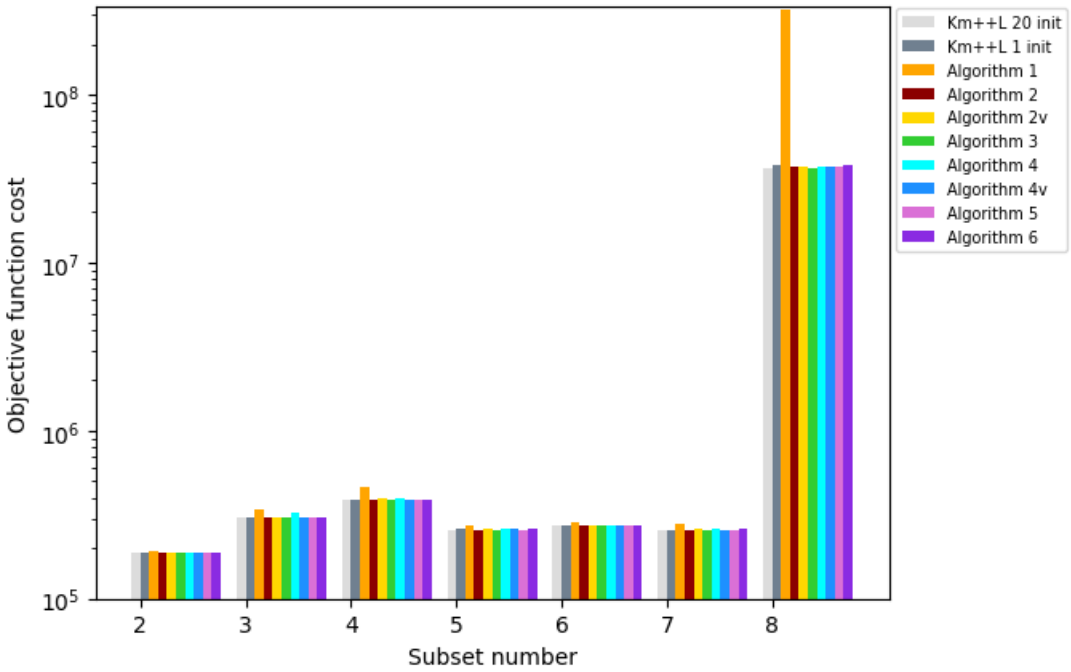

Figure 19: GPS-ephemeris, k=10

Figure 19 shows the objective function cost obtained by each algorithm on each subset of GPS-ephemeris for $k = 10$. Due to the apparently particular distribution of the last subset we had to plot the costs using a logarithmic scale, and as a side effect the bars representing the costs are quite homogeneous, but they are still a valid representation of them. That said, we can immediately see that algorithm 1 did not perform well in any subset. The main reason may be that the naive approach of algorithm 1 does not fit the distribution of the GPS-ephemeris dataset, which apparently requires a more well thought procedure, as the other algorithms propose.

