An exploration of novel heuristics for efficient and accurate k-means clustering [DRAFT]

Samuele Nicolò Straccialini

Update: June 8, 2025

Contents

1 Introduction **5**

1.1 Materials and implementation 7

2 Initializations **8**

2.1 random 8

2.2 maximin 10

2.3 k-means++ 10

2.4 greedy k-means++ 11

3 Algorithms **13**

3.1 Lloyd's Algorithm 13

3.2 Hartigan's Algorithm 16

3.3 Extended-Hartigan's Algorithm 21

3.4 Mixed Modes 24

3.5 Other Heuristics 24

4 Simulations **26**

4.1 Datasets 26

4.2 Metrics 27

4.2.1 Centroid Index (CI) 27

4.2.2 Norm calculations 28

4.3 Results 29

4.3.1 Stability 29

4.3.2 Mean final cost improvement 30

4.3.3 Minimum final cost improvement 31

4.3.4 Mean norm calculations improvement 32
4.3.5 Robustness of Extended Hartigan 34

5 Conclusion **36**

A Tables and Figures **39**
A.1 Tables of Results 39
A.2 Mean Final Cost 44
A.3 Minimum Final Cost 45
A.4 Mean Norm Calculations 46
A.5 Safe and Unsafe iterations in `extended-hartigan` 46

Bibliography **54**

Chapter 1

Introduction

Clustering is one of the main tasks when performing exploratory data analysis or statistical data analysis since it allows to identify relevant patterns in data. Intuitively, it consists of partitioning a set of objects so that similar ones are in the same cluster (i.e., group) and dissimilar ones are in different clusters. Applications range from pattern recognition, data compression, and bio-informatics to image analysis, vector quantization, and anomaly detection [SB14].

K-means is by far the most widely used method for discovering clusters in data[1] [Elk03]. This technique has been proposed by several researchers in different forms and under different assumptions. Due to its importance, k-means is continuously investigated both from the theoretical and the algorithmic sides, by modifying methods, investigating its behavior, and extending its domain of application.

K-means is an unsupervised learning algorithm that aims to partition a set of n observations into k clusters by minimizing the distances between each data point and its cluster's centroid. More formally, given a set of d-dimensional observations $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ for $i \in \{1, 2, \ldots, n\}$, we want to partition them in $k \leq n$ clusters $S_1, S_2, \ldots, S_k$ in order to minimize the cost

$$C = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 , \tag{1.1}$$

[1] Wu et al. (2008) recognize k-means as one of the top 10 algorithms in data mining [Wu+08].

where $\boldsymbol{\mu}_i$ is the centroid of the cluster, i.e. the mean of the points assigned to it:

$$\boldsymbol{\mu}_i = \frac{1}{|S_i|} \sum_{\boldsymbol{x} \in S_i} \boldsymbol{x}. \tag{1.2}$$

Namely, we aim to set

$$\{S_1, \ldots, S_n\} = \arg\min_{S_1,\ldots,S_n} C = \arg\min_{S_1,\ldots,S_n} \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i} \|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 . \tag{1.3}$$

In equations 1.1, 1.2, 1.3, and throughout this text, we denote by $\|\cdot\|$ the standard L2 norm and by $|A|$ the cardinality of the set A.

There are several problems in applying k-means in practice. First of all, it is an NP-hard problem. This means that, according to the current state of knowledge of complexity theory, we do not know any deterministic algorithm that can solve this problem efficiently. Fortunately, there are efficient heuristic algorithms that converge to a local optimum.

Lloyd's algorithm, proposed in 1957, is possibly the most widely used, due to its efficiency and simplicity. It consists of a two-step procedure, in which centroids and associated labels are alternatively updated until convergence. However, the convergence rate can be very small. Also, since it may converge to a suboptimal result, which often depends on the initialization of centroids, it is necessary to run this algorithm multiple times with different initial centroids, aiming for a better result, i.e., a smaller loss. This further increases the expected time (and the expected number of calculations) Lloyd's algorithm could take to get to the actual minimum with a certain rate of confidence.

While Lloyd's algorithm is surprisingly simple and is shown to work quite well in practice, many attempts to approach the problem have been proposed over the years. Another popular algorithm is Hartigan's algorithm. This consists in iterating over all datapoints and verifying whether a reassignment to a different cluster decreases the total cost, until a convenient reassignment is no longer present. Hartigan's algorithm has been proven to perform no

worse than Lloyd's in terms of final cost. Empirically, Hartigan's usually converge to a better minimum with respect to Lloyd's, but it is also slower. Thus, the trade-off between accuracy and complexity does not always imply that Hartigan's algorithm is preferable.

We propose a new approach, which we'll call extended-Hartigan, aiming to reduce the computational cost of the standard Hartigan's algorithm. Basically, it consists of finding a list of possible candidates among all the datapoints, whose individual reassignment would decrease the total cost of the clustering, and trying to reassign them all, hoping the total cost actually decreases. Instead of reassigning each point singularly and recalculating the affected centroids each time, this would allow to perform an entire bunch of reassignments and reduce the number of operations needed.

1.1 Materials and implementation

A complete Python implementation of the algorithms described throughout the entire work is publicly available on the following GitHub repository, together with the datasets used for our tests: [@@ link here @@]. More details about datasets and their authors are contained in Section 4.1.

[@@ Notes @@]

- k-means is slow: superpolinomial running time [AV06] [LZ16]
- Interesting proofs on the convergence of Hartigan's can be found in [TV10]
- TODO:
  - Add GitHub link above
  - Remove this section

Chapter 2

Initializations

Before diving into the actual implementations of k-means algorithms and how these can be improved, we focus on the initialization methods. As we anticipated, centroid initialization can make a crucial difference in performing k-means . Several proposals have been made in different papers: here we present the most classical and widely used initialization techniques.

2.1 random

One of the simplest approaches is the random one: k points are chosen uniformly at random among the dataset to become the initial centroids. The pseudo-code is rather straightforward and is illustrated in Algorithm 1.

Algorithm 1: random initialization

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$
Output: centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$

1: Select k points $\{x_{i_1}, \ldots, x_{i_k}\}$ at random and assign $\mu_j = x_{i_j}$ for $j \in \{1, \ldots, k\}$

Instead of randomly choosing some centroids, and then assigning all datapoints to the nearest centroids, a similar approach is to assign random labels to datapoints, and then calculate the initial centroids as the barycenter of the points assigned to each class. This approach is described in Algorithm 2.

Algorithm 2: random-data initialization

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$
Output: centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$

```
for datapoint x ∈ X do
    Assign x to one of the k clusters at random
end
Compute μ_i as the means of the points assigned to cluster i for i ∈ {1, 2, ..., k}
```

Sampling centroids at random is a simple and time-convenient initialization. However, it is highly probable that the initial centroids are chosen "badly". This means that k-means algorithm, performed with such initial centroids, can take lots of iterations to converge and often returns a sub-optimal result. For example, we could have a dataset with four clusters clearly separated, but get as initial centroids four near points: k-means needs to perform several updates to move the centroids to more appropriate positions, with no guarantees that clusters will be correctly identified. An example of the above claim can be found in Figure 2.1.

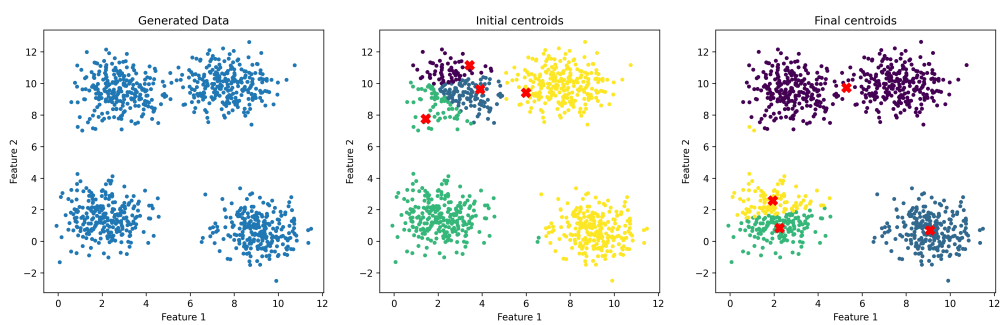

Figure 2.1: A possible random initialization for a 2-dimensional synthetic dataset composed of 1,000 datapoints in 4 clusters, and the final result found by Lloyd's algorithm given such initialization.

The algorithms we present in the following sections describe more clever ways of initializing centroids.

2.2 maximin

Heuristically, initializing centroids more sparsely can give a better guess of the actual position of the centroid. A straightforward approach is to sample the first centroid randomly, and then to choose the subsequent ones as those data points that maximize the minimum distance from the already chosen centroids. This can be written as

$$\boldsymbol{\mu}_i = \arg\max_{\boldsymbol{x} \in X} \min_{j \in \{1,\ldots,i-1\}} \|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2 \qquad \text{for } i \in \{2, 3, \ldots, k\}. \tag{2.1}$$

By repeatedly considering the furthest point from the closest centroid, we get a more evenly distributed initialization of centroids. The `maximin` initialization pseudocode is described in Algorithm 3.

Algorithm 3: maximin initialization

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$
Output: centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$

```
Choose μ_1 randomly from X
for i = 2, ..., k do
    for datapoint x ∈ X do
        D(x) = min_{j∈{1,...,i−1}} ‖x − μ_j‖²        // Squared distance to closest centroid
    end
    μ_i = arg max D(x)        // Select point with max distance from closest centroid
end
```

2.3 k-means++

Another effective initialization method is `k-means++`, proposed by Arthur and Vassilvitskii [AV07]. Similarly to `maximin` initialization, the first centroid is chosen at random, while the next ones are sampled among the datapoints according to a probability distribution that is proportional to the squared distance from the closest centroid of each datapoint. Namely, to

assign the i-th centroid ($i \in \{2, \ldots, k\}$), we calculate for each $\boldsymbol{x} \in X$

$$D(\boldsymbol{x}) := \min_{j \in \{1,\ldots,i-1\}} \|\boldsymbol{x} - \boldsymbol{\mu}_j\|^2 \tag{2.2}$$

which is the squared distance between the point and the closest centroid among the ones already set. By normalizing such distances, we obtain a probability distribution from which we sample the centroid:

$$\boldsymbol{\mu}_i = \boldsymbol{x}' \in X \quad \text{with probability} \quad \frac{D(\boldsymbol{x}')}{\sum_{\boldsymbol{x} \in X} D(\boldsymbol{x})} \tag{2.3}$$

This initialization is proven to produce a result which is $\Theta(\log k)$-competitive with the optimal clustering, and in practice makes a dramatic improvement both in speed and accuracy of k-means clustering with respect to `random` initialization. The pseudocode is shown is Algorithm 4.

Algorithm 4: k-means++ initialization

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$
Output: centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$

```
Choose μ_1 randomly from X
for i = 2, ..., k do
    for datapoint x ∈ X do
        D(x) = min_{j∈{1,...,i−1}} ‖x − μ_j‖^2        // Squared distance to closest centroid
    end
    Select μ_i = x' ∈ X with probability D(x')/Σ_{x∈X} D(x)
end
```

2.4 greedy k-means++

A possible improvement to `k-means++` initialization algorithm is mentioned in the original paper [AV07] and further explored by Vassilvitskii in his PHD thesis. It consists of a greedy application of `k-means++`, where we sample t candidates based on the probability distribution

described in Equation 2.3 and we choose the one that decreases the cost as much as possible as a centroid. So far, not many works have concentrated on the analysis of this initialization technique, and no strong theoretical guarantee has been found. Bhattacharya et al. [Bha+19] have shown that `greedy-k-means++` is $\Omega(t \log k)$ approximate in expectation, while Grunau et al. [Gru+22] have found nearly matching lower and upper bounds of $O(t^3 \log^3 k)$ and $\Omega\left(\frac{t^3 \log^2 k}{\log^2(t \log k)}\right)$, respectively. Empirically, the greedy approach appears to outperform the standard `k-means++` initialization even by setting $t = 2$, as done by Vassilvitskii. Other works advertise the choice of t from $\Theta(\log k)$, which is the choice taken in the popular Scikit-learn Python library [Sci], which implements `greedy-k-means++` as default initialization for k-means algorithm, and sets $t = \lfloor 2 + \ln k \rfloor$. We will replicate this choice in our implementation of this initialization, whose pseudocode is detailed in Algorithm 5.

Algorithm 5: greedy k-means++ initialization

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$, $t =$ number of trials
Output: centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$

```
Choose μ_1 randomly from X
for i = 2, ..., k do
    for datapoint x ∈ X do
        D(x) = min_j ||x − μ_j||^2          // Squared distance to closest centroid
    end
    P(x) = D(x) / Σ_{x'∈X} D(x)
    candidates = sample t candidate points based on P(x)
    for c in candidates do
        temp_centroids = {μ_1, ..., μ_{i−1}} ∪ {c}
        Assign points to closest centroids in temp_centroids
        C(c) = tot cost of current clustering
    end
    μ_i = arg min_c C(c)
end
```

Chapter 3

Algorithms

Having discussed several possible initializations, we can now concentrate on the core of the algorithms that perform the k-means clustering. These algorithms define how centroids are updated and how data points are reassigned across iterations. While the objective remains the same, i.e., minimizing the Within-Cluster Sum of Squares (WCSS) whose formula is 1.1, the update rules and convergence criteria differ significantly. This impacts both the efficiency and performance of clustering.

In this chapter, we will first deal with the classical Lloyd's algorithm and another long-known algorithm by Hartigan. Then we will propose and motivate a modified approach which we will call `extended-Hartigan`, together with some variations of these various algorithms with the objective to combine the main strengths of each of them.

3.1 Lloyd's Algorithm

As anticipated before in this work, the simplest and still most widely used algorithm for k-means clustering was ideated by Stuart P. Lloyd in Bell Labs. It was originally developed in 1957 for pulse-code modulation (PCM) in signal processing, which is a method for representing analog signals with discrete pulses, aiming to optimize quantization by minimizing the mean squared error between input signals and their quantized representations. Although widely circulated within Bell Labs, Lloyd's work remained unpublished until 1982, when it

appeared in the IEEE Transactions on Information Theory [Llo82].

The algorithm consists of three steps, (more details are presented in the pseudocode Algorithm 6):

1. **Assignment Step:** Assign each data point to the closest centroid, forming k clusters:

$$\mu(x_i) = \arg\min_{j \in \{1,\dots,k\}} \|x_i - \mu_j\| \quad \text{for } i \in 1, \dots, n$$

2. **Update Step:** Move each centroid to the barycenter of the points assigned to the corresponding cluster:

$$\mu_j = \frac{\sum_{i=1}^{n} \mathbb{1}_{[\mu(x_i)=j]} x_i}{\sum_{i=1}^{n} \mathbb{1}_{[\mu(x_i)=j]}} \quad \text{for } j \in 1, \dots, k$$

3. **Convergence Check and Iteration:** If no data point has been reassigned (or equivalently, the centroid was not moved), return the clustering. Otherwise, repeat steps 1 and 2.

An interesting fact about Lloyd's algorithm is that its objective function is guaranteed to decrease (or remain constant) in each iteration, as claimed by the following Lemma.

