IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 13, 2022, accepted May 25, 2022, date of publication June 2, 2022, date of current version June 8, 2022.

*Digital Object Identifier 10.1109/ACCESS.2022.3179803*

# An Extensive Empirical Comparison of *k*-means Initialization Algorithms

**SIMON HARRIS AND RENATO CORDEIRO DE AMORIM**
School of Computer Science and Electronic Engineering, University of Essex, Colchester CO4 3SQ, U.K.

Corresponding author: Renato Cordeiro De Amorim (r.amorim@essex.ac.uk)

**ABSTRACT** The *k*-means clustering algorithm, whilst widely popular, is not without its drawbacks. In this paper, we focus on the sensitivity of *k*-means to its initial set of centroids. Since the cluster recovery performance of *k*-means can be improved by better initialisation, numerous algorithms have been proposed aiming at producing good initial centroids. However, it is still unclear which algorithm should be used in any particular clustering scenario. With this in mind, we compare 17 such algorithms on 6,000 synthetic and 28 real-world data sets. The synthetic data sets were produced under different configurations, allowing us to show which algorithm excels in each scenario. Hence, the results of our experiments can be particularly useful for those considering *k*-means for a non-trivial clustering scenario.

**INDEX TERMS** *k*-means, *k*-means initialisation, clustering.

## I. INTRODUCTION

The main goal of any clustering algorithm is to identify groups (i.e. clusters) of data points in such a way that those data points within the same group are similar, and those between groups are dissimilar. Clustering algorithms do not require labelled data points to learn from, and as such are valuable tools for practitioners in exploratory data analysis. Successful applications of these algorithms can be found in fields such as bioinformatics, computer vision, data mining, and many others [1]–[4].

Generally speaking, one can divide clustering algorithms into three main classes: partitional, hierarchical, and density-based. In their original form, partitional clustering algorithms identify $K$ clusters in a data set, so that each data point is assigned to exactly one cluster and no data point is left unassigned (creating a partition, in the mathematical sense). These algorithms have been extended using fuzzy logic so that a data point belongs to each of the $K$ clusters with different degrees of membership, usually adding to one [5]. Hierarchical algorithms identify a set of clusters as well as the tree-like relationship that exists between the clusters themselves. The usual approaches are top-down (divisive) and bottom-up (agglomerative) but even these can be further divided using the many ways one can calculate the distance between clusters (see for instance [6], and references therein). Density-based clustering algorithms define clusters as contiguous areas of higher density separated by contiguous areas of lower density. There are indeed a number of ways one can estimate the density of an area, or density thresholds (for details see [7], and references therein).

We are interested in a particular algorithm, *k*-means [8], [9]. This is believed to be the most popular partitional clustering algorithm in use today [10], [11]. It aims to partition a data set $X$ containing $N$ data points into $K$ homogeneous clusters $S = \{S_1, S_2, \ldots, S_K\}$. One can measure dissimilarity between $x_i, x_t \in X$ using the squared Euclidean distance

$$d(x_i, x_t) = \sum_{v=1}^{V} (x_{iv} - x_{tv})^2, \tag{1}$$

where $V$ is the number of features describing each $x_i, x_t \in X$. Each cluster $S_k \in S$ is represented by a centroid $c_k \in \mathbb{R}^V$, which is the component-wise mean over all $x_i \in S_k$. That is $c_{kv} = |S_k|^{-1} \sum_{x_i \in S_k} x_{iv}$ for $v = 1, 2, \ldots, V$. Thus, *k*-means minimises the criterion

$$W = \sum_{k=1}^{K} \sum_{x_i \in S_k} d(x_i, c_k). \tag{2}$$

The problem of finding minimum of (2) is NP-hard. if $K$ is given and $V = 2$ [12]. The *k*-means algorithm (Algorithm 1) tends to converge quickly, but it does so to a local optimum. That is, it generates a clustering that cannot be improved by changing a single point of cluster, as opposed to a global

The associate editor coordinating the review of this manuscript and approving it for publication was Qilian Liang.

optimum clustering—i.e. a clustering that cannot be improved by any change in cluster assignments [13].

Unfortunately, this is not the only weakness of *k*-means. Others include: (i) it requires $K$ to be known beforehand; (ii) the clustering is biased towards Gaussian clusters; (iii) it is sensitive to outliers (an outlier will always be assigned to a cluster); (iv) the final clustering is sensitive to the initial set of centroids. Implementations of *k*-means can be found in major software packages used for data analysis, including MATLAB, R, WEKA, and scikit-learn.

**Algorithm 1** The *k*-Means Algorithm

```
1) Select $K$ centroids $c_1, c_2, \ldots, c_K$, using a chosen initialisation strategy.
2) For each $x_i \in X$, calculate the distance between $x_i$ and each $c_k \in C$ using (1). Assign $x_i$ to the cluster $S_k$ represented by the nearest $c_k$.
3) Update each $c_k \in C$ to the component-wise mean over all $x_i \in S_k$.
4) If Step 3 produced changes to the centroids, go to Step 2. Otherwise, the algorithm has converged.
```

In this paper we are particularly interested in weakness (iv). In its original form, *k*-means identifies an initial set of centroids by first assigning each $x_i \in X$ to a cluster $S_k \in S$ chosen uniformly at random. Afterwards, each cluster produces a centroid—the component-wise mean over all its data points (see Section II-B1). Hence, if the algorithm is run twice it may generate two different partitions. There are various solutions to this problems (see Section II-B, [14], and references therein). Arguably, the most common solution is to run *k*-means a number of times and then select as the final partition that with the lowest sum of within-cluster distances as computed by Equation (2). However, there is no guarantee that the optimal *k*-means partition is among those found.

The key, one may argue, is to have a computationally cheap method of identifying the set of centroids leading to the best possible *k*-means partition. Over the years, a number of algorithms have been developed, each with the intention of selecting a set of good initial centroids (see Section II). However, to this day there is still no clear guideline for practitioners to follow [15]. The contribution of our paper is to be the most comprehensive survey to date containing empirical results. Here we compare the results of 17 algorithms on various data set configurations.