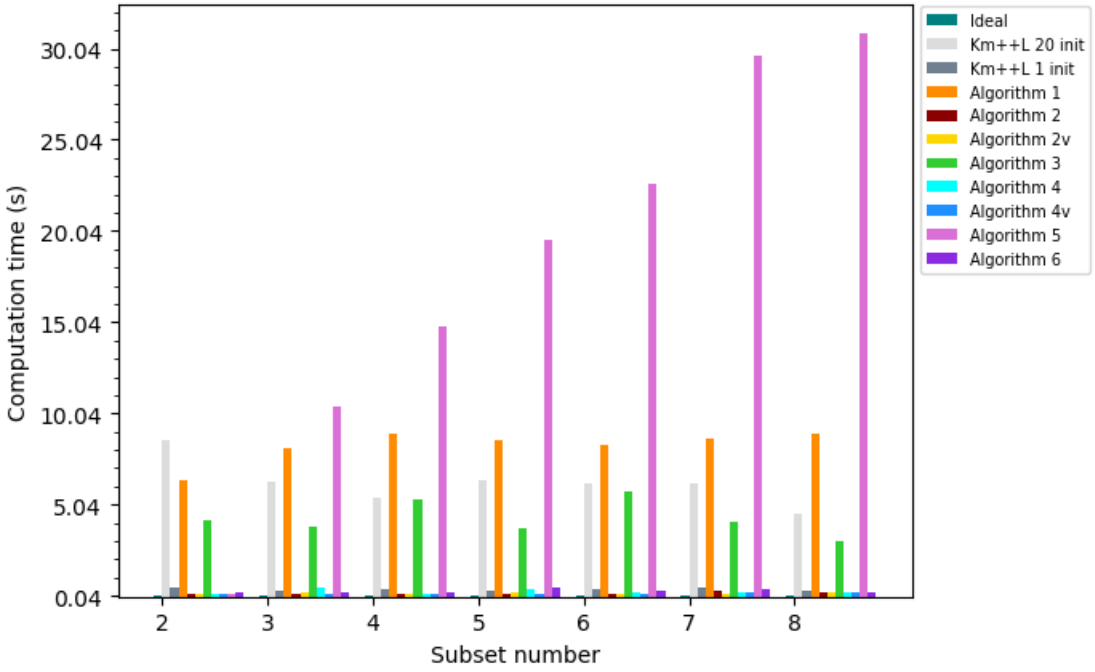

Figure 20: GPS-ephemeris, k=10, computation time

Figure 20 shows the computing time of each algorithm on each subset of the GPS-ephemeris dataset for $k = 10$. Results are very similar to the ones found for the Oregon dataset, depicted in figure 8, so the same considerations apply.

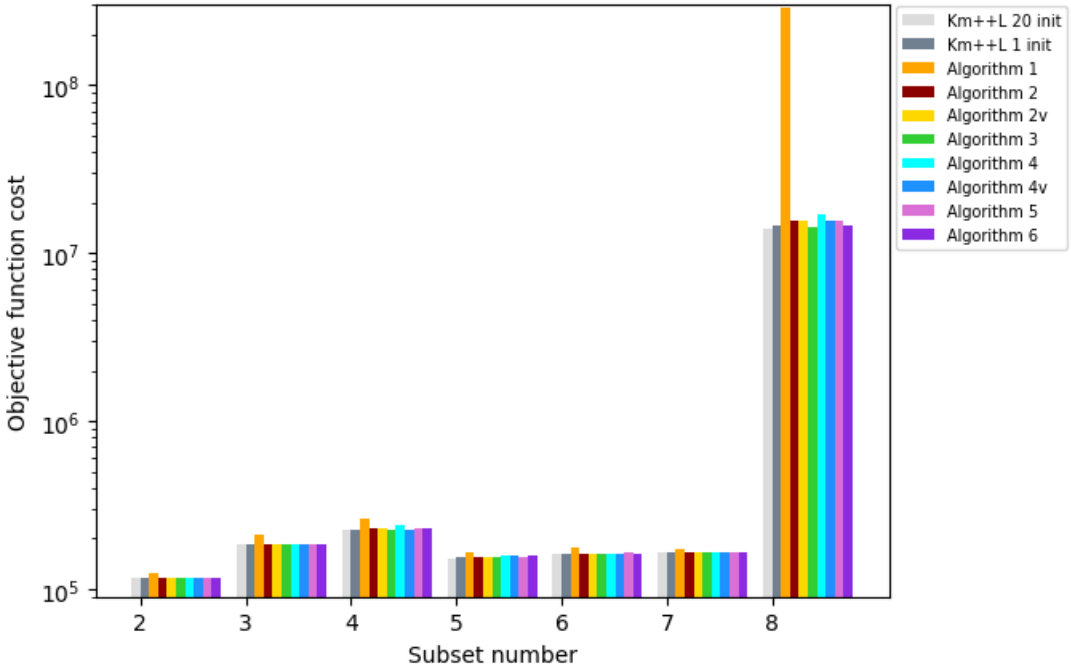

Figure 21: GPS-ephemeris, k=25, objective function cost

Figure 21 shows the objective function cost obtained by each algorithm on each subset of GPS-ephemeris for $k = 25$. Results are very similar to the ones found for the $k = 10$ case, depicted in figure 19, so the same considerations apply.