Algorithm 6: lloyd

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$, initial centroids $\{\mu_1, \mu_2, \dots, \mu_k\}$
Output: final centroids $\{\mu_1, \mu_2, \dots, \mu_k\}$, clusters $\{S_1, \dots, S_k\}$

```
repeat
    for datapoint x ∈ X do
        μ(x) = arg min_j ‖x − μ_j‖²          // assign x to the closest centroid
    end
    for cluster j = 1, ..., k do
        S_j = set of points in cluster j
        μ_j = (1/|S_j|) Σ_{x∈S_j} x          // update centroid to the mean of its points
    end
until convergence;
```

Lemma 1. *Let* $\boldsymbol{\mu}_i = \frac{1}{|S_i|}\sum_{\boldsymbol{x}\in S_i}\boldsymbol{x}$ *the centroid of each of* k *clusters* $\{S_1, \ldots, S_n\}$*. Then, each iteration of Lloyd's algorithm does not increase the cost*

$$C(S_1, \ldots, S_k) = \sum_{i=1}^{k}\sum_{\boldsymbol{x}\in S_i}\|\boldsymbol{x} - \boldsymbol{\mu}_i\|^2 . \tag{3.1}$$

Proof. Let $\left\{S_1^{(t-1)}, \ldots, S_k^{(t-1)}\right\}$ the clusters at the $(t-1)$-th iteration of the algorithm, and let $\boldsymbol{\mu}_i^{(t-1)}$ the centroid of cluster $S_i^{(t-1)}$, i.e.,

$$\boldsymbol{\mu}_i^{(t-1)} = \frac{1}{\left|S_i^{(t-1)}\right|}\sum_{\boldsymbol{x}\in S_i^{(t-1)}}\boldsymbol{x}. \tag{3.2}$$

Firstly, we have the assignment step: fixing centroids, we update the partition. Let $\{S_1^{(t)}, \ldots, S_k^{(t)}\}$ the new clusters at iteration t, obtained by assigning each point to the nearest centroid $\mu_i^{(t-1)}$ from the previous step. The new partition is constructed by minimizing the cost function over all possible partitions while keeping the centroids fixed at $\boldsymbol{\mu}_i^{(t-1)}$, so it holds

$$\sum_{i=1}^{k}\sum_{\boldsymbol{x}\in S_i^{(t)}}\left\|\boldsymbol{x} - \boldsymbol{\mu}_i^{(t-1)}\right\|^2 \leq \sum_{i=1}^{k}\sum_{\boldsymbol{x}\in S_i^{(t-1)}}\left\|\boldsymbol{x} - \boldsymbol{\mu}_i^{(t-1)}\right\|^2, \tag{3.3}$$

since we are assigning each point to the closest centroid, minimizing the squared distance.

Secondly, we have the update step: fixing the partition, we update the centroids. Since the arithmetic mean (in this case the centroid $\boldsymbol{\mu}_i^{(t)}$) minimizes the sum of squared errors, we have that

$$\sum_{\boldsymbol{x}\in S_i^{(t)}}\left\|\boldsymbol{x} - \boldsymbol{\mu}_i^{(t)}\right\|^2 \leq \sum_{\boldsymbol{x}\in S_i^{(t)}}\left\|\boldsymbol{x} - \boldsymbol{\mu}_i^{(t-1)}\right\|^2, \tag{3.4}$$

so, summing over all clusters, it holds

$$\sum_{i=1}^{k}\sum_{\boldsymbol{x}\in S_i^{(t)}}\left\|\boldsymbol{x} - \boldsymbol{\mu}_i^{(t)}\right\|^2 \leq \sum_{i=1}^{k}\sum_{\boldsymbol{x}\in S_i^{(t)}}\left\|\boldsymbol{x} - \boldsymbol{\mu}_i^{(t-1)}\right\|^2. \tag{3.5}$$

By putting together equations (3.3) and (3.5), we obtain

$$C(S_1^{(t)}, \ldots, S_k^{(t)}) = \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i^{(t)}} \left\| \boldsymbol{x} - \boldsymbol{\mu}_i^{(t)} \right\|^2 \leq \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i^{(t)}} \left\| \boldsymbol{x} - \boldsymbol{\mu}_i^{(t-1)} \right\|^2 \tag{3.6}$$

$$\leq \sum_{i=1}^{k} \sum_{\boldsymbol{x} \in S_i^{(t-1)}} \left\| \boldsymbol{x} - \boldsymbol{\mu}_i^{(t-1)} \right\|^2 = C(S_1^{(t-1)}, \ldots, S_k^{(t-1)}) \tag{3.7}$$

which proves the claim. □

Even though the WCSS cost is monotonically decreasing, we have no guarantees that the algorithm will converge to a global minimum. Lloyd's algorithm is very susceptible to local minima, depending on the centroids' initialization: for this reason, it is necessary to reinitialize the algorithm multiple times, tracking the final output and keeping the centroids that produce the least cost, hoping to get a better clustering.

3.2 Hartigan's Algorithm

An alternative heuristic to solve the k-means problem is John A. Hartigan's algorithm. Instead of optimizing the cost function by updating entire batches of data (i.e., recalculating centroids and reassigning all points at each iteration), Hartigan's intuition was to focus on optimizing cost locally, by performing point-wise updates. The first hints about this different heuristic are contained in his book "Clustering Algorithms" from 1975 [Har75], but the algorithmic version was formalized some years later, together with M. Antony Wong [HW79].

The algorithm consists of the following steps:

1. **Iterate all data points:** for each data point:
    - Evaluate whether moving it to a different cluster decreases the overall cost.
    - If such a move exists, reassign the data point the the cluster which provides the greatest reduction in cost.

- Update the centroids of the affected clusters.

2. **Check for convergence:** If no data point has been reassigned in step 1, return the clustering. Otherwise, repeat step 1.

A more detailed pseudocode of how Hartigan's algorithm works can be found in Algorithm 7.

Interestingly, while Lloyd's algorithm only iterates if a data point from some cluster is closer to the centroid of some other cluster than to the centroid of its current one, Hartigan's method considers how the centroids involved would shift after the reassignment itself. As a result, a point can be reassigned even if it is already closest to its current cluster center [TV10].

To see how the overall cost can be calculated in this case, we consider a set of z points in $\mathbb{R}^d$, and let $\mu = \frac{1}{z}\sum_{i=1}^{z} x_i$ their barycenter. Basically, we can interpret this set as a cluster,

Algorithm 7: hartigan

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$, initial centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$
Output: final centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$, clusters $\{S_1, \ldots, S_k\}$

```
repeat
    for datapoint x ∈ X do
        S_d = cluster to which x belongs
        for j = 1, ..., k with j ≠ d do
            Δcost_{d→j} = |S_j|/(|S_j|+1) ‖x − μ_j‖² − |S_d|/(|S_d|−1) ‖x − μ_d‖²
        end
        if min_j Δcost_{d→j} < 0 then
            m = arg min_j Δcost_{d→j}
            reassign x to cluster S_m
            update μ_d and μ_m
        end
    end
until no point is reassigned;
```

with μ its centroid. Then, the sum-of-squares cost of the cluster is

$$\begin{aligned}
\text{cost} &:= \sum_{i=1}^{z} \|x_i - \mu\|^2 && (3.8)\\
&= \sum_{i=1}^{z} \|x_i\|^2 - 2\sum_{i=1}^{z} \langle x_i, \mu\rangle + \sum_{i=1}^{z} \|\mu\|^2 && (3.9)\\
&= \sum_{i=1}^{z} \|x_i\|^2 - 2z \left\langle \frac{1}{z}\sum_{i=1}^{z} x_i, \mu \right\rangle + z\|\mu\|^2 && (3.10)\\
&= \sum_{i=1}^{z} \|x_i\|^2 - 2z\|\mu\|^2 + z\|\mu\|^2 && (3.11)\\
&= \sum_{i=1}^{z} \|x_i\|^2 - z\|\mu\|^2 && (3.12)
\end{aligned}$$

By expanding the norm in equation (3.8), using the linearity of the inner product in (3.10) and the definition of μ in (3.12), we can interpret the cost of a cluster as z times its variance.

Now suppose we want to remove the first s points from such set ($s \leq z$). Let's denote the barycenter of this removed set as

$$g = \frac{1}{s}\sum_{i=1}^{s}, \tag{3.13}$$

and the barycenter of the remaining points as

$$\mu' = \frac{1}{z-s}\sum_{i=s+1}^{z} x_i = \frac{1}{z-s}\left(\sum_{i=1}^{z} x_i - \sum_{i=1}^{s} x_i\right) = \frac{z\mu - sg}{z-s}. \tag{3.14}$$

Analogously to equation (3.8), the new cost of the set of remaining points is then

$$\text{cost}' = \sum_{i=s+1}^{z} \|x_i - \mu'\|^2 = \sum_{i=s+1}^{z} \|x_i\|^2 - (z-s)\|\mu'\|^2. \tag{3.15}$$

Then, the variation of the cost of removing such s points from the set can be calculated as follows:

$$
\begin{aligned}
\Delta\text{cost} &= \text{cost}' - \text{cost} && (3.16)\\
&= \left(\sum_{i=s+1}^{z} \|x_i\|^2 - (z-s)\,\|\mu'\|^2\right) - \left(\sum_{i=1}^{z} \|x_i\|^2 - z\,\|\mu\|^2\right) && (3.17)\\
&= -\sum_{i=1}^{s} \|x_i\|^2 - \frac{1}{z-s}\,\|z\mu - sg\|^2 + z\,\|\mu\|^2 && (3.18)\\
&= -\sum_{i=1}^{s} \|x_i\|^2 - \frac{1}{z-s}\left(z^2\,\|\mu\|^2 + s^2\,\|g\|^2 - 2zs\,\langle\mu, g\rangle\right) + z\,\|\mu\|^2 && (3.19)\\
&= -\sum_{i=1}^{s} \|x_i\|^2 - \frac{1}{z-s}\left(z^2\,\|\mu\|^2 + s^2\,\|g\|^2 - 2zs\,\langle\mu, g\rangle - z^2\,\|\mu\|^2 + zs\,\|\mu\|^2\right) && (3.20)\\
&= -\sum_{i=1}^{s} \|x_i\|^2 - \frac{1}{z-s}\left(s^2\,\|g\|^2 - 2zs\,\langle\mu, g\rangle + zs\,\|\mu\|^2\right) && (3.21)\\
&= -\sum_{i=1}^{s} \|x_i\|^2 - \frac{s^2}{z-s}\,\|g\|^2 - \frac{zs}{z-s}\left(-2\,\langle\mu, g\rangle + \|\mu\|^2\right) && (3.22)\\
&= -\sum_{i=1}^{s} \|x_i\|^2 - \frac{s^2}{z-s}\,\|g\|^2 - \frac{zs}{z-s}\left(-\,\|g\|^2 + \|\mu - g\|^2\right) && (3.23)\\
&= -\sum_{i=1}^{s} \|x_i\|^2 - \frac{s^2}{z-s}\,\|g\|^2 + \frac{zs}{z-s}\,\|g\|^2 - \frac{zs}{z-s}\,\|\mu - g\|^2 && (3.24)\\
&= -\underbrace{\left(\sum_{i=1}^{s} \|x_i\|^2 - s\,\|g\|^2\right)}_{(\star)} - \underbrace{\frac{zs}{z-s}\,\|\mu - g\|^2}_{(\star\star)} && (3.25)
\end{aligned}
$$

where in equation (3.18) we used (3.14), in (3.19) we expanded the norm, and 3.23 derives from the fact that $\|\mu - g\|^2 = \|\mu\|^2 - 2\,\langle\mu, g\rangle + \|g\|^2$. Notice that we recovered the variance of the removed set (with barycenter g) times its size s ($\star$), and a correction term which is due to the movement of the centroid ($\star\star$).

Therefore, given two clusters S_1 and S_2 of sizes z_1, z_2 and centroids μ_1, μ_2, respectively, if we want to move s points of barycenter g from S_1 to S_2, we will get a sum-of-squares cost decrease for S_1 and increase for S_2. The Δcost calculation for S_2 is completely analogous to the one in (3.16); in particular, the variance part is the same but with an opposite sign since

we are now adding the set of data points to the cluster. The resulting variation in the total cost caused by this reassignment is given by:

$$\Delta\text{cost}_{1\to 2} = s\left(\frac{z_2}{z_2+s}\left\|\mu_2 - g\right\|^2 - \frac{z_1}{z_1-s}\left\|\mu_1 - g\right\|^2\right) \tag{3.26}$$

In the special case in which we are reassigning just one point, say $x \in X$, we have $s = 1$ and $g = x$, which leads to:

$$\Delta\text{cost}_{1\to 2} = \frac{z_2}{z_2+1}\left\|\mu_2 - x\right\|^2 - \frac{z_1}{z_1-1}\left\|\mu_1 - x\right\|^2 \tag{3.27}$$

Notice that the squared norms are just the contribution of each single data point to the change in the overall cost, which are the same quantities involved in Lloyd's algorithm. However, the decision of whether to accept the reassignment is not the same as in Lloyd's procedure, due to the presence of the prefactors $\frac{z_1}{z_1-1}$ and $\frac{z_2}{z_2+1}$. This, as anticipated, allows reassignments that would not be performed by Lloyd's algorithm despite being convenient in terms of cost (the converse does not happen). This justifies the following Lemma:

Lemma 2. *The fixed points of Hartigan's scheme are a subset of the fixed points of Lloyd's scheme.*

Indeed, in Lloyd's algorithm, a fixed point occurs when no point is closer to another cluster's centroid than its own. In Hartigan's algorithm, instead, a fixed point occurs when reassigning any point (and updating centroids) would not lower the total cost. The insight is that each final configuration reached by Lloyd's algorithm might be reached by Hartigan's, but the latter can possibly find them as suboptimal and continue the optimization process. Therefore, Hartigan's algorithm usually leads to better clusterings (clusterings with lower WCSS) and is usually more robust to bad initializations.

The drawback of Hartigan's algorithm is the computational complexity: the scheme requires a high number of calculations, by simulating the reassignment of each datapoint to each

centroid for each iteration (so, until all datapoints and centroids are checked and no update has been made). Also, since the Δcost formula relies on the assumption that centroids are located in the barycenters of the clusters, we must recompute the centroids and the cost of the other points in the two clusters involved.

3.3 Extended-Hartigan's Algorithm

To address the limitations of Hartigan's algorithm, highlighted in the previous section, we propose a slightly different approach that doesn't rely on the need to recalculate the centroids and the related costs for each data point which is reassigned.

The idea is to attempt reassigning all data points to a cluster at once, in a way that would reduce the cost, as defined by equation (3.27). However, instead of updating the position of the centroids of the two clusters affected by each reassignment, we skip this step, thereby breaking the assumption of using the most up-to-date centroids. Intuitively, each iteration can be thought of as taking a snapshot of the current state, applying Hartigan's scheme independently for each data point, and doing all convenient reassignments together at once. The goal is to obtain, within a single iteration, a much bigger decrease in the total cost than by performing single local updates. The underlying hope is that each individual reassignment won't significantly distort the centroids, or that the combined effect of all reassignments will balance out the inaccuracy introduced by using outdated centroid positions. Only after accepting all proposed reassignments, we check if the total cost actually decreased; in case it didn't, we roll back to the previous situation and proceed in a "safe mode", similar to what Hartigan's algorithm would have done, to avoid "bad" reassignments.