## II. BACKGROUND

### A. PRIOR SURVEYS

Given the widespread popularity of *k*-means, surprisingly few surveys containing empirical comparisons of initialisation algorithms have been published. We are interested in neutral surveys, by which we mean surveys which are not introducing a new initialisation. In this section we offer a brief overview of such surveys.

The earliest survey we found [16] presents an empirical comparison of four *k*-means initialisation algorithms run against three real-world data sets. In their experiments, the authors did not presume to know the number of clusters in advance. Rather, several values were explored for each data set. He *et al.* [17] goes a bit further and presents a comparative study of five initialisation algorithms on 25 two-dimensional synthetic, and four real-world, data sets. These papers certainly present very interesting results. However, current computing resources allow for a considerable increase in the scope of such experiments.

For instance, Steinley and Brusco [18] critically evaluate 12 *k*-means initialisation algorithms on synthetic and real-world data sets. Their experiments include various algorithms we also compare (e.g. Milligan, Bradley & Fayyad, Intelligent *k*-means, Faber, Hand & Krzanowski). However, because their paper was published over a decade ago they were unable to include some of the more recent algorithms we experiment with (Section II-B describes the algorithms we compare in chronological order). One of these, *k*-means++ (for details, see Section II-B10), is currently the default *k*-means option in popular software packages used for clustering, such as MATLAB, R, Weka, and scikit-learn [19]–[22]. Celebi *et al.* [23] perform an empirical comparison of eight *k*-means initialisation algorithms (some of which also covered in our work) on real-world and synthetic data sets. Their comparison is made in terms of cluster recovery (measuring the *k*-means criterion output), and CPU time. Their work makes various interesting recommendations.

Regarding our work, given the computational power we have nowadays, we were able to compare a higher number of algorithms. We measured cluster recovery in terms of adjusted Rand index and *k*-means criterion output. We have also measured how CPU time increases for each algorithm as the number of points in a data set increases. Finally, we find it important to provide an unbiased comparison. Hence, we do not propose any new algorithm.

### B. ALGORITHMS COVERED

This section discusses each of the *k*-means initialisation algorithms explored in our experiments. In order to ensure our experiments are reproducible we state any assumptions or arbitrary decisions we may have been forced to make. This is particularly the case when an algorithm requires a parameter but there is no clear method of how to identify its optimal value. In addition, we made available all of our source code under a permissive open-source licence via GitHub.[1]

We believe this survey to be more extensive than any previous work in this field. We selected the algorithms we experiment with (for details see Section II-B) based on different factors, not least academic influence. Some algorithms were chosen based on popularity: for example *k*-means++ is the default initialisation algorithm used in Python's scikit-learn and MATLAB and so must not be excluded.

[1] https://github.com/simonharris/pykmeans

Other algorithms (including Global *k*-means, random and Bradley & Fayyad) are considered canonical and are cited widely in the relevant literature. Further, we include, among others, the Onoda, Hatamlou and Khan works as some of the most recent algorithms published in this field.

Whilst many of these algorithms have been covered in surveys before, such as those discussed in Section II-A, we believe that no direct comparison of all of these algorithms has been published thus far.

#### 1) RANDOM INITIALISATION (RANDOM)

This is a rather simple initialisation algorithm, usually attributed to Forgy [9]. Assign each $x_i \in X$ to a cluster $S_k \in S$, chosen uniformly at random. The initial centroids for *k*-means $c_1, c_2, \ldots, c_K$ are the component-wise mean of each cluster $S_1, S_2, \ldots, S_K$. Clearly, these initial clusters are unlikely to be homogeneous—contrary to the very definition of cluster. Other initialisation algorithms are often compared in their original publication against this random initialisation (and sometimes against others), so we find it important to reproduce a baseline here. This is a non-deterministic algorithm, so in our experiments we run it 50 times.

#### 2) CONTINUOUS *K*-MEANS (C*K*-MEANS)

Faber presents the *Continuous k-means* algorithm [24], which comprises both an initialisation strategy and a modification to the centroids updating process. In order to maintain consistency with our other experiments, here we consider solely the initialisation strategy.

This is, arguably, the most popular initialisation after *k*-means++. Its steps are quite straightforward. Select $K$ data points from $X$ uniformly at random, and then copy their values to $c_1, c_2, \ldots, c_K$. It is intuitive that the values of the initial centroids are more likely to come from denser regions. However, this algorithm offers no protection against two or more initial centroids coming from the same region of the data.

This initialisation is so popular that it is not uncommon to see descriptions of *k*-means that actually describe Continuous *k*-means, sometimes even using names like "standard" *k*-means. Continuous *k*-means is often used as a baseline for comparisons aiming to validate newer algorithms. Hence, we find it important to have it among the algorithms we experiment with.

#### 3) MILLIGAN (Milligan)

Hierarchical clustering algorithms also provide a popular source for initial centroids. This approach is usually attributed to the work of Milligan [25], [26], and involves the use of Ward's method [27]. Ward's method begins with a clustering containing $N$ singletons. It then iteratively merges the two clusters with the lowest Ward distance, given by

$$d_w(S_k, S_l) = \frac{|S_k||S_l|}{|S_k| + |S_l|} d(c_k, c_l), \tag{3}$$

where $S_k, S_l \in S$, with $c_k$ and $c_l$ being their centroids, respectively. The above identifies the two clusters whose merge leads to the lowest increase in within-cluster variance. This process repeats until there are $K$ clusters in the data set. Milligan states this approach tends to work better when clusters are well separated. Ward's method is a popular hierarchical clustering algorithm, which can be found in many software packages for data analysis.