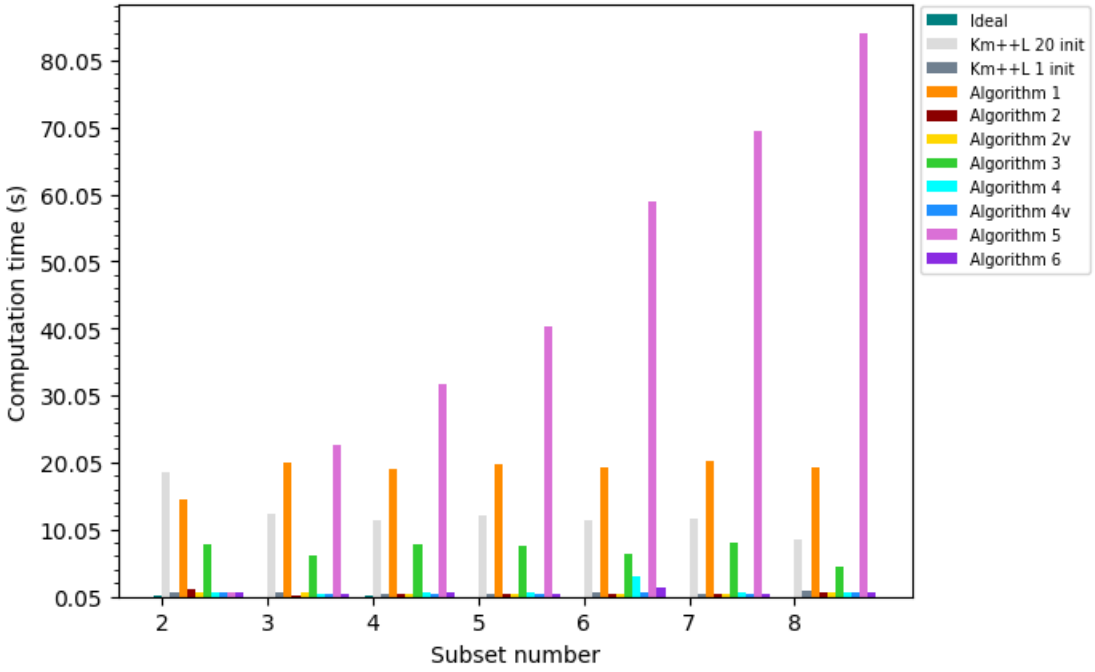

Figure 22: GPS-ephemeris, k=25, computation time

Figure 22 shows the computing time of each algorithm on each subset of the GPS-ephemeris dataset for $k = 25$. Results are very similar to the ones found for the Oregon dataset, depicted in figure 8, so the same considerations apply.

Chapter 6

Additional Analyses

In this section we are reporting additional analyses on some of the algorithms described above. These analyses serve the purpose to explore more in depth the algorithms, for example by testing them with different parameters.

6.1 Additional analysis: algorithm 2

In algorithm 2 we use the centers found on the first subset as a starting set of centers for `Lloyd` applied on the subsequent subsets. As it can be seen from the results above, sometimes this algorithm works well and in some other cases it does not. In order to have an initial understanding of whether it is worth applying algorithm 2 to a subset or not to it could be reasonable to analyze the normalized (clustering) costs. The normalized cost nc of a set P, given a set S of k centers, is defined as $nc = \frac{cost(P,S)}{|P|}$, where $cost(P, S)$ is the clustering cost of P when using S as set of centers and $|P|$ is the cardinality of P.

We analyzed the normalized costs in all the four real datasets for $k = 10, 25$ and reported the results in the table below. We indicated with nc_0 the normalized cost of the first subset, and with nc_i the normalized cost of the i-th subset.