Extended-Hartigan algorithm works as follows:

1. **Find candidates:** for each data point:
    - Compute Δcost as in equation (3.27) of reassigning it to each different centroid.
    - Any time the result is negative, store it in a "candidates list", together with the

information about the move. If a point has more than one potential better cluster, pick the one with the most negative Δcost.

2. **Convergence check:** if the candidates list is empty, return the clustering. Otherwise, proceed to next step.

3. **Accept candidates**: depending on the operational mode do one of the following:

   - In "unsafe" mode, accept all proposed reassignments and check whether the overall cost actually decreased. In case it didn't, revert all changes and proceed in "safe" mode.

   - In "safe" mode, sort the proposed reassignments by their produced Δcost (the most negative first), then accept them in order, but only if the operation does not involve clusters which were already involved in some reassignments. In this way, the assumptions of formula (3.27) hold, ensuring the overall cost decreases. At most $\frac{k}{2}$ moves will be performed (where k is the number of clusters).

4. **Iterate:** repeat all the steps from step 1.

At first sight, this algorithm may seem quite different to Hartigan's scheme. However, looking at its pseudocode (Algorithm 8), we can notice that classical Hartigan's algorithm consists just of running the first step ("Find candidates") and returning the candidates list as soon as it is no longer empty (so it contains just one candidate). Then, the "unsafe" and "safe" modes basically collapse into the same operation (the reassignment of the only element in the candidates list), and we proceed to he next iteration.

We have no theoretical guarantees about this new scheme, but since the reassignments made by Hartigan's algorithm can be seen as a special case of extended-Hartigan, we do not expect worse results. In particular, if accepting full candidate lists does not decrease the overall cost, we end up making point-wise updates that Hartigan's algorithm would have found. More in general, starting from the convergence configuration of one of the two algorithms, the other

will not be able to perform any updates: the difference between the schemes lies just in the update dynamics, indicating that they should intuitively have the same fixed points.

Algorithm 8: extended-hartigan

Input: $k \in \mathbb{R}$, $X \in \mathbb{R}^{n+d}$, initial centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$
Output: final centroids $\{\mu_1, \mu_2, \ldots, \mu_k\}$, clusters $\{S_1, \ldots, S_k\}$

```
repeat
    initialize empty candidates_list
    1. Find candidates
    for datapoint x_i ∈ X do
        S_d = cluster to which x_i belongs
        for j = 1, ..., k with j ≠ d do
            Δcost_{d→j} = |S_j|/(|S_j|+1) ||x_i − μ_j||^2 − |S_d|/(|S_d|−1) ||x_i − μ_d||^2
        end
        if min_j Δcost_{d→j} < 0 then
            m = arg min_j Δcost_{d→j}
            add x_i to candidates_list together with d, m and Δcost_{d→m}
        end
    end
    2. Unsafe Mode
    for x^(i) in candidates_list do
        reassign x^(i) to cluster S_{m^(i)}                    // accept all candidates
    end
    if total cluster cost is decreased then
        accept all reassignments
        proceed to (4.) and start next iteration
    else
        rollback to original assignment
        proceed to safe mode (3.)
    end
    3. Safe Mode
    sort candidates_list in order of increasing Δcost
    for x^(i) in candidates_list do
        if S_{d^(i)} and S_{m^(i)} are unchanged then
            reassign x^(i) to cluster S_{m^(i)}                // accept at most one edit per cluster
        end
    end
    4. End Iteration
    update each centroid to the mean of points assigned to it
until candidates_list is empty;
```

3.4 Mixed Modes

To combine the advantages of the methods proposed above, we can design hybrid strategies which leverage the strengths of each approach. In particular, we can first run Lloyd's algorithm up to convergence, taking advantage of its simplicity and speed to quickly reach a reasonable local minimum. Once this initial convergence is achieved, we can switch to another method, running either Hartigan or extended-Hartigan to perform more granular and point-wise reassignments and find a more refined solution. This two-phase strategy allows us to benefit from Lloyd's efficiency in the early stages and from Hartigan's finer solution in the later ones, increasing the chances of obtaining a better clustering. Such mixed-mode procedures can be particularly useful in practice, where we need to carefully balance computational efficiency and solution quality.

3.5 Other Heuristics

Inspired by the extended-Hartigan variation of Hartigan's scheme, we can design some other modifications. For instance we can code a "safe-Hartigan" version, which proceeds as the extended-Hartigan algorithm by discarding the "unsafe" mode and proceeding always in "safe" mode, by accepting just some of the candidates so that the overall cost certainly decreases. This is not very dissimilar by the standard Hartigan's scheme. But, while in a single iteration of the latter, multiple data points can be removed from the same cluster (by, however, keeping centroids always update), a safe-Hartigan iteration needs to go over all data points, but at the end performs at most $\frac{k}{2}$ reassignments. As expected, some preliminary tests suggested that this method is just more computationally intensive than running the standard Hartigan's algorithm: while we save on centroids movement, we spend lots of calculations to simulate each possible reassignment of datapoints, but then perform just a few reassignments. This is not very convenient, considering that usually $k \ll n$.

Another possible heuristic, which we'll call "binary-Hartigan", may consist in modifying

extended-Hartigan's safe mode, by proceeding in a "divide and conquer" fashion: if accepting all candidates does not decrease the total cost, we recursively split the candidates list in half, and try to accept each portion in turn. Again, the objective would be to accept as many candidates as possible, hoping to decrease the overall cost of clustering much more than single reassignment could do. While this approach is more promising than the first one, we will show in the next chapters that testing on real datasets, reassignments made by extended-Hartigan are usually convenient, and the algorithm rarely enters safe mode, which discourages the effort of performing extra operations in a "binary" mode, as suggested above (more details in Section 4.3.5).

Chapter 4

Simulations

4.1 Datasets

We will now show the experimental results obtained through some simulations. We will use some classical datasets for clustering tasks: *A-Sets* from a research project by Kärkkäinen and Fränti [KF02], *Bridge* and *House*, from a work by Fränti, Rezaei, and Zhao [FRZ14]. The table below summarizes the main characteristics of each dataset, such as number of samples N, dimensionality D, and number of clusters k.

Dataset	N	D	k
A1	3000	2	20
A2	5250	2	35
A3	7500	2	50
Bridge	4096	16	$5^\star$
Bridge	4096	16	$10^\star$
Bridge	4096	16	$100^\star$
House	34112	3	$50^\star$

Table 4.1: Information about datasets we will use for our clustering task. Values of k marked with a star ($\star$) indicate that the dataset does not have a definitive or "true" number of clusters. However, we will still perform k-means clustering using these values for simulation purposes.

These datasets were chosen to evaluate the differences between classical and proposed k-means clustering algorithms under varying number of samples, dimensionalities and cluster counts.

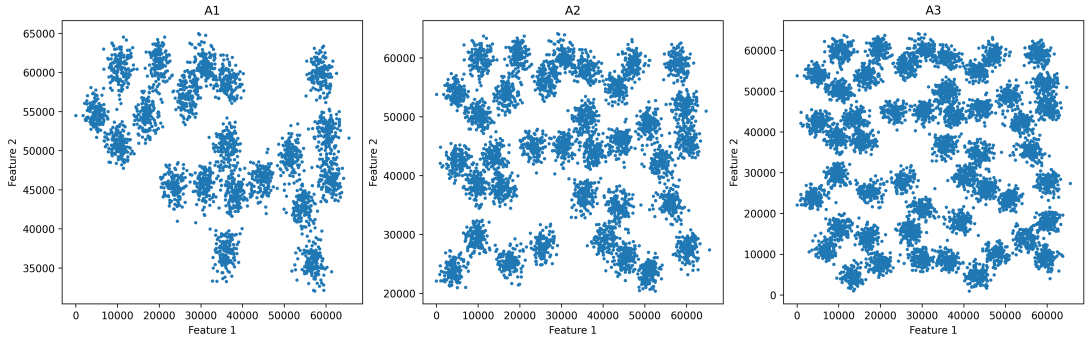

Figure 4.1: *A-Set* datasets.

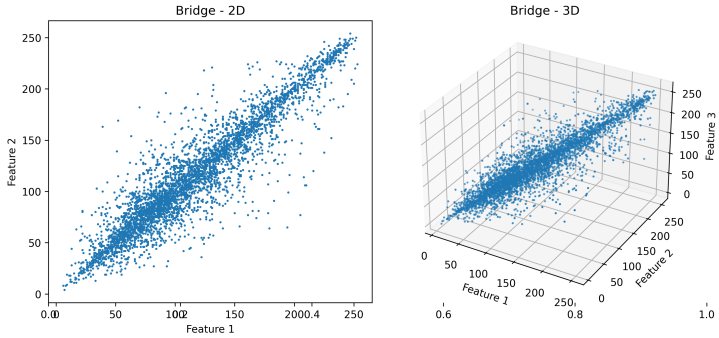

Figure 4.2: *Bridge* dataset.

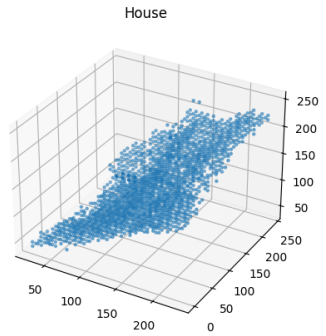

Figure 4.3: *House* dataset.

4.2 Metrics

4.2.1 Centroid Index (CI)

While *Bridge* and *House* datasets don't have a specific predefined number of classes and exhibit more irregular, overlapping structures – indeed we will perform the simulations with different values of k and analyze the final cost – *A-Sets* datasets are composed of synthetic two-dimensional data points aggregated in specific numbers of clusters, so are known to contain compact, well-separated clusters. Therefore, the centroid's "true" position is well defined. Notice that there's a duality between the partition and the centroids: given one of them, the other can be determined using equation (1.2) or (1.3). However, we are often interested in a high-level view of how similar the overall clustering structures are, rather than the correct assignment of each single data point. For this reason, we can use the Centroid Index (CI) as a metric to assess the cluster-level differences between the true and

ground truth [FRZ14]. More specifically, given two sets of centroids $\boldsymbol{\mu} = \{\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_k\}$ and $\boldsymbol{\mu}' = \{\boldsymbol{\mu}'_1, \ldots, \boldsymbol{\mu}'_k\}$ we construct a nearest neighbor mapping $\boldsymbol{\mu} \to \boldsymbol{\mu}'$ as follows:

$$q_i = \arg \min_{j \in \{1,\ldots,k\}} \left\| \boldsymbol{\mu}_i - \boldsymbol{\mu}'_j \right\| \qquad \forall i \in \{1, \ldots, k\} \tag{4.1}$$

For each target centroid $\boldsymbol{\mu}'_j$, we then analyze how many $\boldsymbol{\mu}_i$'s consider it as nearest and consider as "orphans" clusters which are not associated to any other:

$$orphan(c'_j) = \begin{cases} 0 & \exists\ i : q_i = j \\ 1 & \text{otherwise} \end{cases} \tag{4.2}$$

We finally count how many orphan clusters we are left:

$$CI(\boldsymbol{\mu}, \boldsymbol{\mu}') = \sum_{j=1}^{k} orphan(c'_j) \tag{4.3}$$

Clearly, two clusterings have the same cluster-level structure if their centroid index is 0, while any other value indicates that there is a discrepancy in the positioning of corresponding cluster centroids. This means that, although the number of clusters and possibly even the data point assignments might be similar, the overall spatial configuration of the clusters differs. A higher centroid index reflects greater dissimilarity between the clusterings, capturing important shifts in cluster centers.

4.2.2 Norm calculations

Due to the different nature of Lloyd-based and Hartigan-based algorithms, we cannot really rely on the number of iterations performed as a metric of comparison between the different schemes. Instead, we will evaluate the computational efficiency using the number of distance calculations, as proposed by Elkan [Elk03] and widely adopted by many other authors in subsequent works. Since distance computations constitute the most expensive operation in each iteration of k-means , especially in high-dimensional or large-scale datasets, tracking how

many such operations are performed provides a machine-independent and algorithm-agnostic way to assess performance, regardless of implementation details or hardware.

4.3 Results

We will now provide some empirical results we obtained by performing 1,000 tests for all combinations of initialization methods and algorithms:

Initialization	Algorithm
	`lloyd`
`k-means++`	`hartigan`
`maximin`	`extended-hartigan`
`greedy k-means++`	`mixed-hartigan`
	`mixed-extended-hartigan`

We will not produce simulations using random initialization methods, which have been widely shown to produce suboptimal results. Instead, we will concentrate on the other more effective ways of initialization, as described in Chapter 2.

4.3.1 Stability

The analysis of the Centroid Index (CI) for A-Sets datasets, summarized in tables A.1, A.2, and A.3, reveals that the initialization strategy is definitely the main factor in determining final clustering quality. While `maximin` initialization method generally outperforms `k-means++`, the greedy approach consistently delivers a significantly more accurate clustering. However, this comes to a non-negligible cost as highlighted in the tables in Appendix A.1: `greedy-k-means++` needs to perform a good amount of norm calculations, noticeably increasing the computational expense of a single run.

Within this context, our proposed `extended-hartigan` and `mixed` modes demonstrate a consistent advantage, refining the results further by achieving a lower mean CI than both the classical Lloyd's and Hartigan's algorithms. This underscores the stability of these proposed

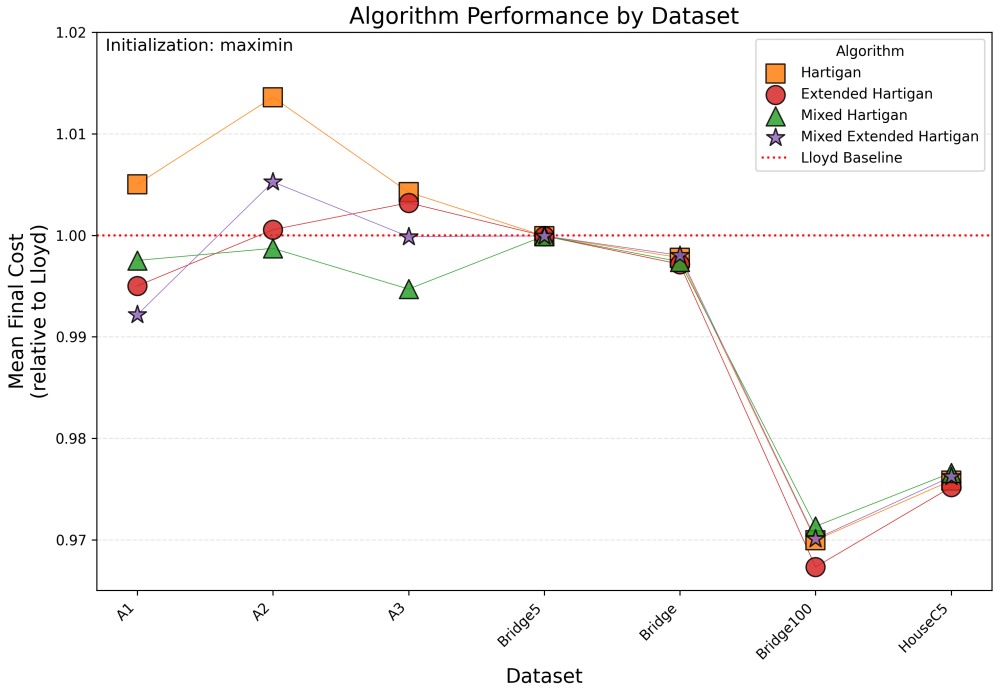

Figure 4.4: Mean final cost obtained by performing 1,000 simulations of the different algorithms on the various datasets using `maximin` initialization (graphs using other initializations are visible in Appendix A.2). Lloyd's results are used as a baseline, represented by the red dotted line.

methods, even though this analysis alone does not yet reveal clear differences.