The use of such hierarchical algorithm to identify suitable initial centroids for *k*-means is rather simple. One runs Ward's method on a data set $X$, identifying $K$ clusters. The component-wise average of each of these clusters is then used as a initial centroid. We formally present the steps of this algorithm under Algorithm 2.

```
Algorithm 2 Milligan (Milligan)
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres
  1) Set $S = \{S_1, S_2, \ldots, S_N\}$, so that each cluster contains
     one data point of $X$.
  2) Identify $S_k, S_l \in S$, the two clusters with minimum (3).
     Merge $S_k$ and $S_l$. Repeat this step until $|S| = K$.
  3) Return the centroids of each cluster in $S$.
```

#### 4) KATSAVOUNIDIS, KUO & ZHANG (KKZ)

Katsavounidis *et al.* [28] introduce a deterministic algorithm able to generate initial centroids for *k*-means. More recently, their work has been used as a baseline for experiments involving another initialisation algorithm we also experiment with [29] (for details see Section II-B15). Katsavounidis' algorithm begins by calculating the Euclidean norm of each $x_i \in X$, given by

$$||x_i||_2 = \sqrt{\sum_{v=1}^{V} x_{iv}^2}. \tag{4}$$

It then identifies $\underset{x_i \in X}{\operatorname{argmax}} ||x_i||_2$, and copies its values to $c_1$. The Euclidean norm of $||x_i||_2$ is the Euclidean distance between $x_i$ and the zero vector. Hence, this algorithm may benefit from a normalisation process that ensures the component-wise mean over all $x_i \in X$ is zero (which we adopt, for details see Section III). Katsavounidis's algorithm then identifies the other $K - 1$ centroids based on distances between entities and already identified centroids, given by $d_i$

$$d_i = \min_{c_k \in C} d(x_i, c_k). \tag{5}$$

A formal description can be found in Algorithm 3. When adopting this initialisation, one should have in mind that selecting $c_1$ based on the highest value of (4) is certainly sensitive to the presence of outliers.

#### 5) BRADLEY & FAYYAD (BF)

Bradley & Fayyad [30] introduce a *k*-means initialisation algorithm leading to initial centroids that are not corrupted by

**Algorithm 3** Katsavounidis *et al.* (KKZ)

```
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres
1) Set $C = \emptyset$. Identify the data point $\underset{x_i \in X}{\arg\max}\, ||x_i||_2$ using (4), copy its values to $c_1$ and
   add $c_1$ to $C$
2) Identify the data point $\underset{x_i \in X}{\arg\max}\, d(x_i, c_1)$, copy its values
   to $c_2$ and add $c_2$ to $C$.
3) Identify $\underset{x_i \in X \wedge x_i \notin C}{\arg\max}\, d_i$ using (5), copy its values to a new
   centroid $c_k$ and add $c_k$ to $C$.
4) If $|C| < K$ go to Step 3.
```

outliers. Their algorithm has two main loops (for details see Algorithm 4). The first runs a modified version of *k*-means in which if an empty cluster is found its centroid is set to the values of the farthest data point in the data sample, and *k*-means is re-run.

The original authors validate their algorithm by experimenting on real-world and synthetic data sets. Experiments on the latter present an average reduction on (2) of 9%. This certainly seems to be a very good result, but needs to be compared with more recent advancements. Another point of interest is that this algorithm uses only a small proportion of the data at a time. Hence, it may be suitable for large-scale clustering problems.

This algorithm has a few parameters that we need to set beforehand, which forced us to make some decisions in our experiments. We have adopted the original author's suggestion and set the number of subsamples $T = 10$, and the sample size $n = \frac{N}{T}$. Also, we set the initial centroids $IC$ to be chosen uniformly at random from $X$. Clearly, this makes the algorithm non-deterministic so we run it 50 times on each of the data sets we experiment with.

**Algorithm 4** Bradley & Fayyad (BF)

```
Input: Data set $X$; number of clusters $K$; number of subsamples $T$; sample size $n$; set of initial centroids $IC$
Output: Initial centroids
1) Set $X_t$ to be a subsample of $X$ such that $|X_t| = n$, for
   $t = 1, 2, \dots, T$.
2) Set $C'_t$ to be the set of $K$ centroids generated by a
   modified version of k-means (see Section II-B5) on $X_t$,
   initialised with $IC$, for $t = 1, 2, \dots, T$.
3) Set $C' = \bigcup_{t=1}^{T} C'_t$.
4) Set $C_t$ to be the set of $K$ centroids generated by k-means
   on $C'$, initialised with $C'_t$, for $t = 1, 2, \dots, T$.
5) Set $C$ to be the set of centroids $C_t$ which has the lowest
   criterion output (2).
```

### 6) GLOBAL *K*-MEANS (GKM)

Global *k*-means [31] presents an incremental approach that identifies one initial centroid for *k*-means at a time. It does so by setting the first centroid as the component-wise mean of over all $x_i \in X$, and then making a deterministic search over $x_i \in X$ to determine the other $K - 1$ centroids. The original authors' general idea behind Global *k*-means is that an optimal solution for (2) with $K$ clusters can be obtained by applying a series of local searches. The algorithm employs *k*-means itself for these local searches over each of the data points in the data set.

This is a popular algorithm which does not require any extra user-defined parameter (Algorithm 5 presents the exact steps to be followed). This algorithm also presented very good experimental results on real-world and synthetic data sets on its original publication. It is certainly of interest adding Global *k*-means to the set of algorithms we experiment with.

The original authors also introduce a fast version of their algorithm, which relates to the partitioning of the data using a $k-$d tree structure. As the latter may have a small impact on performance (in terms of cluster recovery) we do not use it here.

**Algorithm 5** Global *k*-Means (Gkm)

```
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres
1) Set $c_{1v} = N^{-1} \sum_{x_i \in X} x_{iv}$ for $v = 1, 2, \dots, V$, and
   $C = \{c_1\}$.
2) For each $x_i \in X$, run k-means with initial centroids $C \cup$
   $\{x_i\}$. Set a new centroid $c_k = x_i$ for that $x_i$ leading to
   the lowest (2). Add $c_k$ to C.
3) If $|C| < K$, go to Step 2. Otherwise output $C$.
```

### 7) YUAN *et al.* (Yuan)

Yuan *et al.* [32] introduce a deterministic initialisation for *k*-means. In this, clusters are formed based on densities. The component-wise mean of each of these clusters is then fed to *k*-means as initial centroids. The algorithm has a user-defined parameter, $\alpha$, which helps define the cardinality necessary for a set of data points to be considered one of these clusters. The value of $\alpha$ is subject to $0 < \alpha \leq K$. The original authors suggest $\alpha = 0.75$, and we use this value in our experiments. Algorithm 6 describes its steps in detail.