Dataset	Sub	$k = 10$		$k = 25$	
		nc_0	nc_i	nc_0	nc_i
Oregon	2	$2.59 \cdot 10^{-5}$	$1.47 \cdot 10^{-4}$	$6.53 \cdot 10^{-6}$	$5.84 \cdot 10^{-5}$
	3		$8.39 \cdot 10^{-5}$		$2.70 \cdot 10^{-5}$
	4		$1.61 \cdot 10^{-4}$		$5.31 \cdot 10^{-5}$
	5		$2.72 \cdot 10^{-5}$		$1.05 \cdot 10^{-5}$
	6		$5.38 \cdot 10^{-4}$		$9.59 \cdot 10^{-5}$
	7		$7.54 \cdot 10^{-5}$		$3.34 \cdot 10^{-5}$
	8		$7.58 \cdot 10^{-5}$		$3.26 \cdot 10^{-5}$
	9		$1.72 \cdot 10^{-4}$		$7.34 \cdot 10^{-5}$
Bank	2	$1.18 \cdot 10^{8}$	$2.09 \cdot 10^{8}$	$5.27 \cdot 10^{7}$	$1.34 \cdot 10^{8}$
	3		$2.52 \cdot 10^{8}$		$1.61 \cdot 10^{8}$
	4		$2.15 \cdot 10^{8}$		$1.41 \cdot 10^{8}$
	5		$1.49 \cdot 10^{8}$		$8.55 \cdot 10^{7}$
	6		$1.99 \cdot 10^{8}$		$1.24 \cdot 10^{8}$
	7		$1.40 \cdot 10^{8}$		$8.20 \cdot 10^{7}$
Power	2	$1.22 \cdot 10^{7}$	$1.26 \cdot 10^{7}$	$5.61 \cdot 10^{6}$	$5.83 \cdot 10^{6}$
	3		$1.25 \cdot 10^{7}$		$5.95 \cdot 10^{6}$
	4		$1.22 \cdot 10^{7}$		$5.75 \cdot 10^{6}$
	5		$1.24 \cdot 10^{7}$		$5.95 \cdot 10^{6}$
	6		$1.24 \cdot 10^{7}$		$5.73 \cdot 10^{6}$
	7		$1.22 \cdot 10^{7}$		$5.74 \cdot 10^{6}$
GPS	2	6.34	5.43	3.91	3.66
	3		$1.11 \cdot 10^{1}$		7.23
	4		$1.45 \cdot 10^{1}$		9.86
	5		8.19		5.34
	6		8.00		5.26
	7		8.14		5.35
	8		$1.97 \cdot 10^{4}$		$1.94 \cdot 10^{4}$

Table 38: additional analysis algorithm 2, real datasets

6.2 Additional analysis: algorithm 3

In algorithm 3 we divide a given subset in half, perform `Km++L` on one of the two halves and use the set of centers found in this way as an initial set of centers for `Lloyd` applied on the entire subset. This algorithm is quite fast and often yields good results compared to the baselines, but we were interested to know how the results change when changing the size of the analyzed part of the subset. For this reason we run again algorithm 3 on the real datasets, in the case $k = 10$, but this time we took one fifth, one tenth and one twentieth of the subsets, instead of their half, and gathered the results we obtained, both in terms of time and of costs of the objective function.

The table containing the results we found is reported below.