4.3.2 Mean final cost improvement

An analysis of the mean final cost reveals some benefits of the proposed algorithms. As shown in Figure 4.4, where we chose to include the results obtained using the `maximin` initialization, which showed a good trade off between goodness of results and complexity of initialization (the other plots are included in Appendix A.2), `mixed-hartigan` and `mixed-extended-hartigan` variants consistently achieve a final cost that is better than or comparable to the standard Hartigan algorithm, on average, and frequently improve upon the Lloyd's baseline.

This advantage is evident the *Bridge-100* and *House* datasets, which are the most complex ones, where Hartigan and our variants converge to better solutions, achieving a significant

cost reduction of nearly 3% relative to Lloyd's. This strongly suggests their ability to navigate the solution space more effectively on certain complex data structures. At the same time, when looking for simplier structures, `extended-hartigan` and `mixed` algorithms still achieve good performances with respect both to the standard `lloyd` and `hartigan` schemes, asserting the reasonableness of such methods, even if the additional optimizations introduced do not yield significant improvements.

4.3.3 Minimum final cost improvement

The results contained in the previous section are confirmed by a quick analysis of the minimum final cost obtained. In Figure 4.5 we show the results obtained using the `greedy-k-means++` initialization, that despite being more computationally expensive, is shown to obtain the best results, really allowing to understand the powerfulness of the various schemes (the other cases can be found in Appendix A.3). We can notice that from some excellent starting point, the specific refinement algorithm used doesn't make a difference in the final quality of the best run on the simpler *A-Sets* and *Bridge-5/10* datasets.

This completely changes for *Bridge-100* and *House*: on these more complex datasets, even a powerful initialization is not enough to guarantee an absolute best solution. All the variants are able to find a better final clustering (lower cost) than Lloyd's algorithm. In particular, minima found are comparable to, or even slightly better than what is found using standard Hartigan's algorithm, indicating that these algorithm successfully escape local minima `lloyd` is trapped in.

In this context, extended-Hartigan shows the highest potential: in this best-case scenario, it founds the single best solution on both *Bridge-100* and *House*, demonstrating that its potential to discover the most optimal clustering on the hardest problems, even if its average performance might be less consistent than the mixed modes, as pointed out in Section 4.3.2.

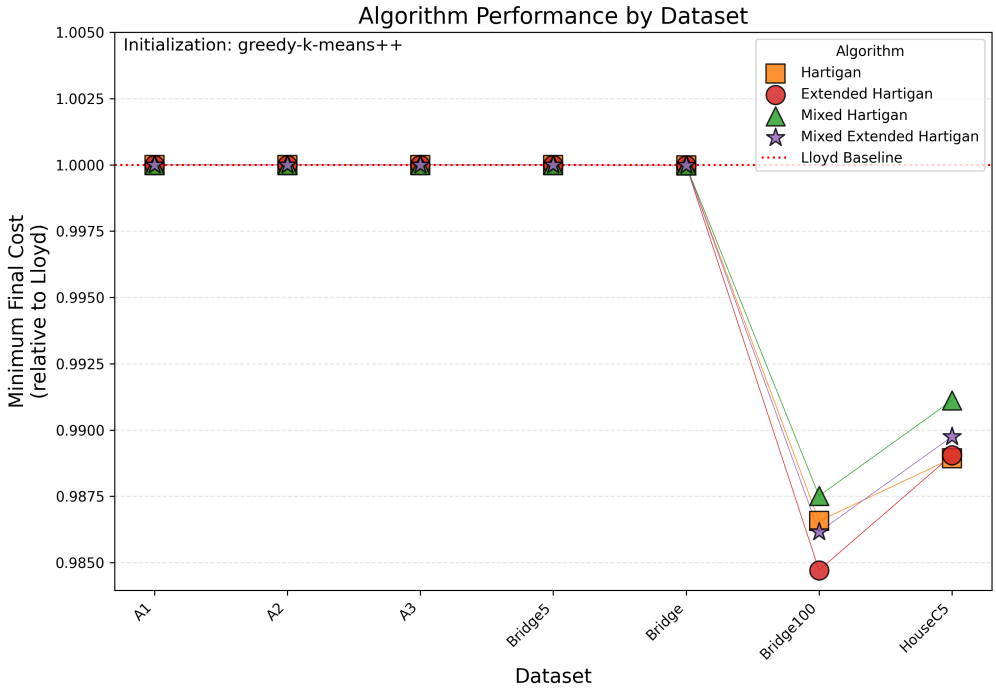

Figure 4.5: Minimum final cost obtained by performing 1,000 simulations of the different algorithms on the various datasets using `greedy-k-means++` initialization (graphs using other initializations are visible in Appendix A.3). Lloyd's results are used as a baseline, represented by the red dotted line.

4.3.4 Mean norm calculations improvement

Beyond the quality of the final clustering, it is crucial to evaluate the computational effort required to achieve it, whose reduction was one of the main goals of this work. Figure 4.6 presents the mean total norm calculations, a proxy for the computational cost of each algorithm, as previously discussed in section 4.2.2.

The most noticeable feature is the inefficiency of the standard Hartigan algorithm on the *Bridge-5* dataset, where, independently from the initialization method used, as one can check looking at Appendix A.4, it required several times the computational effort of the Lloyd's baseline. Again, this effect can be partially mitigated by using a better initialization method such as `greedy-k-means++`, where still 8 times Lloyd's norm calculations are needed on average. This highlights a critical instability in the classical Hartigan's algorithm, where it can potentially enter a cycle of point-swapping with minimal improvement, leading to a

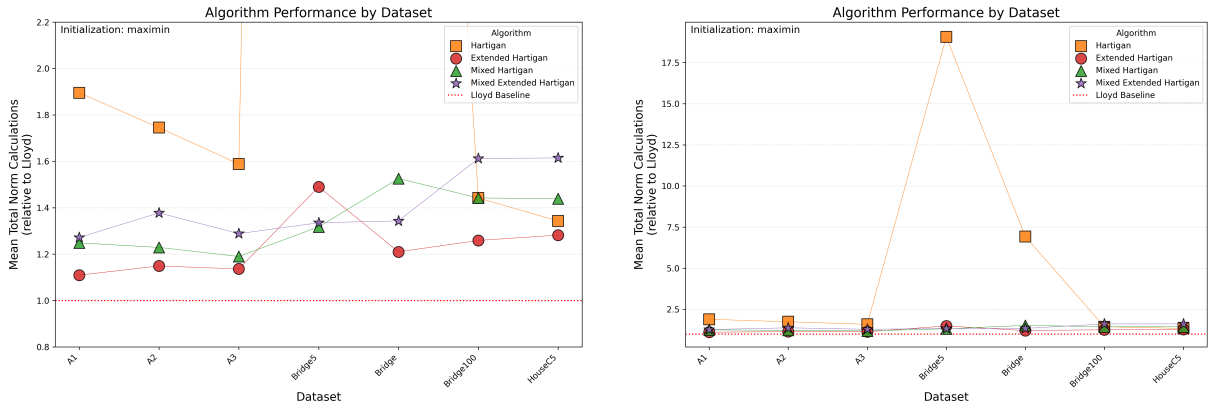

Figure 4.6: Mean norm calculations obtained by performing 1,000 simulations of the different algorithms on the various datasets using `maximin` initialization (graphs using other initializations are included in Appendix A.4). Lloyd's results are used as a baseline, represented by the red dotted line. The right panel displays the full data range, highlighting the extreme computational cost incurred by `hartigan` on *Bridge-5*. The left panel provides a zoomed-in view to enable a clear comparison of the more stable computational performance of the other algorithms across all datasets.

dramatic and unpredictable increase in computational cost.

In contrast, our proposed extended, mixed, and mixed-extended Hartigan algorithms all demonstrate significantly more stable behavior, by successfully avoiding the extreme cost blow-up obtained by the standard Hartigan method on *Bridge-5* and performing a number of norm calculations sometimes comparable to, and in all cases not higher than 60% of Lloyd's.

While all Hartigan's variants inherently demand more calculations than Lloyd's algorithm, the extended-Hartigan method consistently proves to be the most efficient, with a cost profile closest to the baseline. The mixed modes occupy a middle position, requiring a moderate increase in calculations to achieve the superior clustering results, as previously shown.

Furthermore, the fact that `extended-hartigan` requires, on average, fewer norm calculations than `mixed-extended-hartigan` suggests that the former scheme alone is more efficient to find a favorable path to a lower cost. This might be because `extended-hartigan` requires an initial "warmup" period after starting from Lloyd's converged configuration (recall that

`mixed-extended-hartigan` runs `lloyd` up to convergence, and then proceeds with the extended scheme). This could explain the additional norm calculations needed to continue minimization.

4.3.5 Robustness of Extended Hartigan

Having a closer look at our proposed `extended-hartigan`, we can notice that the vast majority of the total iterations are "unsafe", while the "safe" iterations are infrequent, short-lived, and usually tend to be near the end, as shown in Group 4.1. Therefore, for the majority of its runtime, the algorithm successfully uses an aggressive "batch update" mode where multiple candidate moves are accepted simultaneously, leading to rapid convergence.

The absence of unsafe iterations in the first part of the runs means the batch optimization works well, and accepting a large list of cost-reducing moves simultaneously consistently results in a better overall solution within a few iterations. In later phases, then, as the algorithm approaches a local minimum, the clustering becomes more stable, and the risk that a set of individually promising moves will have negative interactions when performed together increases. In this context, safe updates seem to allow the algorithm to reach a favorable configuration again, and to perform some other unsafe iteration before converging.

This trend is also confirmed on the more complicated datasets, where, however, the algorithm tends to perform safe iterations more frequently. Even if some safe iterations are present also in earlier stages, the majority of them happen again closer to convergence. Looking at *Bridge-100* and is datasets in Appendix A.5, in particular, one can notice how safe updates are spread throughout the algorithm iterations, with an evident increase towards the end of the runs. Still, safe iterations are isolated – there are sometimes just two or three of them in a row – and seem to have a key role in adjusting the current configuration in such a way that unsafe iterations can continue their updates.

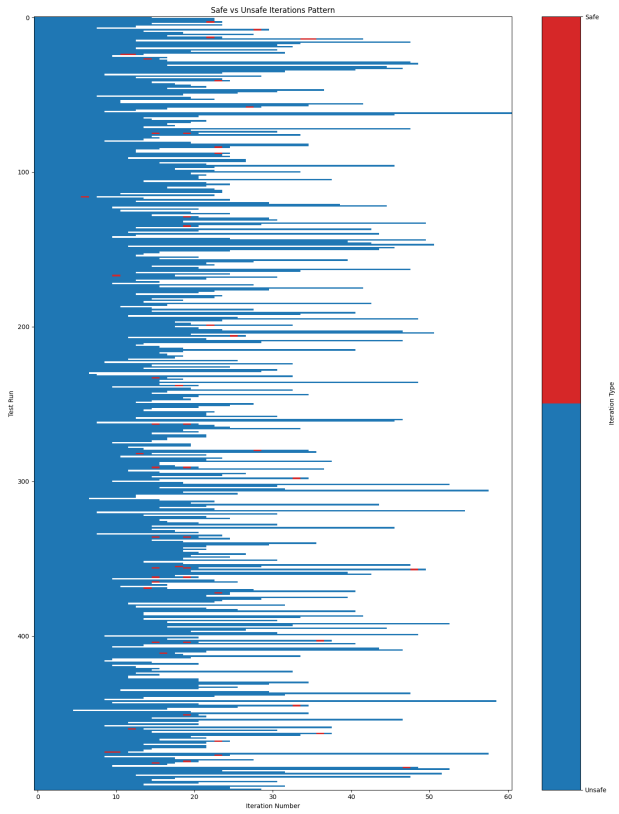

Figure 4.7: Record of unsafe (blue) and safe (red) iterations in time.

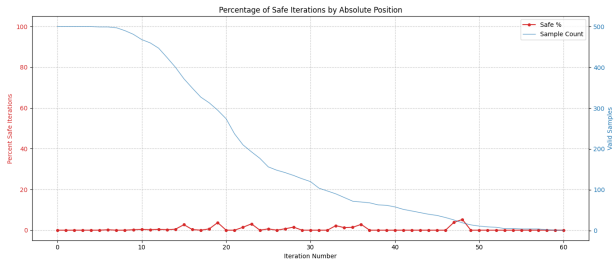

Figure 4.8: Percentage of safe iterations by absolute position. In blue the runs which are still active (i.e., have not converged yet) at such iteration.

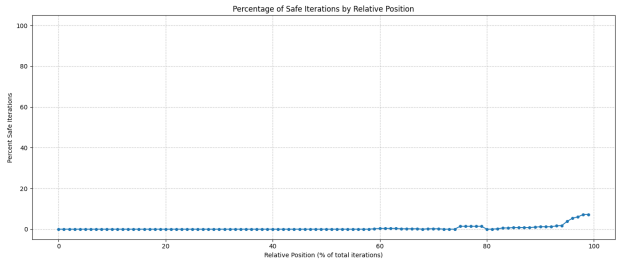

Figure 4.9: Percentage of safe iterations by relative position.

Group 4.1: Results of 500 tests of `extended-hartigan` with `maximin` initialization on $A3$ dataset.

Chapter 5

Conclusion

This work has explored the performance of several k-means clustering algorithms, with a particular focus on a novel approach, the extended-Hartigan algorithm, and some hybrid variations, namely mixed-Hartigan and mixed-extended-Hartigan. In particular, the "extended" approach consisted of allowing the classical Hartigan's scheme to accept multiple cost-reducing candidates in each iteration, instead of just single candidates, hoping to reduce the overall clustering cost more effectively.

Through a series of simulations on various datasets, we have shown the strengths and weaknesses of each of the above clustering algorithms, coupled with different initialization methods. Our results indicate that the choice of initialization strategy is a critical factor in determining the final clustering quality, with the `greedy-k-means++` method consistently yielding the most accurate and stable results, albeit at a higher computational cost, as already found by several researchers.

As for the proposed algorithm, they all have shown considerable promise. They consistently achieve final clusterings that are comparable, or superior, to both the classic Lloyd's and Hartigan's algorithms, particularly on more complex datasets such as *Bridge-100)* and *House*. On these datasets, our proposed methods achieved a cost reduction of nearly 3% relative to Lloyd's algorithm. Furthermore, in best-case scenarios, the extended-Hartigan algorithm demonstrated the highest potential by identifying the optimal clustering solution on the

most challenging problems.