The original authors compare their algorithm to "standard" *k*-means (for details, see Section II-B2) on four real-world data sets. Yuan's algorithm certainly performs much better than the average of 10 *k*-means runs on these data sets. However, their experiments were not adjusted for chance. For instance, *k*-means yields an average (classification) accuracy of 0.617 over ten runs on the popular Iris data set. This is a little less than we would usually expect. In order to deal with this we have run each non-deterministic algorithm we experiment with (including its initialisation) 50 times per data set, and use an accuracy measure that is adjusted for chance (see Section III).

**Algorithm 6** Yuan *et al.* (Yuan)

```
Input: Data set $X$; number of clusters $K$; factor $\alpha$
Output: Initial cluster centres
1) Set $m = 1$.
2) Set $A_m = \{x_i, x_j\}$, where $x_i, x_j \in X$ and $\forall x_t, x_\tau \in X$,
   $d(x_i, x_j) \leq d(x_t, x_\tau)$. Set $X = X \setminus A_m$.
3) Set $x_\iota = \underset{x_\iota \in X}{\text{argmin}} \min_{x_t \in A_m} d(x_\iota, x_t)$. Add $x_\iota$ to $A_m$, and
   remove $x_\iota$ from $X$. Repeat until $|A_m| = \alpha \frac{N}{K}$.
4) If $m < K$, set $m = m + 1$ and go to Step 2.
5) Return the component-wise mean of $A_m$ (for $m = 1, 2, \ldots, K$) as an initial centroid.
```

### 8) HAND & KRZANOWSKI (HK)

Hand and Krzanowski [13] introduce a simulated annealing based algorithm, where *k*-means is run repeatedly and, with each iteration, ‘‘perturbations’’ are introduced. In other words, data points are moved at random between clusters before *k*-means is run again.

The intention is to introduce the possibility that the search process will be moved to a different part of the feature space at any stage of the process, and so may find a pathway to the global optimum solution rather than a local optimum one. Accordingly, with each iteration, the probability of perturbation is decreased as it is hoped that the algorithm begins to approach the global optimum.

The original authors demonstrate that the default options of software packages for clustering are not always the best. They do so by showing their algorithm to outperform S-PLUS’ implementation of a group-average hierarchical clustering initialisation, as well as Random initialisation (see Section II-B1) with 20 starts.

A drawback of Hand and Krzanowski’s approach, one could argue, is that it has three user-defined parameters (for details see the steps under Algorithm 7). Its perturbation of data points in each cluster is governed by a probability $\alpha$, a learning rate $\beta$, and a threshold for the maximum number of iterations $T$. In our experiments we follow all suggestions provided by the original authors. We set $\alpha = 0.3$, $\beta = 0.95$, and $T = 100$. In Step 3, the inertia, that is the output of (2), is deemed to have stabilised (indicating the algorithm has converged) if it is unchanged in 10 iterations. However, it is difficult to trust these would be the best parameters for all data sets we experiment with. Given the non-deterministic nature of this algorithm, we run it 50 times per data set in our experiments.

### 9) INTELLIGENT *K*-MEANS (ik_1 AND ik_2)

Intelligent *k*-means [33] is a successful *k*-means initialisation algorithm [34], [35] that can identify both the number of clusters, $K$, and a set of good initial centroids. It identifies anomalous clusters in the data, and uses their centroids as initial centroids for *k*-means (see the steps in Algorithm 8).

**Algorithm 7** Hand & Krzanowski (HK)

```
Input: Data set $X$; number of clusters $K$; probability $\alpha$; learning rate $\beta$; iterations threshold $T$
Output: Initial centroids
1) Run $k$-means on $X$, generating a clustering $S = \{S_1, S_2, \ldots, S_K\}$ with a criterion output $\text{SSE}_0$ measured
   with (2). Set $t = 1$.
2) Perturb each $S_k \in S$ by moving each $x_i \in S_k$ to a different (arbitrary) cluster with probability $\alpha$. Repeat
   $k$-means, leading to $\text{SSE}_t$.
3) Set $t = t + 1$, and $\alpha = \alpha\beta$. Stop if either inertia stabilises, or if $t = T$. Otherwise go to Step 2.
```

**Algorithm 8** Intelligent *k*-means (Ik)

```
Input: Data set $X$; number of clusters $K$; cardinality threshold $\theta$.
Output: Initial centroids
1) Set $C = \emptyset$, and $c_c$ to be the component-wise mean over all $x_i \in X$,
2) Set $c_t$ to be equal to the data point $x_t \in X$ that is the farthest from $c_c$ as per (1).
3) Run $k$-means on $X$ using $c_c$ and $c_t$ as initial centroids. Do not allow $c_c$ to move in the centroid update step of
   $k$-means. This will form clusters $S_c$ and $S_t$.
4) If $|S_t| \geq \theta$, add $c_t$ to $C$. In any case remove all $x_i \in S_t$ from $X$. If there are still data points in $X$ go to Step 2.
5) Run $k$-means on the original data set setting $K = |C|$, and using all $c_k \in C$ as an initial centroids.
```

Formally, intelligent *k*-means applies the anomalous pattern method to identify one cluster $S_t$ and its centroid $c_t$ by alternatingly minimising

$$W = \sum_{x_i \in X} d(x_i, c_t) + \sum_{x_i \in X} d(x_i, c_c),$$

where $c_c$ is the component-wise average over all $x_i \in X$.

This algorithm addresses a much harder problem than the other algorithms we deal with, that of identifying the number of clusters as well as the initial centroids. It would be unfair for us to provide more information (i.e. the number of clusters, $K$) to the other algorithms than we do to intelligent *k*-means. Hence, in our experiments we set $\theta = 1$ and select the initial centroids in two different ways:

1) Select the $K$ initial centroids identified by intelligent *k*-means that have the highest cardinality for $S_t$. We believe these would be the $K$ most representative initial centroids. We refer to this as ik_1.
2) Select the first $K$ initial centroids identified by intelligent *k*-means. These would be related to the $K$ most anomalous clusters. We refer to this as ik_2.

Setting $\theta = 1$ is likely to lead to an overestimation of the number of clusters, hence the two approaches we presented above. However, it is also possible for this algorithm to underestimate $K$. This happened in some of our experiments, which were then ignored when calculating cluster recovery. Such behaviour, perhaps not surprisingly, becomes more common as $K$ increases.