Dataset	Sub	1/5		1/10		1/20	
		cost	time	cost	time	cost	time
Oregon	2	$6.842 \cdot 10^{-1}$	0.0954s	$7.390 \cdot 10^{-1}$	0.0724s	$6.823 \cdot 10^{-1}$	0.1013s
	3	$4.175 \cdot 10^{-1}$	0.0780s	$4.175 \cdot 10^{-1}$	0.0670s	$4.267 \cdot 10^{-1}$	0.1068s
	4	$6.365 \cdot 10^{-1}$	0.0892s	$6.513 \cdot 10^{-1}$	0.1096s	$6.395 \cdot 10^{-1}$	0.0994s
	5	$2.177 \cdot 10^{-1}$	0.1048s	$1.996 \cdot 10^{-1}$	0.0771s	$2.039 \cdot 10^{-1}$	0.1095s
	6	$8.601 \cdot 10^{-1}$	0.0787s	$8.520 \cdot 10^{-1}$	0.0729s	$8.615 \cdot 10^{-1}$	0.0946s
	7	$2.877 \cdot 10^{-1}$	0.0784s	$2.877 \cdot 10^{-1}$	0.0719s	$2.763 \cdot 10^{-1}$	0.1297s
	8	$4.154 \cdot 10^{-1}$	0.0773s	$4.053 \cdot 10^{-1}$	0.0729s	$4.233 \cdot 10^{-1}$	0.1009s
	9	$5.987 \cdot 10^{-1}$	0.0830s	$6.052 \cdot 10^{-1}$	0.1145s	$6.043 \cdot 10^{-1}$	0.0936s
Bank	2	$1.771 \cdot 10^{12}$	0.5026s	$1.757 \cdot 10^{12}$	0.2306s	$1.947 \cdot 10^{12}$	0.8180s
	3	$2.360 \cdot 10^{12}$	1.0074s	$2.220 \cdot 10^{12}$	0.5317s	$2.041 \cdot 10^{12}$	1.0393s
	4	$1.848 \cdot 10^{12}$	0.5958s	$2.866 \cdot 10^{12}$	0.6354s	$2.012 \cdot 10^{12}$	2.4237s
	5	$1.414 \cdot 10^{12}$	0.6877s	$1.425 \cdot 10^{12}$	0.4038s	$1.301 \cdot 10^{12}$	1.0245s
	6	$1.760 \cdot 10^{12}$	0.8195s	$1.927 \cdot 10^{12}$	0.4358s	$1.966 \cdot 10^{12}$	0.8573s
	7	$1.586 \cdot 10^{12}$	0.9755s	$1.587 \cdot 10^{12}$	0.6329s	$1.338 \cdot 10^{12}$	1.1813s
Power	2	$9.389 \cdot 10^{10}$	0.1212s	$9.390 \cdot 10^{10}$	0.1049s	$9.389 \cdot 10^{10}$	0.9010s
	3	$9.366 \cdot 10^{10}$	0.1525s	$9.305 \cdot 10^{10}$	0.0929s	$9.306 \cdot 10^{10}$	0.9139s
	4	$9.074 \cdot 10^{10}$	0.1111s	$9.074 \cdot 10^{10}$	0.1097s	$9.104 \cdot 10^{10}$	1.0596s
	5	$9.268 \cdot 10^{10}$	0.1222s	$9.268 \cdot 10^{10}$	0.0936s	$9.268 \cdot 10^{10}$	0.9317s
	6	$9.240 \cdot 10^{10}$	0.1248s	$9.241 \cdot 10^{10}$	0.1281s	$9.240 \cdot 10^{10}$	0.9709s
	7	$9.095 \cdot 10^{10}$	0.1312s	$9.095 \cdot 10^{10}$	0.1145s	$9.144 \cdot 10^{10}$	0.9583s
GPS	2	$1.863 \cdot 10^{5}$	2.4877s	$1.863 \cdot 10^{5}$	1.3227s	$1.863 \cdot 10^{5}$	3.0269s
	3	$3.015 \cdot 10^{5}$	2.3281s	$3.030 \cdot 10^{5}$	1.1493s	$3.030 \cdot 10^{5}$	4.6549s
	4	$3.874 \cdot 10^{5}$	2.1195s	$3.888 \cdot 10^{5}$	1.3109s	$3.874 \cdot 10^{5}$	2.7957s
	5	$2.577 \cdot 10^{5}$	3.6540s	$2.584 \cdot 10^{5}$	3.1218s	$2.577 \cdot 10^{5}$	2.9868s
	6	$2.702 \cdot 10^{5}$	2.2354s	$2.702 \cdot 10^{5}$	1.2949s	$2.702 \cdot 10^{5}$	3.1444s
	7	$2.570 \cdot 10^{5}$	2.0410s	$2.579 \cdot 10^{5}$	1.4570s	$2.570 \cdot 10^{5}$	4.6924s
	8	$3.772 \cdot 10^{7}$	1.4898s	$3.805 \cdot 10^{7}$	1.1576s	$3.689 \cdot 10^{7}$	2.0166s

Table 39: Additional analysis algorithm 3

Chapter 7

Discussion and conclusion

The work presented in this thesis represents a novel approach to the k-means problem in a setting where datasets are assumed to belong to a common distribution $\mathcal{D}$. In order to exploit this fundamental characteristic we developed original algorithms that also made use of machine learning techniques, which is in line with the framework of algorithms with predictions. We tested our algorithms on two different types of datasets, that is synthetic datasets and real datasets, and from each of them we were able to draw some interesting conclusions which we are now going to unravel separately, for the sake of clarity.

7.1 Synthetic datasets

The synthetic datasets we created respect the assumption of "datasets coming from the same distribution", since, as we stated earlier, we have created them exactly with this characteristic. This allowed us to clearly see how our algorithms perform with such instances, and consequently to have an initial understanding of what to expect in real datasets.

Focusing on the Synth20K dataset, in figure 1 it was clear that most of our algorithms outperformed both the baselines in terms of cost of the objective function. Additionally, almost all of these algorithms were faster compared to the baselines, and some of them were even close to the `Ideal` one. This last consideration is true also for Synth100K and Synth2M, whereas the objective function costs were at most equal to those of the baselines. However, these results hold a great importance for two reasons. First of all, we empirically showed that it is possible to obtain better results than the `Km++L` algorithm under the condition that the

datasets come from a common distribution. Second, working under this assumption makes it possible to develop very fast algorithms without having to trade off accuracy also for big datasets, at least for the synthetic ones.

Nevertheless, in order to draw more accurate conclusions it would be necessary to perform more tests with different values of the hyperparameter k and in different instances. Alternatively, it could be useful to study the matter from a theoretical point of view, hence obtaining a better understanding of the experimental results presented in this thesis.