A significant advantage of the extended-Hartigan algorithm is its computational efficiency and stability compared to the standard Hartigan algorithm. While the classic Hartigan method showed extreme inefficiency on the *Bridge-5* dataset, our proposed variations successfully avoided this computational blow-up, with a cost profile closest to Lloyd's baseline. This efficiency is largely attributed to its "unsafe" batch update mode. The less frequent "safe" iterations appear to play a crucial role in refining the clustering as it approaches a local minimum.

While this study provides strong empirical evidence for the effectiveness of `mixed` modes and of the `extended-hartigan` algorithm, further research could concentrate on a formal theoretical analysis of the convergence properties and performance guarantees of the latter. Moreover, the `extended-hartigan` scheme's design may allow for the application of deterministic acceleration methods, similar to those proposed for Lloyd's algorithm by Elkan [Elk03], Newling and Fleuret [NF16] or Xia et al. [Xia+22]. These optimizations are difficult to apply to the standard Hartigan's method, as it requires recalculating centroids after each individual point is moved. The extended version, however, could potentially incorporate these techniques. This could allow further enhancing the computational efficiency already demonstrated, while also preserving the advantages in the final clustering cost that this work has shown.

Appendix A

Tables and Figures

A.1 Tables of Results

In this section, we will include some tables summarizing all the results found through our simulations. Each row contains data obtained using a certain algorithm and initialization method, as follows:

Algorithms:

- `lloyd`: Lloyd's standard algorithm
- `hart`: Hartigan's standard algorithm
- `x-hart`: extended-Hartigan
- `x-hart`: mixed-Hartigan
- `m-x-hart`: mixed-extended-Hartigan

Initialization methods:

- `mm`: maximin initialization
- `k++`: k-means++ initialization
- `g-km++`: greedy-k-means++ initialization

The metrics contained in the tables are the following:

- min, max, mean, std CI: data relative to the Centroid Index (only available for *A-Sets*).
- min, max, mean, std cost: data relative to the final WCSS cost.
- safe, unsafe, lloyd, hart iter: data relative to the mean number of safe and unsafe (for extended modes), lloyd (for Lloyd and mixed modes), hartigan (for classical Hartigan and mixed-Hartigan modes) iterations, and their mean total number.
- init norms, norms, tot norms: the mean number of norm calculations performed during the initialization phase, the algorithm body, and their total number, respectively.

Algo	Init	min_CI	max_CI	mean_CI	std_CI
lloyd	mm	0	3	1.014000	0.577758
	k++	0	4	1.532000	0.764837
	g-km++	0	2	0.628000	0.579324
hart	mm	0	3	1.051000	0.595314
	k++	0	4	1.510000	0.768049
	g-km++	0	2	0.639000	0.569806
x-hart	mm	0	3	1.005000	0.548612
	k++	0	4	**1.455000**	0.752313
	g-km++	0	2	0.646000	0.587098
m-hart	mm	0	2	1.019000	0.544646
	k++	0	4	1.507000	0.736173
	g-km++	0	2	0.615000	0.559263
m-x-hart	mm	0	3	**0.999000**	0.566568
	k++	0	4	1.514000	0.750869
	g-km++	0	2	**0.602000**	0.574105

Table A.1: $A1$ CI results

Algo	Init	min_CI	max_CI	mean_CI	std_CI
lloyd	mm	0	5	2.597000	0.931982
	k++	0	6	2.905000	1.034396
	g-km++	0	3	1.147000	0.699565
hart	mm	0	5	2.716000	0.954643
	k++	0	6	2.885000	1.044880
	g-km++	0	4	1.157000	0.707355
x-hart	mm	0	5	2.613000	0.935538
	k++	0	6	2.925000	1.005671
	g-km++	0	4	1.152000	0.689127
m-hart	mm	0	5	**2.596000**	0.956443
	k++	0	6	2.889000	0.995329
	g-km++	0	4	**1.135000**	0.686130
m-x-hart	mm	0	5	2.667000	0.995043
	k++	0	6	**2.847000**	1.006773
	g-km++	0	3	1.161000	0.681967

Table A.2: $A2$ CI results

Algo	Init	min_CI	max_CI	mean_CI	std_CI
lloyd	mm	0	6	2.847000	1.016657
	k++	1	8	4.123000	1.141872
	g-km++	0	4	1.653000	0.810303
hart	mm	1	6	2.938000	0.998076
	k++	1	9	4.171000	1.183959
	g-km++	0	5	1.646000	0.810360
x-hart	mm	0	6	2.915000	0.994874
	k++	1	8	**4.076000**	1.152486
	g-km++	0	4	1.648000	0.811231
m-hart	mm	0	6	**2.823000**	0.990793
	k++	1	8	4.213000	1.198178
	g-km++	0	4	1.623000	0.782861
m-x-hart	mm	0	6	2.879000	0.955175
	k++	1	8	4.110000	1.188234
	g-km++	0	4	**1.589000**	0.753710

Table A.3: $A3$ CI results

Algo	Init	min_cost	mean_cost	max_cost	std_cost	safe_iter	unsafe_iter	lloyd_iter	hart_iter	tot_iter	init_norms	norms	tot_norms
lloyd	mm	12146257522.258896	14841946964.405014	21841814398.116207	1657080428.488605	0.000000	0.000000	18.937000	0.000000	18.937000	10.500000	18.937000	29.437000
	k++	12146257522.258896	16417649238.353601	25163192245.000164	2316354400.425482	0.000000	0.000000	16.501000	0.000000	16.501000	10.500000	16.501000	27.001000
	g-km++	12146257522.258896	13717924072.308132	18879378855.630257	1499700904.516583	0.000000	0.000000	10.180000	0.000000	10.180000	90.300000	10.180000	100.480000
hart	mm	12146257522.258894	14916636982.657883	21606378507.168976	1701562833.030088	0.000000	0.000000	0.000000	11.670000	11.670000	10.500000	45.295100	55.795100
	k++	12146257522.258896	16295308066.882206	26169761656.456001	2317829227.628590	0.000000	0.000000	0.000000	10.755000	10.755000	10.500000	40.277750	50.777750
	g-km++	12146257522.258894	13742493183.545908	19084660192.080318	1482971666.062825	0.000000	0.000000	0.000000	7.196000	7.196000	90.300000	19.959400	110.259400
x-hart	mm	12146257522.258894	14767844303.212078	21606075084.302711	1557628467.297744	0.020000	20.218000	0.000000	0.000000	20.238000	10.500000	22.161800	32.661800
	k++	12146257522.258896	16162358519.179180	25306232141.392090	2277324643.338295	0.062000	19.278000	0.000000	0.000000	19.340000	10.500000	21.174000	31.674000
	g-km++	12146257522.258894	13737587933.751692	19084323233.644440	1519376173.215931	0.034000	12.127000	0.000000	0.000000	12.161000	90.300000	13.277100	103.577100
m-hart	mm	12146257522.258894	14805167169.490540	19033383097.766228	1518771315.064257	0.000000	0.000000	19.858000	4.881000	24.739000	10.500000	26.254050	36.754050
	k++	12146257522.258896	16291873344.416574	25314836827.733368	2223268813.698408	0.000000	0.000000	15.991000	5.187000	21.178000	10.500000	22.755850	33.255850
	g-km++	12146257522.258894	13642797084.942059	18774063956.371967	1416031023.781070	0.000000	0.000000	9.793000	3.523000	13.316000	90.300000	14.066800	104.366800
m-x-hart	mm	12146257522.258896	14725718470.129992	21396065976.336056	1583311718.438927	0.013000	7.379000	18.912000	0.000000	26.304000	10.500000	26.943200	37.443200
	k++	12146257522.258896	16364523761.424688	25225931596.925411	2283541398.466086	0.039000	8.583000	16.260000	0.000000	24.882000	10.500000	25.644200	36.144200
	g-km++	12146257522.258894	13627148564.933023	18136304360.669247	1472637047.965268	0.023000	4.996000	9.466000	0.000000	14.485000	90.300000	14.886900	105.186900

Table A.4: A1 metrics

Algo	Init	min_cost	mean_cost	max_cost	std_cost	safe_iter	unsafe_iter	lloyd_iter	hart_iter	tot_iter	init_norms	norms	tot_norms
lloyd	mm	20286736641.652187	26843756945.622459	35414539297.908661	2464843591.951113	0.000000	0.000000	17.186000	0.000000	17.186000	18.000000	17.186000	35.186000
	k++	20286817823.751282	27966703202.759396	38493380306.013077	3004942091.734379	0.000000	0.000000	19.434000	0.000000	19.434000	18.000000	19.434000	37.434000
	g-km++	20286736641.652184	22953883768.901768	30052880407.236160	1695268897.218208	0.000000	0.000000	12.169000	0.000000	12.169000	192.857143	12.169000	205.026143
hart	mm	20286736641.652180	27208756265.974281	35410827796.578789	2544490310.177399	0.000000	0.000000	0.000000	12.117000	12.117000	18.000000	43.416229	61.416229
	k++	20286736641.652187	27984463659.314816	38546834166.080139	3178070475.933357	0.000000	0.000000	0.000000	11.663000	11.663000	18.000000	40.924771	58.924771
	g-km++	20286736641.652180	22935564915.098049	30471804871.576130	1682961032.380597	0.000000	0.000000	0.000000	7.639000	7.639000	192.857143	18.830857	211.688000
x-hart	mm	20286736641.652184	26858530523.048119	33849993499.323875	2442177550.299162	0.070000	21.204000	0.000000	0.000000	21.274000	18.000000	22.432514	40.432514
	k++	20286736641.652184	28013447199.545464	38249025036.744049	2983727665.410038	0.080000	20.955000	0.000000	0.000000	21.035000	18.000000	22.179856	40.179856
	g-km++	20286736641.652180	22953918313.358009	28896676854.014309	1641912124.019049	0.049000	13.700000	0.000000	0.000000	13.749000	192.857143	14.477514	207.334657
m-hart	mm	20286736641.652180	26809230626.549976	33761932752.446331	2507135288.406207	0.000000	0.000000	17.053000	6.553000	23.606000	18.000000	25.246486	43.246486
	k++	20286736641.652184	27848903584.945461	37739379645.149033	2930632879.648864	0.000000	0.000000	19.220000	4.932000	24.152000	18.000000	25.104657	43.104657
	g-km++	20286736641.652176	22934135733.766678	29690994301.385555	1666741136.462880	0.000000	0.000000	12.472000	3.271000	15.743000	192.857143	16.122743	208.979886
m-x-hart	mm	20286736641.652187	26984760706.507454	35410827796.578789	2569203688.894794	0.055000	12.189000	17.597000	0.000000	29.841000	18.000000	30.483514	48.483514
	k++	20286736641.652180	27781746889.950504	38255746074.679085	2972370318.014033	0.079000	8.204000	19.155000	0.000000	27.438000	18.000000	27.854171	45.854171
	g-km++	20286736641.652180	22976274917.518543	28259264467.185600	1653583777.626450	0.045000	4.823000	12.157000	0.000000	17.025000	192.857143	17.246029	210.103171

Table A.5: A2 metrics

Algo	Init	min_cost	mean_cost	max_cost	std_cost	safe_iter	unsafe_iter	lloyd_iter	hart_iter	tot_iter	init_norms	norms	tot_norms
lloyd	mm	28937496281.788746	36426122703.087250	44958428134.805336	2766762954.021780	0.000000	0.000000	18.812000	0.000000	18.812000	25.500000	18.812000	44.312000
	k++	31170694404.507622	40343968440.816238	52348444852.033607	3539993413.406665	0.000000	0.000000	20.722000	0.000000	20.722000	25.500000	20.722000	46.222000
	g-km++	28937415099.689640	33056672746.663124	42631132598.293625	2162500893.605343	0.000000	0.000000	13.547000	0.000000	13.547000	275.400000	13.547000	288.947000
hart	mm	30842267752.205608	36580778650.622635	44840878736.617378	2747831952.926758	0.000000	0.000000	0.000000	13.483000	13.483000	25.500000	44.873380	70.373380
	k++	30790373477.959969	40376336404.443153	55292237633.228104	3611432919.651457	0.000000	0.000000	0.000000	12.434000	12.434000	25.500000	40.125080	65.625080
	g-km++	28937415099.689632	32999554530.319359	41824106626.861649	2096077457.154475	0.000000	0.000000	0.000000	8.478000	8.478000	275.400000	19.216280	294.616280
x-hart	mm	28937415099.689644	36542665544.294624	46433031716.294762	2717278628.369716	0.143000	23.795000	0.000000	0.000000	23.938000	25.500000	24.855520	50.355520
	k++	31434025633.737648	40120764758.159256	55799096113.123489	3487587439.259630	0.106000	22.039000	0.000000	0.000000	22.145000	25.500000	22.990800	48.490800
	g-km++	28937415099.689640	33001895126.986610	41170612420.495880	2105593846.944352	0.050000	14.939000	0.000000	0.000000	14.989000	275.400000	15.548560	290.948560
m-hart	mm	28937415099.689644	36233105622.741081	45698009900.369308	2685908733.160125	0.000000	0.000000	19.495000	6.714000	26.209000	25.500000	27.231280	52.731280
	k++	30845671827.411148	40572576778.573875	53289394784.770447	3643333576.555728	0.000000	0.000000	20.236000	5.956000	26.192000	25.500000	27.235820	52.735820
	g-km++	28937415099.689640	32938322322.102299	40813875962.328598	2039069477.004125	0.000000	0.000000	13.180000	3.977000	17.157000	275.400000	17.545840	292.945840
m-x-hart	mm	28937415099.689648	36421253928.252731	45698009900.369308	2588532431.541341	0.155000	11.997000	18.995000	0.000000	31.147000	25.500000	31.593080	57.093080
	k++	31220300249.261681	40240447164.763603	52688369187.118584	3579905003.876310	0.123000	9.736000	20.686000	0.000000	30.545000	25.500000	30.899360	56.399360
	g-km++	28937415099.689640	32846295431.091446	40100603512.120880	1937201295.141368	0.063000	6.015000	13.355000	0.000000	19.433000	275.400000	19.636120	295.036120