#### 10) $K$-MEANS++ (km++)

Arthur and Vassilvitskii introduce the $k$-means++ algorithm [36]. This is a very popular algorithm, and in fact the default $k$-means initialisation in MATLAB and scikit-learn. The first initial centroid is selected uniformly at random from $X$, and further centroids are selected with probability

$$P(x_i) = \frac{D(x_i)^2}{\sum_{x_t \in X} D(x_t)^2}, \qquad (6)$$

where $D(x_i)$ is the shortest distance from a data point $x_i \in X$ to the nearest centroid already chosen. Algorithm (9) formally describes the steps of $k$-means++.

**Algorithm 9** $k$-Means++ (km++)

```
Input: Data set $X$; number of clusters $K$
Output: Initial centroids
  1) Select a $x_i \in X$ uniformly at random, and copy its values to $c_1$
  2) Select a $x_i \in X$ with probability given by (6), and copy its values to a new centroid $c_k$.
  3) Repeat Step 2 until a total of $K$ initial centroids have been chosen.
```

The motivation behind $k$-means++ is rather intuitive. Given a Gaussian distribution, the first centroid is relatively likely to be found within a higher-density area of the data, ideally corresponding to a meaningful cluster. Similarly, selecting subsequent initial centroids based on increasing their probability of selection in line with their distance from already-selected centroids leans towards finding new centroids in different clusters. A particular advantage of this algorithm is that it performs a $\mathcal{O}(logK)$ approximation, which is a clear theoretical guarantee in terms of the quality of the obtained clustering.

The probabilistic approach adopted by $k$-means++ may lead to outliers being selected as initial centroids. Hence, the original authors ran this algorithm 20 times per experiment. Here, we run it 50 times per experiment in order to maintain consistency with our treatment of other non-deterministic algorithms we experiment with.

#### 11) SINGLE PASS SEED SELECTION (SPSS)

Pavan *et al.* [37] introduce the Single Pass Seed Selection (SPSS) algorithm, which is a modification of $k$-means++. SPSS is a deterministic $k$-means initialisation algorithm that seems to lead to a lower number of $k$-means iterations than $k$-means++. The original authors drew this conclusion after experimenting with 10 real-world data sets. They also seem to state their algorithm works better on high-dimensional data sets, however, only two of the data sets they experiment with have more than 50 features. Hence, we find it would be of interest to see if the latter claim holds true when we increase the number of features considerably (for the results, see Section IV).

SPSS begins by identifying the data point $x_i \in X$ that is the most similar to all $x_t \in X$. This seems to be a considerable computational effort that effectively identifies the data point that is the nearest to the component-wise average of all points in the data set (given the use of the Euclidean distance). If one applies the range normalisation to a data set (we did so, for details see Section III), the component-wise average of all data points in the data set is given by the zero vector. Hence, the data point $x_i \in X$ being sought is that which minimises $\sum_{v=1}^{V} x_{iv}^2$. The first centroid is set to this data point.

**Algorithm 10** Single Pass Seed Selection (SPSS)

```
Input: Data set $X$; number of clusters $K$
Output: Initial centroids
  1) Set $c_1 = \underset{x_i \in X}{\operatorname{argmin}} \sum_{x_t \in X} d(x_i, x_t)$, $C = \{c_1\}$.
  2) For each $x_i \in X$ set $D(x_i) = \min_{c_k \in C} d(x_i, c_k)$.
  3) Set $y$ to be the the sum of distances of the $\frac{N}{K}$ nearest data points to the last centroid added to $C$.
  4) Find the index $l$, such that $\sum_{t=1}^{l} D(x_t)^2 \geq y > \sum_{\tau=1}^{l-1} D(x_\tau)^2$. Set a new centroid $c_k = x_l$, and add it to $C$.
  5) If $|C| < K$ go to Step 2.
```

#### 12) ERISOGLU, CALIS & SAKALLIOGLU (ECS)

Erisoglu *et al.* [38] propose a $k$-means initialisation algorithm that first selects two features of a data set, and then uses only these for its computations. This algorithm iteratively identifies $K$ initial centroids for $k$-means by maximising the distances between data points and already identified centroids. We describe the process in Algorithm 11.

**Algorithm 11** Erisoglu *et al.* (ECS)

```
Input: Data set $X$; number of clusters $K$
Output: Initial centroids
  1) Identify the feature $v'$ as that with the highest coefficient of variation. Set $C = \emptyset$.
  2) Identify the feature $v''$, as that with the lowest absolute Pearson correlation to $v'$.
  3) Set $c_1 = \underset{x_i \in X}{\operatorname{argmax}}\, d(x_{iv'v''}, (\bar{x}_{v'}, \bar{x}_{v''}))$, and add $c_1$ to $C$.
  4) Set $c_2 = \underset{x_i \in X}{\operatorname{argmax}}\, d(x_{iv'v''}, c_{1v'v''})$, and add $c_2$ to $C$.
  5) If $|C| < K$, set a new centroid $c_k = \underset{x_i \in X}{\operatorname{argmax}} \sum_{c_l \in C} d(x_{iv'v''}, c_{lv'v''})$, and add $c_k$ to $C$.
     Repeat this step as necessary.
  6) Perform a clustering around $C$ using only the features $v'$ and $v''$. Return the component-wise means of the resulting clusters as the initial centroids.
```

This algorithm begins by calculating the coefficient of variation for $v = 1, 2, \ldots, V$ given by

$$cv_v = \left| \frac{\sigma_v}{\bar{x}_v} \right|, \tag{7}$$

where $\sigma_v$ and $\bar{x}_v$ are the standard deviation and mean of feature $v$ over all $x_i \in X$, respectively. The first selected feature is that with the highest coefficient of variation. Clearly, the above will not work if $\bar{x}_v = 0$ for any feature $v$. This will be the case if the data pre-processing stage includes either the range normalisation or minmax normalisation. Given we apply the former in our experiments (for details see Section III), we have reduced this to be solely the standard deviation.

The second feature to be selected is that with the lowest absolute correlation to the first identified feature. The original paper presents a correlation equation but we have only been able to reproduce their results using Pearson's correlation instead. Hence, we have adopted the latter in our experiments.