7.2 Real datasets

In order to better understand how our algorithms perform in real case scenarios we tested them on real datasets with different cardinalities and number of features. The results we found are not completely similar to those obtained on the synthetic datasets, in particular regarding the cost of the objective function. In fact, the performance of our algorithms is not always consistent from one dataset to another, and sometimes this also happens within the same dataset. On the other hand, in some instances our algorithms work even better than the baselines, thus becoming competitive to the baselines themselves.

This inconsistent behaviour does not allow us to outline an error proof algorithm or approach to the clustering task in the setting defined earlier, but instead it seems to suggest that the choice of the algorithm depends on the dataset at issue, maybe due to certain characteristics of the given dataset. Of course, it would be optimal to know what these characteristics are, but in order to do so we believe that more tests and a theoretical approach would be needed.

As a last consideration, in terms of computation time the majority of our algorithms outperform the baselines in the real datasets, making this a similarity with what we have seen for the synthetic ones. This means that there exists a tradeoff between objective function cost and running time, and depending on the situation

one can choose whether to use the standard `Km++L` algorithm or instead use one or more of ours. For example, when dealing with massive datasets, coming from the same distribution, one could prefer to obtain a quick but slightly imprecise clustering of the data, in which case our algorithms have proven to be effective.

7.3 Final considerations and future works

The work we have done in this thesis can be considered as the starting point for the study of clustering in a setting where datasets belong to a common distribution. Our contributions are mainly empirical, hence in future works it could be effective to perform a theoretical analysis that can guide the development of new algorithms in a better direction. Although obvious, it could be useful trying to understand why some algorithms perform well in a given dataset and not so well in others, maybe focusing on the differences between the datasets both in terms of the elements they contain and of their distributions.

Acknowledgments

A conclusione di questo lavoro ci tengo a ringraziare dovutamente tutti coloro che hanno reso possibile e mi hanno accompagnato nell'esperienza biennale, ma tutt'altro che prolissa, della laurea magistrale.

Primo fra tutti, e non per convenzione, voglio ringraziare il professor Fabio Vandin che, oltre ad aver tenuto due fra i corsi più interessanti che abbia frequentato, è stato anche il supervisore di questa tesi. La parte più difficile del ringraziarlo è condensare le decine di pagine che potrei scrivere in un solo paragrafo. Il motivo principale è che sia dal punto di vista umano che accademico il professor Vandin è immancabilmente una figura eccezionale. Tra tutto, lo ringrazio per gli svariati meeting e i feedback alle varie versioni della tesi, e di conseguenza per la cura con cui mi ha seguito in questo progetto. La volontà di essere all'altezza delle sue aspettative mi ha spinto a impegnarmi a fondo, e anche di questo gli sono profondamente riconoscente.

Tuttavia, ci sono anche altre persone che in questi due anni mi hanno, involontariamente, esortato a impegnarmi e a superare i miei limiti. E non solo, in quanto hanno indubbiamente aggiunto un grande valore ad ogni esperienza che abbiamo vissuto insieme. Per questo, ringrazio profondamente Giorgio Venturin, Filippo Boldrin, Davide Sarraggiotto, Cristian Boldrin e Alberto Makosa.

Infine, ma non meno importante, ringrazio la mia famiglia e i miei amici per il supporto che mi hanno dato in questi anni. Siete stati una presenza importante e un sostegno sempre presente, e spero in futuro di poter ricambiare quanto avete fatto per me.

Grazie di tutto.

Leonardo

References

[1] T. M. Mitchell, "Artificial neural networks." Boston, MA: McGraw-Hill, 1997.

[2] Z. Ghahramani, *Unsupervised Learning*, pp. 72–112. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[3] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, (USA), p. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[4] Y. Lu and H. H. Zhou, "Statistical and computational guarantees of lloyd's algorithm and its variants," *arXiv preprint arXiv:1612.02099*, 2016.

[5] M. Mitzenmacher and S. Vassilvitskii, "Algorithms with predictions," *Communications of the ACM*, vol. 65, no. 7, 2022.

[6] J. C. Ergun, Z. Feng, S. Silwal, D. P. Woodruff, and S. Zhou, "Learning-augmented k-means clustering," *arXiv preprint arXiv:2110.14094*, 2021.

[7] J. K. J. Leskovec and C. Faloutsos, "Graphs over time: Densification laws, shrinking diameters and possible explanations," 2005. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).

[8] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection." `http://snap.stanford.edu/data`, June 2014.