Table A.6: A3 metrics

Algo	Init	min_cost	mean_cost	max_cost	std_cost	safe_iter	unsafe_iter	lloyd_iter	hart_iter	tot_iter	init_norms	norms	tot_norms
lloyd	mm	37450555.033932	37453268.670207	37459577.740492	4023.292673	0.000000	0.000000	35.875000	0.000000	35.875000	3.000000	35.875000	38.875000
	k++	37450555.033932	37454942.018448	37459577.740492	4395.479542	0.000000	0.000000	32.809000	0.000000	32.809000	3.000000	32.809000	35.809000
	g-km++	37450555.033932	37454799.548028	37459577.740492	4389.563704	0.000000	0.000000	30.857000	0.000000	30.857000	17.400000	30.857000	48.257000
hart	mm	37450552.016165	37450697.770185	37451188.496601	267.442233	0.000000	0.000000	0.000000	21.956000	21.956000	3.000000	738.036800	741.036800
	k++	37450552.016165	37450849.252528	37451188.496601	317.546334	0.000000	0.000000	0.000000	20.302000	20.302000	3.000000	491.921200	494.921200
	g-km++	37450552.016165	37450815.592782	37451262.213444	313.593815	0.000000	0.000000	0.000000	18.693000	18.693000	17.400000	371.572400	388.972400
x-hart	mm	37450552.016165	37450749.961580	37451188.496601	294.628574	0.000000	39.520000	0.000000	0.000000	39.520000	3.000000	54.928000	57.928000
	k++	37450552.016165	37450895.715600	37451188.496601	317.220215	0.000000	38.847000	0.000000	0.000000	38.847000	3.000000	53.985800	56.985800
	g-km++	37450552.016165	37450858.799735	37451188.496601	318.033931	0.002000	34.914000	0.000000	0.000000	34.916000	17.400000	48.482400	65.882400
m-hart	mm	37450552.016165	37450736.595491	37451188.496601	288.810669	0.000000	0.000000	36.318000	5.424000	41.742000	3.000000	48.219600	51.219600
	k++	37450552.016165	37450874.711746	37451188.496601	318.209029	0.000000	0.000000	32.881000	7.355000	40.236000	3.000000	51.065600	54.065600
	g-km++	37450552.016165	37450851.161970	37451188.496601	317.666869	0.000000	0.000000	30.026000	6.845000	36.871000	17.400000	46.600200	64.000200
m-x-hart	mm	37450552.016165	37450746.142698	37451188.496601	293.040321	0.000000	9.115000	36.551000	0.000000	45.666000	3.000000	48.912000	51.912000
	k++	37450552.016165	37450882.349511	37451188.496601	318.010365	0.000000	12.629000	32.499000	0.000000	45.128000	3.000000	49.779600	52.779600
	g-km++	37450552.016165	37450867.710461	37451188.496601	318.230034	0.000000	11.781000	29.931000	0.000000	41.712000	17.400000	46.024400	63.424400

Table A.7: Bridge ($k = 5$) metrics

Algo	Init	min_cost	mean_cost	max_cost	std_cost	safe_iter	unsafe_iter	lloyd_iter	hart_iter	tot_iter	init_norms	norms	tot_norms
lloyd	mm	29225968.220905	29531987.934780	30178156.723077	224223.172988	0.000000	0.000000	77.787000	0.000000	77.787000	5.500000	77.787000	83.287000
	k++	29225517.831373	29423289.949534	30510623.159897	201326.256425	0.000000	0.000000	59.219000	0.000000	59.219000	5.500000	59.219000	64.719000
	g-km++	29225342.919201	29411300.847063	30051638.981498	174425.267391	0.000000	0.000000	53.947000	0.000000	53.947000	45.100000	53.947000	99.047000
hart	mm	29225354.031010	29466744.544103	30508125.546515	254432.618091	0.000000	0.000000	0.000000	43.054000	43.054000	5.500000	571.163400	576.663400
	k++	29224830.737353	29377827.453603	30031714.582518	166925.584188	0.000000	0.000000	0.000000	32.019000	32.019000	5.500000	381.379300	386.879300
	g-km++	29224840.566727	29372348.494121	30080292.338818	153329.252328	0.000000	0.000000	0.000000	28.767000	28.767000	45.100000	301.849500	346.949500
x-hart	mm	29225272.132593	29447786.726762	30036350.440556	207296.960477	0.117000	79.450000	0.000000	0.000000	79.567000	5.500000	95.280400	100.780400
	k++	29224830.737353	29384879.138948	30441145.688745	167009.198763	0.146000	58.318000	0.000000	0.000000	58.464000	5.500000	69.956799	75.456799
	g-km++	29224840.566727	29365669.732266	30033309.678663	152888.976124	0.130000	55.570000	0.000000	0.000000	55.700000	45.100000	66.640000	111.740000
m-hart	mm	29225210.340865	29455132.684482	30464815.930057	216009.374226	0.000000	0.000000	79.689000	14.009000	93.698000	5.500000	121.572500	127.072500
	k++	29224840.566727	29389121.843439	30441145.688745	177458.779153	0.000000	0.000000	58.789000	10.377000	69.166000	5.500000	85.155300	90.655300
	g-km++	29224830.737353	29374072.306429	30026114.830243	158965.935218	0.000000	0.000000	54.523000	10.068000	64.591000	45.100000	78.191800	123.291800
m-x-hart	mm	29225177.835557	29473611.018293	30041569.109010	210878.533066	0.191000	24.289000	77.214000	0.000000	101.694000	5.500000	106.390000	111.890000
	k++	29224840.566727	29398112.192672	30043400.619610	176647.380965	0.227000	17.856000	57.681000	0.000000	75.764000	5.500000	79.180600	84.680600
	g-km++	29224840.566727	29382598.550468	30026196.600088	162501.262528	0.195000	17.481000	53.850000	0.000000	71.526000	45.100000	74.861200	119.961200

Table A.8: Bridge ($k = 10$) metrics

Algo	Init	min_cost	mean_cost	max_cost	std_cost	safe_iter	unsafe_iter	lloyd_iter	hart_iter	tot_iter	init_norms	norms	tot_norms
lloyd	mm	15169044.143402	15413526.054082	15686285.117460	72752.984775	0.000000	0.000000	36.112000	0.000000	36.112000	50.500000	36.112000	86.612000
	k++	15006885.207483	15261330.634641	15637575.306265	93823.705600	0.000000	0.000000	31.943000	0.000000	31.943000	50.500000	31.943000	82.443000
	g-km++	14888158.119967	15080490.512997	15270653.332074	62057.236196	0.000000	0.000000	29.249000	0.000000	29.249000	650.440000	29.249000	679.689000
hart	mm	14790342.159888	14950466.975632	15160982.251727	56336.430898	0.000000	0.000000	0.000000	28.417000	28.417000	50.500000	74.412580	124.912580
	k++	14715851.007302	14866516.266135	15142030.234913	61845.890402	0.000000	0.000000	0.000000	26.137000	26.137000	50.500000	65.579562	116.079562
	g-km++	14688170.172373	14804686.358514	14965883.122493	44555.312414	0.000000	0.000000	0.000000	25.378000	25.378000	650.440000	57.090485	707.530485
x-hart	mm	14734885.809208	14909871.736426	15130673.611199	54121.611623	2.492000	54.952000	0.000000	0.000000	57.444000	50.500000	58.550307	109.050307
	k++	14720083.494351	14863172.313783	15136376.454066	67804.969647	2.494000	52.163000	0.000000	0.000000	54.657000	50.500000	55.728810	106.228810
	g-km++	14660559.967231	14795875.915432	14954653.496460	42421.681991	2.619000	51.183000	0.000000	0.000000	53.802000	650.440000	54.857492	705.297492
m-hart	mm	14790186.811389	14971580.971431	15171001.384794	56890.040181	0.000000	0.000000	36.650000	24.421000	61.071000	50.500000	74.423139	124.923139
	k++	14703697.924712	14880210.738334	15180616.531359	65683.335500	0.000000	0.000000	31.626000	23.436000	55.062000	50.500000	68.160188	118.660188
	g-km++	14702268.272609	14808325.435188	15014075.925214	43004.495138	0.000000	0.000000	29.079000	22.658000	51.737000	650.440000	63.221235	713.661235
m-x-hart	mm	14795743.705575	14952340.519967	15157306.798513	56440.474505	2.575000	49.395000	36.151000	0.000000	88.121000	50.500000	89.108015	139.608015
	k++	14708874.198344	14866330.303354	15134095.453951	64261.108705	2.999000	46.302000	32.068000	0.000000	81.369000	50.500000	82.333382	132.833382
	g-km++	14682048.526204	14800524.961708	14934658.985271	42875.163423	2.570000	44.779000	29.083000	0.000000	76.432000	650.440000	77.358345	727.798345

Table A.9: Bridge ($k = 100$) metrics

Algo	Init	min_cost	mean_cost	max_cost	std_cost	safe_iter	unsafe_iter	lloyd_iter	hart_iter	tot_iter	init_norms	norms	tot_norms
lloyd	mm	371278.303206	382715.960453	395631.437444	4118.852818	0.000000	0.000000	22.487000	0.000000	22.487000	25.500000	22.487000	47.987000
	k++	370417.853799	380862.655932	402226.464774	4453.177247	0.000000	0.000000	25.127000	0.000000	25.127000	25.500000	25.127000	50.627000
	g-km++	370071.315560	378008.734612	390312.404085	3180.683966	0.000000	0.000000	22.077000	0.000000	22.077000	275.400000	22.077000	297.477000
hart	mm	366520.755445	373468.045605	385246.417067	2978.593390	0.000000	0.000000	0.000000	18.052000	18.052000	25.500000	38.938130	64.438130
	k++	366595.165011	374296.035692	387971.630651	3389.335372	0.000000	0.000000	0.000000	19.178000	19.178000	25.500000	42.599092	68.099092
	g-km++	365974.942987	372578.933177	380593.031789	2535.719573	0.000000	0.000000	0.000000	17.326000	17.326000	275.400000	35.292574	310.692574
x-hart	mm	366674.505745	373218.323535	383225.645717	2731.395745	0.792000	33.885000	0.000000	0.000000	34.677000	25.500000	36.023144	61.523144
	k++	366107.320671	373996.104347	391138.410258	3458.082248	0.834000	36.615000	0.000000	0.000000	37.449000	25.500000	38.906857	64.406857
	g-km++	366018.516758	372496.688959	385531.234817	2525.460599	0.844000	32.349000	0.000000	0.000000	33.193000	275.400000	34.480703	309.880703
m-hart	mm	366040.947440	373758.809011	385127.585050	3060.351235	0.000000	0.000000	22.368000	15.137000	37.505000	25.500000	43.534909	69.034909
	k++	366966.502594	374540.623989	389857.578470	3482.283079	0.000000	0.000000	25.045000	14.510000	39.555000	25.500000	44.842536	70.342536
	g-km++	366783.307582	372657.840438	382595.083341	2470.383826	0.000000	0.000000	22.288000	13.692000	35.980000	275.400000	40.755402	316.155402
m-x-hart	mm	366235.226634	373613.335873	383482.149514	2972.416158	0.745000	27.854000	22.286000	0.000000	50.885000	25.500000	51.988532	77.488532
	k++	366374.321218	374437.582813	389514.862165	3534.399679	0.843000	27.686000	25.231000	0.000000	53.760000	25.500000	54.861075	80.361075
	g-km++	366277.255274	372420.173869	384458.617626	2571.090404	0.813000	25.683000	22.283000	0.000000	48.779000	275.400000	49.798828	325.198828

Table A.10: House metrics

A.2 Mean Final Cost

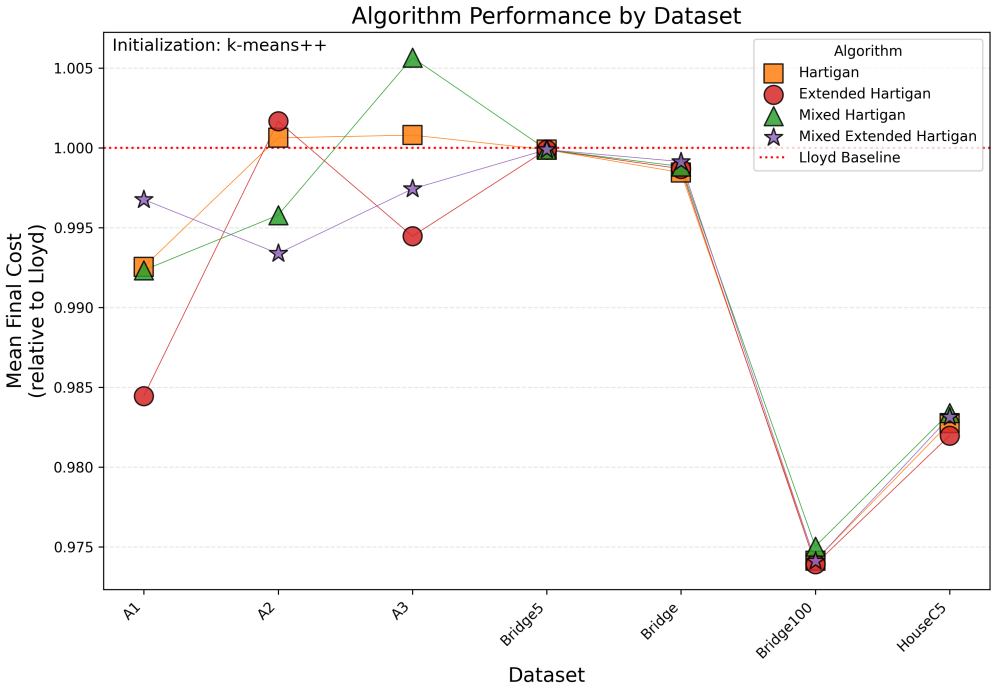

Figure A.1: Mean final cost using `k-means++` initialization.

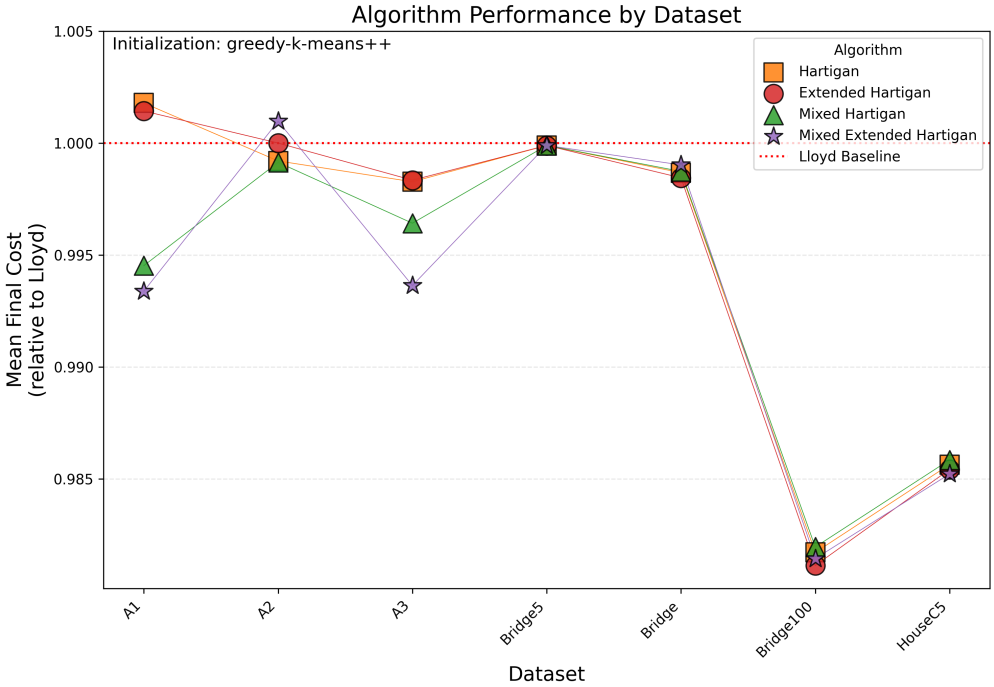

Figure A.2: Mean final cost using `greedy-k-means++` initialization.