The algorithm follows to identify initial centroids based on the distance of data points to the centre of the data set, and the distance between data points and other previously identified centroids.

This algorithm shows very promising results when compared against the random initialisation on five real-world data sets. This is not a particularly high number of data sets, so we find it would be interesting to explore how it performs on other data sets (for that see Section IV). The approach of selecting a hard number of features (exactly two in this case) is, arguably, somewhat fragile. We understand the original authors were probably attempting to reduce complexity. However, there is no reason to believe exactly two features will always represent all characteristics of a given data set.

Given this algorithm calculates cumulative distances to all previously found centroids, it may select two nearby data points provided they have a large cumulative distance [15]. In our experiments, this algorithm sometimes even re-selected a previously found centroid—particularly if $K \geq 10$ (see Figure 10).

### 13) HATAMLOU BINARY SEARCH (BS)

Hatamlou [39] introduces a deterministic binary search algorithm capable of identifying good initial centroids for *k*-means. In this algorithm each centroid $c_k \in C$ is initially selected from a different part of the data set $X$. The algorithm then optimises the location of each of these centroids, with respect to (2), by exploring around them. This optimisation process occurs iteratively until either a maximum number of iterations is reached, or, the centroids have converged. Algorithm 12 formally presents the steps to be followed.

The original author validates his algorithm by experimenting on six real-world data sets, and comparing its results against those of seven other algorithms. Success is measured using the output of (2), and the $F$-Measure. The results shown on the original paper are certainly good. Hence, we find it would be of interest to see how this algorithm performs on a more comprehensive set of experiments.

**Algorithm 12** Hatamlou (BS)

```
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres
1) Set $SSM_v = \max_{x_{iv} \in X} x_{iv}$ and $g_v = \left( SSM_v - \min_{x_i \in X} x_{iv} \right) / K$, for $v = 1, 2, \ldots, V$.
2) Set $c_{kv} = \min_{x_i \in X} x_{iv} + g_v(k-1)$, for $k = 1, 2, \ldots, K$ and $v = 1, 2, \ldots, V$.
3) Assign each $x_i \in X$ to the cluster $S_k$ represented by its nearest centroid $c_k$, and calculate (2).
4) Set $k = 1$ and $v = 1$.
5) Set $c_{kv} = c_{kv} + SSM_v$, and recalculate (2).
6) If there is no improvement: (i) if $SSM_v < 0$, set $SSM_v = -\frac{1}{2} SSM_v$, or (ii) if $SSM_v > 0$, set $SSM_v = -SSM_v$.
7) If $v < V$, set $v = v + 1$, or if $v = V$ and $k < K$, set $k = k + 1$. In any of these two cases, go to Step 5.
8) If the termination criterion is not met, go to Step 4.
```

### 14) KHAN's SEED SELECTION ALGORITHM (KHAN)

Khan [40] presents an interesting algorithm to identify initial centroids for *k*-means. Using a single feature it calculates the "gaps" among data points in a data set. The data points delineated by these "gaps" are deemed to be members of the same initial clusters, with the intention of increasing the distances between points in separate clusters. The steps for this can be found in Algorithm 13.

**Algorithm 13** Khan

```
Input: Data set $X$; number of clusters K; initial feature for clustering $v'$
Output: Initial centroids
1) Sort the data points $\{x_{iv'} : x_i \in X\}$ in terms of increasing magnitude, such that $x_{1v'}$ and $x_{nv'}$ have the minimum and maximum magnitudes, respectively.
2) Set $D_i = x_{(i+1)v'} - x_{iv'}$ for $i = 1, \ldots, n-1$.
3) Sort $D$ in descending order without changing its indices. Identify the $K-1$ values of $i$ related to the $K-1$ highest values of $D$, leading to $(i_1, \ldots, i_{(K-1)})$.
4) Sort $(i_1, \ldots, i_{(K-1)})$ in ascending order.
5) The corresponding set of indices of data points $x_i \in X$ which are the lower bounds of clusters $S_1, S_2, \ldots, S_K$ are defined as $(i_0, i_1 + 1, \ldots, i_{K-1} + 1)$, where $i_0 = 1$.
6) The centroid $c_k$ is calculated as the component-wise mean of the $x_i$ values falling within the upper and lower bounds calculated above.
```

The original publication shows Khan's method to outperform *k*-means++ in terms of running time and the *k*-means criterion (2). The original paper also shows this method to lead to a lower variance of centroids than *k*-means++.

The latter is hardly surprising given that Khan's method is deterministic and *k*-means++ is not.

The decision to define an initial partition based on a single feature is certainly unexpected. Unfortunately, no guidance is provided as to which feature should be selected from a data set for this purpose. In our experiments, we made the decision to select a feature uniformly at random. This effectively modifies the algorithm from deterministic to non-deterministic, so we performed 50 runs for each data set as with all other non-deterministic algorithms.

#### 15) ONODA, SAKAI & YAMADA (OSY_1 AND OSY_2)

Onoda *et al.* [29] introduce a deterministic *k*-means initialisation algorithm based on independent component analysis [41], [42]. In their experiments it outperforms *ck*-means, *k*-means++ and KKZ (for details on these algorithms, see Sections II-B2, II-B10 and II-B4, respectively), when the ratio between the number of features and the number of data points is less than 10. Also, their initialisation seems to be faster than *k*-means (when run 100 times). However, the largest data set for this particular experiment has 690 data points each described over 8,261 features. We think it would be interesting to see how this initialisation performs in terms of cluster recovery and speed on a higher number of data sets, under different configurations.

Algorithm 14 presents the steps for this in detail (which we refer to as OSY_1). The original authors also introduce a very similar alternative initialisation algorithm. The only difference is that it uses principal component analysis, instead of independent component analysis. We also perform experiments using the latter (see Section IV), and refer to it as OSY_2.

Given its use of independent component analysis (or principal component analysis) it is unclear how this algorithm would produce a set of initial centroids if $V < K$. In our experiments we were forced to ignore such cases.

```
Algorithm 14 Onoda et al. (OSY_1)
Input: Data set $X$; number of clusters $K$
Output: Initial cluster centres
  1) Extract $K$ independent components $IC_1, IC_2, \ldots, IC_k, \ldots, IC_K$ from $X$.
  2) Select $K$ initial centroids $c_1, c_2, \ldots, c_k, \ldots, c_K$,
     selecting $c_k = x_i \in X$ with a minimum $\frac{IC_k x_i}{|IC_k||x_i|}$
```

## III. EXPERIMENTAL SETUP

In this section we present the data sets we use in our experiments (synthetic and real-world), as well as our data preprocessing stage.

In terms of synthetic data sets, we generated 50 of these for each of the configurations stated in Table 1, leading to a total of 6,000 data sets. Each data set contains isotropic Gaussian clusters, which we generated with scikit-learn's `make_blobs()` function. The tool used to generate the datasets is available, with documentation of its use, under a permissive open-source licence via GitHub.[2]

**TABLE 1.** The configurations used to generate our synthetic data sets. There are 50 data sets for each of the above, leading to a total of 6,000 synthetic data sets. Each data set contains isotropic Gaussian clusters.

| Parameter | Value |
|---|---|
| Clusters ($K$) | 2, 5, 10, 20 |
| Features ($V$) | 2, 10, 50, 100, 1000 |
| Data points ($N$) | 1000 |
| Clusters cardinality | Uniform, random |
| Within-cluster standard deviation | 0.5, 1, 1.5 |

The chosen configurations allow us to analyse each algorithm in various ways. For instance, we can determine how each algorithm performs when the number of clusters and/or features increases. We can also analyse the impact of having clusters with a uniform cardinality (i.e. each cluster in a data set has the same number of data points) or random (i.e. each cluster in a data set has a random number of data points, subject to a minimum of 30). Finally, our experimental setup allows us to analyse how cluster separation impacts each algorithm. A higher within-cluster standard deviation implies higher cluster overlap.

The values of the parameters in Table 1 were chosen with the aim of generating a wide variety of data set characteristics within practical constraints. Whilst many of the lower bounds are self-explanatory—for example, cluster recovery on a data set where $K = 1$ would be a trivial problem and of no interest—intervals and upper bounds are chosen on the understanding that it simply is not possible to run algorithms against data sets of every conceivable size and shape.

All data sets are composed of Gaussian clusters, in other words clusters convex in shape with density increasing towards the centroids, since this is the clustering problem addressed by *k*-means.

We feel our experiments would not be complete without results on real-world data. With this in mind we downloaded 27 data sets from the popular UCI Machine Learning repository [43], and sourced the Chernoff Fossil data set from [44]. This provides us with 28 real-world data sets with varying characteristics which are detailed in Table 2.

We have normalised all data sets we experiment with using

$$x_{iv} = \frac{x_{iv} - \bar{x}_v}{max(x_v) - min(x_v)}, \tag{8}$$

where $\bar{x}_v$ is the mean of feature $v$, given by $N^{-1} \sum_{x_i \in X} x_{iv}$. This approach forced us to remove any feature whose range is zero (i.e. a feature $v$ whose value is constant for all $x_i \in X$). We have opted for (8) rather than the popular $z$-score because the latter is biased towards unimodal distributions. This is certainly easier to explain with an example. Let $v_1$ and $v_2$ be unimodal, and multimodal, respectively. Clearly, the standard deviation of $v_2$ will be higher than that of $v_1$. Hence, the

[2] https://github.com/simonharris/pycleandata

**TABLE 2.** The characteristics of the real-world data sets.

| Data set | Points | Features | Clusters |
|---|---|---|---|
| Avila | 10,430 | 10 | 12 |
| Blood Transfusion | 748 | 4 | 2 |
| Breast Cancer (Diag.) | 569 | 30 | 2 |
| Breast Cancer (Orig.) | 683 | 9 | 2 |
| Breast Tissue | 106 | 9 | 6 |
| Ecoli | 336 | 7 | 8 |
| Fossil | 87 | 6 | 3 |
| Glass | 214 | 9 | 6 |
| HTRU2 | 17,898 | 8 | 2 |
| Haberman | 306 | 3 | 2 |
| Iris | 150 | 4 | 3 |
| Leaf | 340 | 15 | 30 |
| Letter Recognition | 20,000 | 16 | 26 |
| Libras Movement | 360 | 90 | 15 |
| Musk 1 | 476 | 166 | 2 |
| Musk 2 | 6,598 | 166 | 2 |
| Optical Recognition | 3,823 | 62 | 10 |
| Page Blocks | 5,473 | 10 | 5 |
| Parkinsons | 195 | 22 | 2 |
| Pen-Based Recognition | 7,494 | 16 | 10 |
| Sonar all | 208 | 60 | 2 |
| Spambase | 4,601 | 57 | 2 |
| Vehicle Silhouettes | 846 | 18 | 4 |
| Vertebral Column | 310 | 6 | 3 |
| Wine | 178 | 13 | 3 |
| Wine Quality (Red) | 1,599 | 11 | 6 |
| Wine Quality (White) | 4,898 | 11 | 7 |
| Yeast | 1,484 | 8 | 10 |

$z$-scores for $v_2$ will be lower than those of $v_1$. However, we are interested in the clustering information most likely present in the modes of $v_2$ rather than that contained in the single mode of $v_1$.

Here we experiment with deterministic and non-deterministic algorithms. We run non-deterministic algorithms 50 times on each data set, real-world or synthetic. This is the reason why some of our results present a mean and standard deviation (non-deterministic) and others do not (deterministic) on the experiments with real-world data sets. Given all algorithms are presented with the correct number of clusters, we disregard any experiment for which the number of identified centroids is not $K$.

Finally, we measure cluster recovery in two ways. First, given we have the labels of each and every data set we experiment with, we use the Adjusted Rand Index (ARI) [45]. This is a corrected-for-chance version of the Rand Index [46]. The ARI between the clusterings $S, S'$ is given by

$$ARI(S,S') = \frac{\sum_{kl}\binom{N_{kl}}{2} - [\sum_k \binom{a_k}{2} \sum_l \binom{b_l}{2}]/\binom{N}{2}}{\frac{1}{2}[\sum_k \binom{a_k}{2} + \sum_l \binom{b_l}{2}] - [\sum_k \binom{a_k}{2} \sum_l \binom{b_l}{2}]/\binom{N}{2}},$$

where $N_{kl} = |S_k \cap S'_l|$, $a_k = |S_k|$, and $b_l = |S'_l|$. Second, we analyse inertia—that is, the $k$-means criterion output given by Equation (2). We do so because this is the actual value iteratively minimised by $k$-means. Hence, any good initialisation algorithm should improve inertia.