A.3 Minimum Final Cost

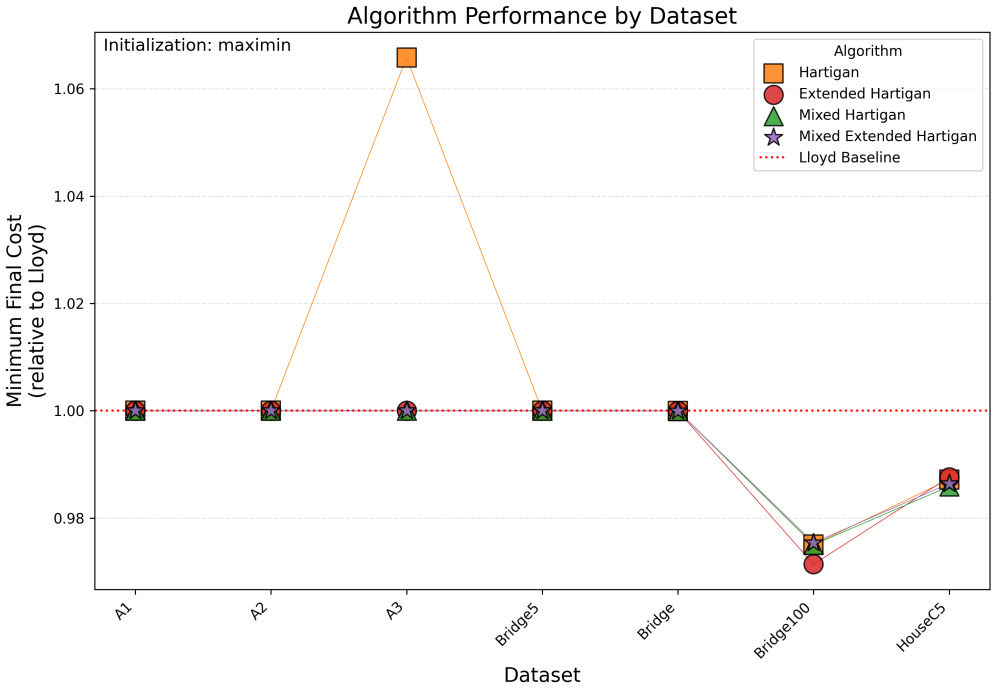

Figure A.3: Minimum final cost using `maximin` initialization.

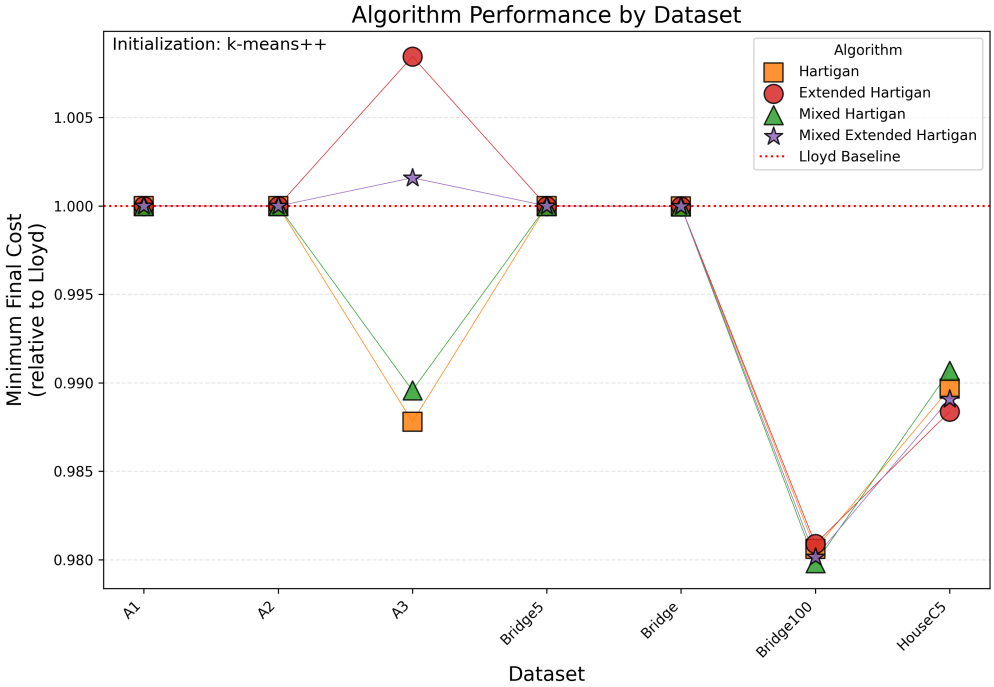

Figure A.4: Minimum final cost using `k-means++` initialization.

A.4 Mean Norm Calculations

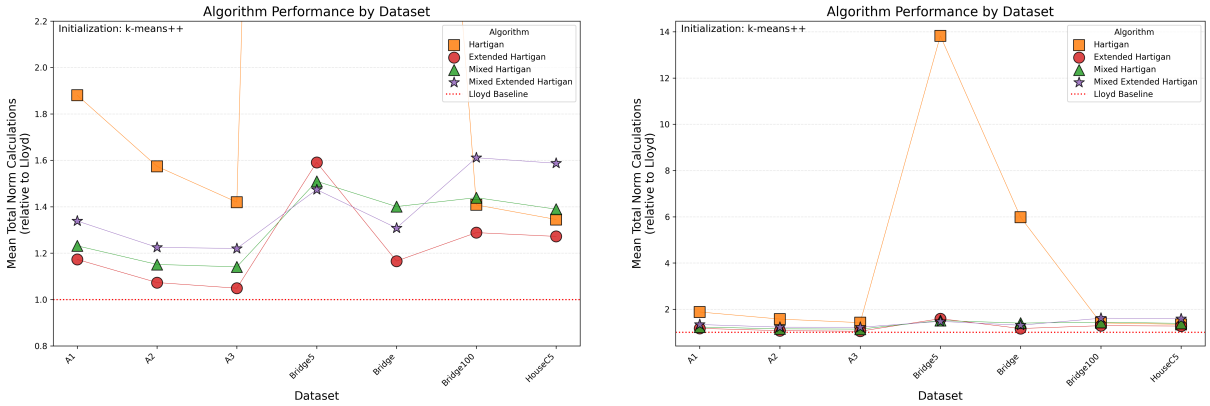

Figure A.5: Mean norm calculations using `k-means++` initialization.

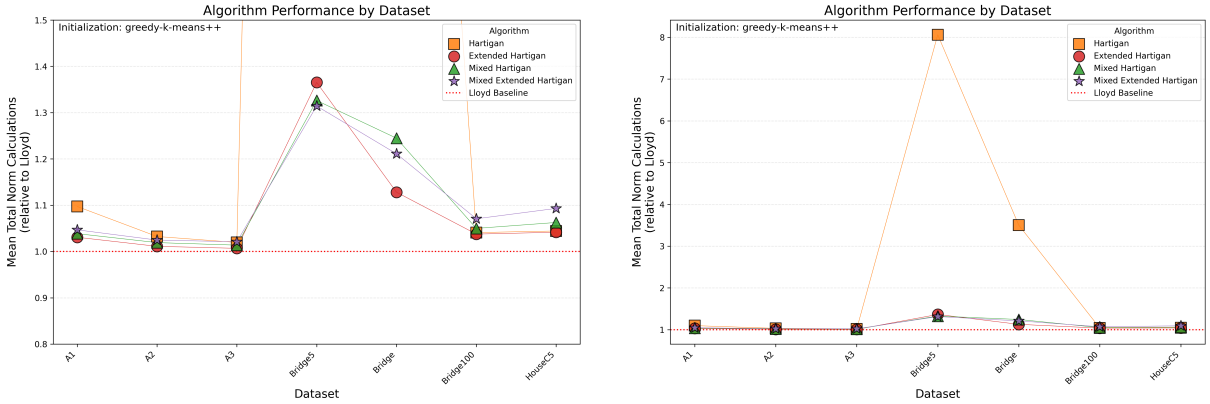

Figure A.6: Mean norm calculations using `greedy-k-means++` initialization.

A.5 Safe and Unsafe iterations in `extended-hartigan`

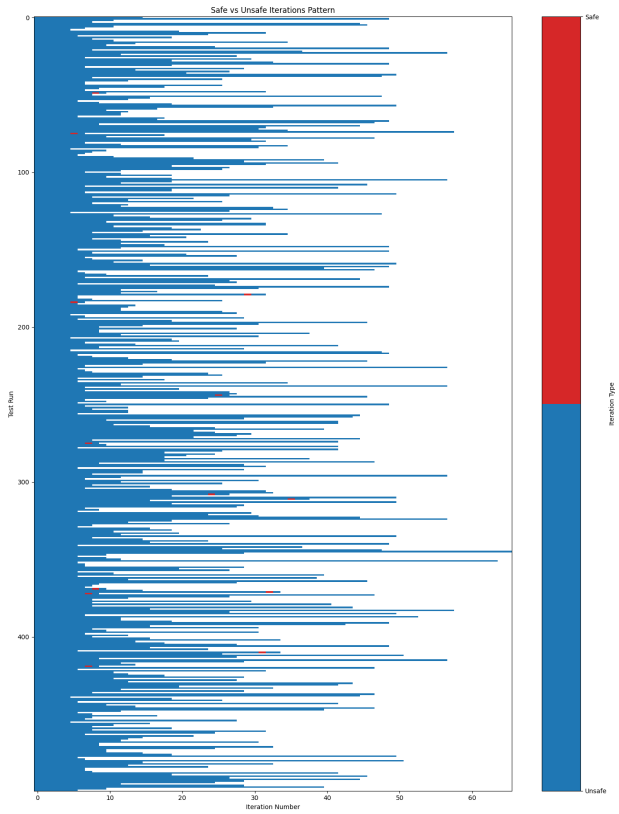

Figure A.7: Record of safe and unsafe iterations in time.

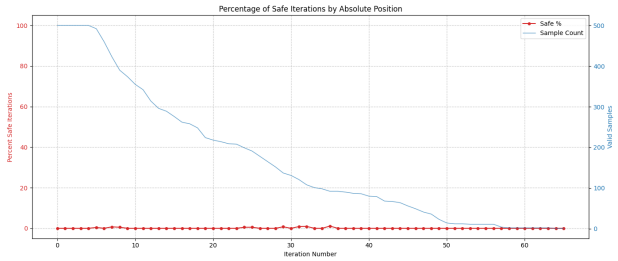

Figure A.8: Percentage of safe iterations by absolute position.

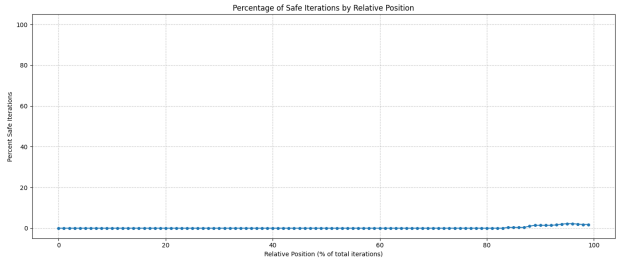

Figure A.9: Percentage of safe iterations by relative position.

Group A.1: Results of 500 tests of `extended-hartigan` with `maximin` initialization on $A1$ dataset.

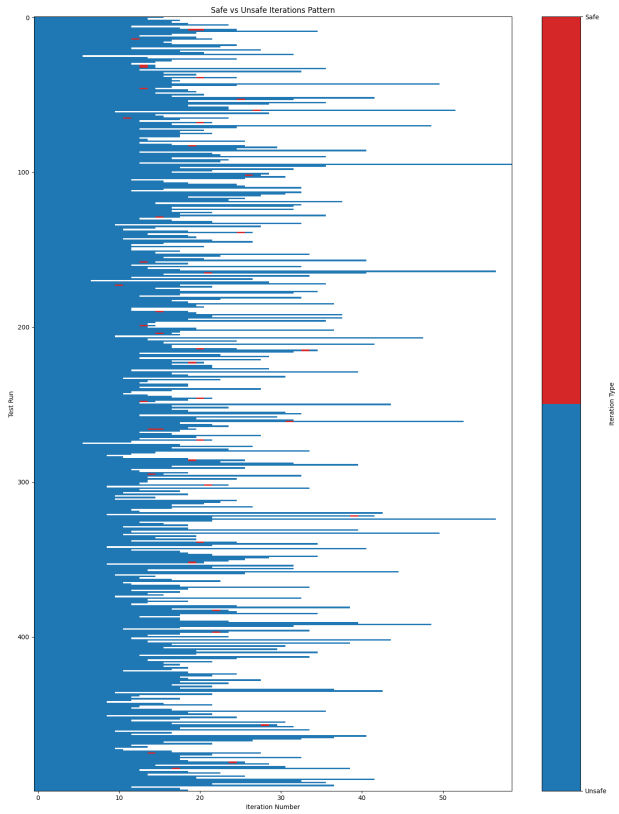

Figure A.10: Record of safe and unsafe iterations in time.

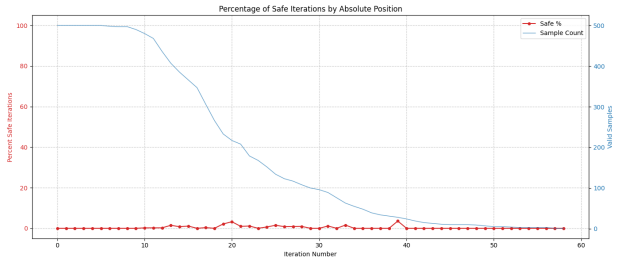

Figure A.11: Percentage of safe iterations by absolute position.

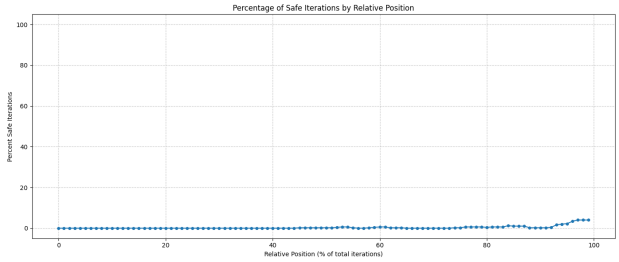

Figure A.12: Percentage of safe iterations by relative position.

Group A.2: Results of 500 tests of `extended-hartigan` with `maximin` initialization on $A2$ dataset.

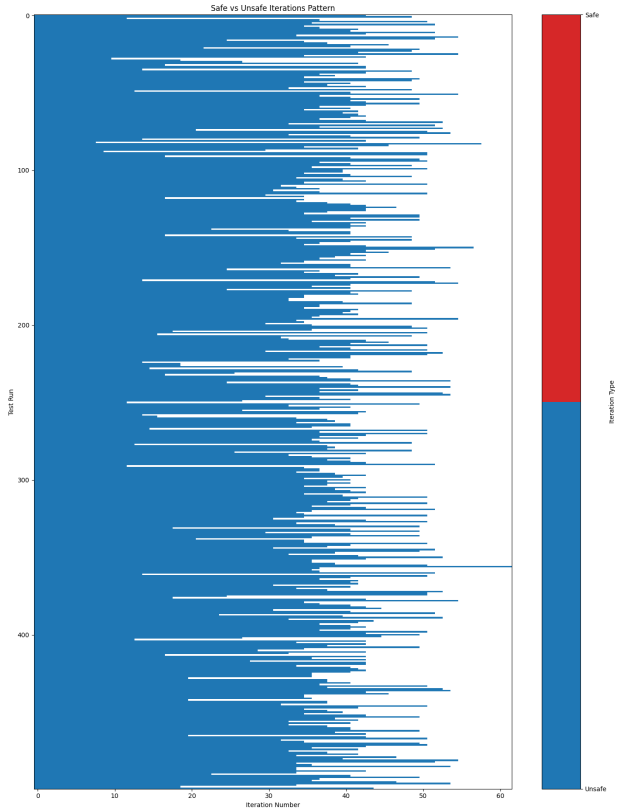

Figure A.13: Record of safe and unsafe iterations in time.

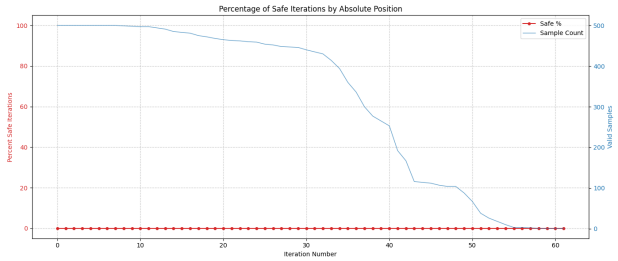

Figure A.14: Percentage of safe iterations by absolute position.

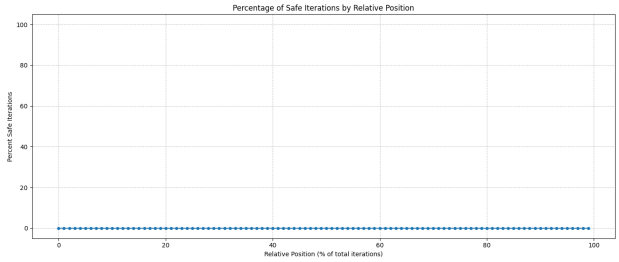

Figure A.15: Percentage of safe iterations by relative position.

Group A.3: Results of 500 tests of `extended-hartigan` with `maximin` initialization on *Bridge-5* dataset.

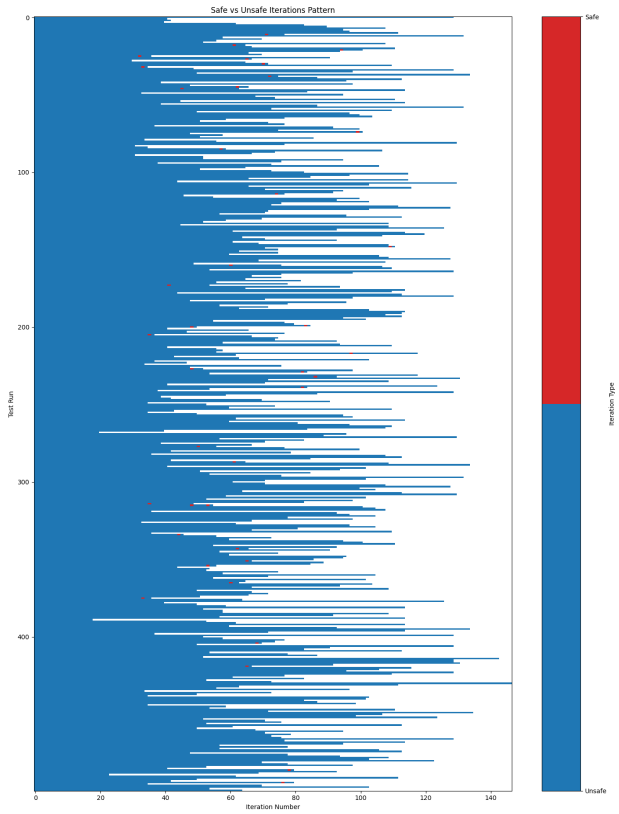

Figure A.16: Record of safe and unsafe iterations in time.

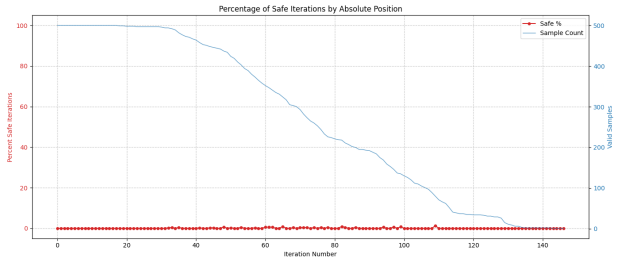

Figure A.17: Percentage of safe iterations by absolute position.

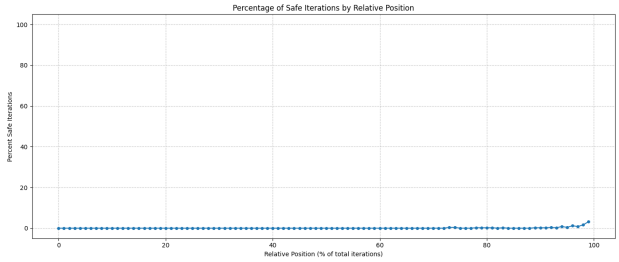

Figure A.18: Percentage of safe iterations by relative position.

Group A.4: Results of 500 tests of `extended-hartigan` with `maximin` initialization on *Bridge-10* dataset.

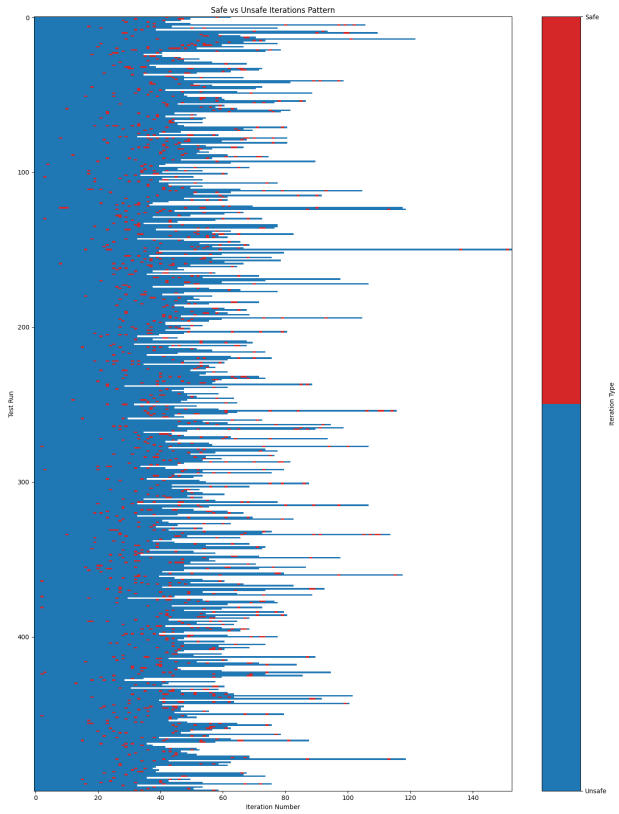

Figure A.19: Record of safe and unsafe iterations in time.

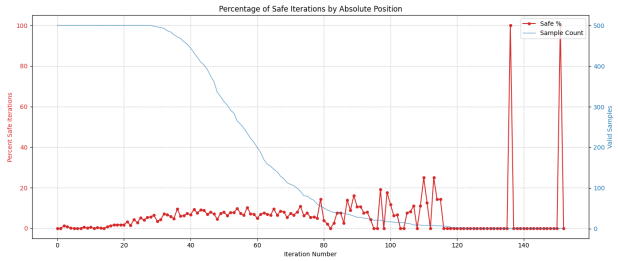

Figure A.20: Percentage of safe iterations by absolute position.

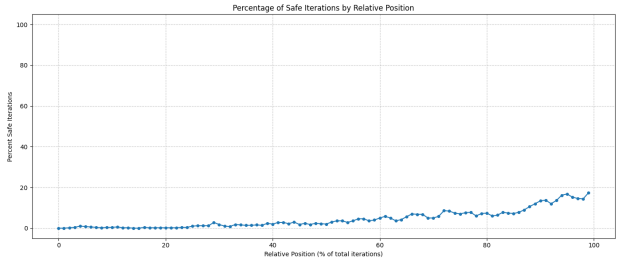

Figure A.21: Percentage of safe iterations by relative position.

Group A.5: Results of 500 tests of `extended-hartigan` with `maximin` initialization on *Bridge-100* dataset.

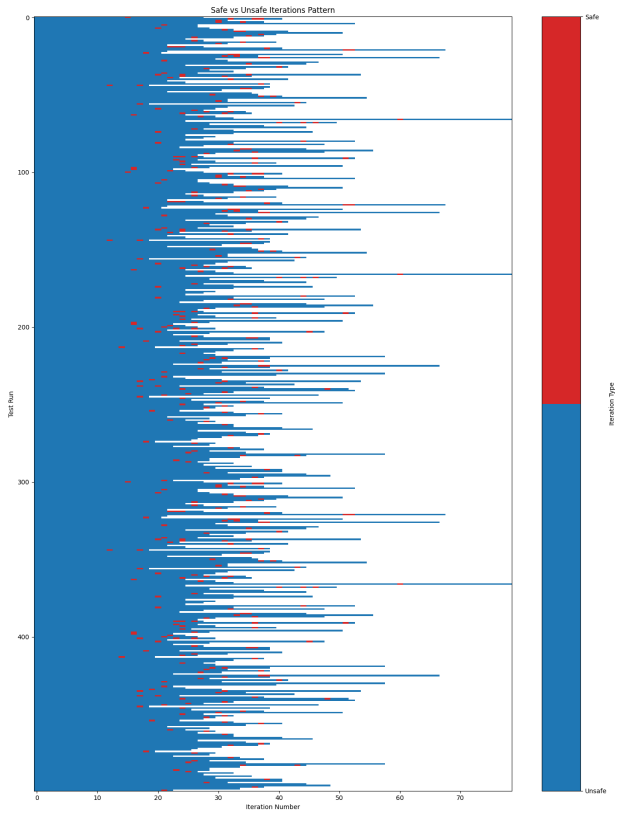

Figure A.22: Record of safe and unsafe iterations in time.

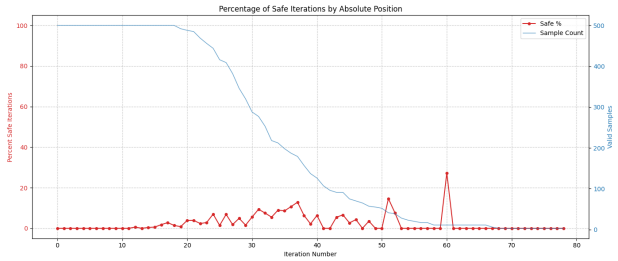

Figure A.23: Percentage of safe iterations by absolute position.

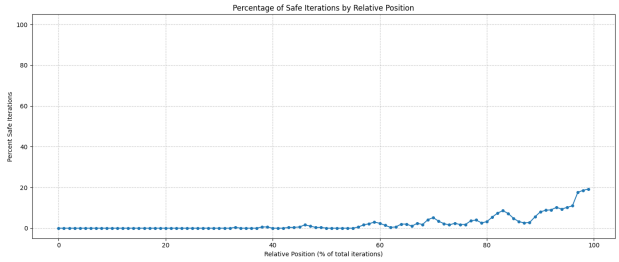

Figure A.24: Percentage of safe iterations by relative position.

Group A.6: Results of 500 tests of `extended-hartigan` with `maximin` initialization on *House* dataset.

Bibliography

[AV06] David Arthur and Sergei Vassilvitskii. "How slow is the k -means method?" en. In: *Proceedings of the twenty-second annual symposium on Computational geometry*. Sedona Arizona USA: ACM, June 2006, pp. 144–153. ISBN: 9781595933409. DOI: 10.1145/1137856.1137880. URL: https://dl.acm.org/doi/10.1145/1137856.1137880.

[AV07] David Arthur and Sergei Vassilvitskii. "K-Means++: The Advantages of Careful Seeding". In: vol. 8. Jan. 2007, pp. 1027–1035. DOI: 10.1145/1283383.1283494.

[Bha+19] Anup Bhattacharya et al. *Noisy, Greedy and Not So Greedy k-means++*. arXiv:1912.00653. Dec. 2019. DOI: 10.48550/arXiv.1912.00653. URL: http://arxiv.org/abs/1912.00653.

[Elk03] Charles Elkan. "Using the triangle inequality to accelerate k-means". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*. ICML'03. Washington, DC, USA: AAAI Press, 2003, pp. 147–153. ISBN: 1577351894.

[FRZ14] Pasi Fränti, Mohammad Rezaei, and Qinpei Zhao. "Centroid index: Cluster level similarity measure". en. In: *Pattern Recognition* 47.9 (Sept. 2014), pp. 3034–3045. ISSN: 00313203. DOI: 10.1016/j.patcog.2014.03.017. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320314001150.

[Gru+22] Christoph Grunau et al. *A Nearly Tight Analysis of Greedy k-means++*. arXiv:2207.07949. July 2022. DOI: 10.48550/arXiv.2207.07949. URL: http://arxiv.org/abs/2207.07949.

[Har75] John A. Hartigan. *Clustering algorithms*. eng. A Wiley publication in applied statistics. New York: Wiley, 1975. ISBN: 9780471356455.

[HW79] J. A. Hartigan and M. A. Wong. "Algorithm AS 136: A K-Means Clustering Algorithm". In: *Journal of the Royal Statistical Society. Series C (Applied Statistics)* 28.1 (1979), pp. 100–108. ISSN: 00359254, 14679876. URL: http://www.jstor.org/stable/2346830.

[KF02] Ismo Kärkkäinen and Pasi Fränti. *Dynamic local search algorithm for the clustering problem*. eng. Report series / University of Joensuu, Department of Computer Science. A, 2002-6. OCLC: 58380784. Joensuu: University of Joensuu, 2002. ISBN: 9789524581431.

[Llo82] S. Lloyd. "Least squares quantization in PCM". en. In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137. ISSN: 0018-9448. DOI: 10.1109/TIT.1982.1056489. URL: http://ieeexplore.ieee.org/document/1056489/.

[LZ16] Yu Lu and Harrison H. Zhou. *Statistical and Computational Guarantees of Lloyd's Algorithm and its Variants*. arXiv:1612.02099. Dec. 2016. DOI: 10.48550/arXiv.1612.02099. URL: http://arxiv.org/abs/1612.02099.

[NF16] James Newling and Francois Fleuret. "Fast k-means with accurate bounds". en. In: *Proceedings of The 33rd International Conference on Machine Learning*. PMLR, June 2016, pp. 936–944. URL: https://proceedings.mlr.press/v48/newling16.html.

[SB14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: from theory to algorithms*. eng. New York: Cambridge university press, 2014. ISBN: 9781107057135.

[Sci] Scikit-learn. *KMeans*. en. URL: https://scikit-learn/stable/modules/generated/sklearn.cluster.KMeans.html.

[TV10] Matus Telgarsky and Andrea Vattani. "Hartigan's Method: k-means Clustering without Voronoi". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, May 2010, pp. 820–827. URL: https://proceedings.mlr.press/v9/telgarsky10a.html.

[Wu+08] Xindong Wu et al. "Top 10 algorithms in data mining". en. In: *Knowledge and Information Systems* 14.1 (Jan. 2008), pp. 1–37. ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-007-0114-2. URL: http://link.springer.com/10.1007/s10115-007-0114-2.

[Xia+22] Shuyin Xia et al. "Ball kk-Means: Fast Adaptive Clustering With No Bounds". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.1 (Jan. 2022), pp. 87–99. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2020.3008694. URL: https://ieeexplore.ieee.org/document/9139397.