In Section IV we describe the many experiments we ran and discuss their results. These experiments are intended to show how each of the initialisation algorithms described in Section II-B is affected by the different parameters of each data set. We measure this based on four measures: (i) cluster recovery given by the ARI between the found clustering and the known labels; (ii) the minimisation of the $k$-means criterion (inertia) given by Equation (2); (iii) the number of iterations $k$-means takes to converge under each initialisation; and (iv) how $k$-means scales to much larger data sets, in terms of convergence time, also under each initialisation. We also perform experiments with real-world data in order to represent scenarios in which the distribution of the data is unknown.

## IV. RESULTS AND DISCUSSION

Here, we measure the performance of each of the initialisation algorithms discussed in Section II in terms of cluster recovery and computational performance. First, we measure cluster recovery by applying the Adjusted Rand Index (9) to compare the clustering produced by each algorithm on each data set against the true labels, and by measuring inertia. Second, we measure computational performance by counting the number of iterations $k$-means takes to converge under each initialisation on each data set (as per previous work [47]), and by analysing how each algorithm behaves when the amount of synthetic data is increased considerably.

### A. CLUSTER RECOVERY ON SYNTHETIC DATA SETS

Figure 1 shows the average cluster recovery, measured using the Adjusted Rand Index (ARI) for each of the algorithms we experiment with. This particular figure presents results broken down by the number of clusters (i.e. synthetic data sets with 2, 5, 10, and 20 clusters). We can see that on data sets containing two clusters, all algorithms perform similarly on average. This in itself is an important result. When we increase the number of clusters some algorithms (eg. BS, Khan, and SPSS) begin to perform rather poorly, reaching average values of ARI under 0.5—this is lower than that achieved by the random initialisation.

Of course, there are reasons for this. BS takes centroids from $K$ different parts of the data, with each part calculated independently for each feature. Hence, $K$ is proportional to the probability of a mismatch between centroids and dense areas, which is probably the reason for the poor result. In the case of Khan, centroids are calculated using a single feature. Thus, as $K$ increases the probability of one feature containing information about all $K$ clusters decreases.

The initialisations gkm, HK, ik_1, ik_2, KKZ, km++, MI, and Yuan seem to be those that are the most resistant to an increase of $K$. We note that ik_1 and ik_2 were the only initialisations producing a higher ARI for $K = 20$ than for $K = 2$. However, this may be a side-effect of the fact that, as mentioned in Section II-B9, both ik_1 and ik_2 were unable to find the pre-specified number of clusters in all cases, particularly as $K$ increases. Figure 12 in the Appendix shows the same experiment but measuring inertia. This shows a very similar pattern, with the major difference that now OSY_1 clearly performs worse than the random initialisation when $K \neq 2$.

Figure 2 shows the average ARI on the same data sets but now broken down by the number of features in each data set (i.e. 2, 10, 50, 100, and 1000). Generally speaking, the increase in the number of features usually leads to better cluster recovery in our experiments. This is probably because these are synthetic data sets containing Gaussian clusters, and the maximum number of features we experiment with is 1,000. The only exceptions to this trend, with poor results, were BS, Khan, and SPSS. The reasons for this are probably similar to those for our experiments broken down by the number of clusters. For instance, an increase in $V$ has a similar effect to that of an increase in $K$ for BS. The initialisations OSY_1 and OSY_2 produced the highest average ARI for $V = 2$, this seem to suggest these are good initialisations for this scenario. However, OSY_1 and OSY_2 are unable to produce a set of initial centroids when $V < K$ (see Section II-B15). Hence, the results shown for $V = 2$ only include the scenario where $K = 2$. For the other values of $V$ (i.e. 10, 50, 100, and 1000) gkm, HK, ik_1, ik_2, KKZ, km++, and Milligan produced the best results—followed closely by Yuan. Figure 13 in the Appendix shows the same experiments but measuring inertia. It is fair to expect inertia to increase when $V$ increases. However, we can see considerable differences in the degree of increase, this being consistent with our ARI-based results. BS, Khan, and SPSS had much worse inertia than the random initialisation. OSY_1 and ck-means performed close to random, being slightly worse and slightly better, respectively.

Figure 3 shows the average ARI on the same data sets but now broken down by the within-cluster standard deviation. A higher within-cluster standard deviation means clusters are more sparse, leading to a higher chance of overlap. The highest ARI for a standard deviation of 0.5 was given by gkm, ik_1, ik_2, and Milligan—followed very closely by HK, KKZ, km++, and Yuan. With a higher standard deviation we can see a small lead by ik_1 and ik_2. Figure 14 in the Appendix shows the same experiments but measuring inertia. The pattern is basically the same, with the small difference that HK now performs slightly better then ik_1 and ik_2, but still very much similar to gkm and Milligan.

Figure 4 shows the average ARI on the same data sets but broken down by cardinality of clusters. A uniform cardinality means that each cluster had the same number of data points. A random cardinality allows clusters to have a uniformly random number of data points, subject to a minimum of 30 data points per cluster. Surprisingly some algorithms had very similar (and good) results for both experiments. Again, we can see a small lead by ik_1 and ik_2. Figure 15 in the Appendix shows the same experiments but measuring inertia. Generally speaking the pattern in this is very much the same. However, ik_1 and ik_2 are now slightly below gkm, HK, KKZ, km++, and Milligan—but still among the best.

### B. *K*-MEANS ITERATIONS ON SYNTHETIC DATA SETS

In this section we discuss the impact of each initialisation algorithm on the number of iterations *k*-means takes to converge. The lower the number of iterations the better. We do not show the results for gkm and HK because we feel it would be unfair to do so. In our experiments, *k*-means took a single iteration to converge when initialised with either gkm or HK, which may seem like an incredibly good result. However, these algorithms already incorporate *k*-means (without changes) within them, and run it multiple times.

Figure 5 shows the average number of iterations *k*-means took to converge when initialised by each algorithm, broken down by the number of clusters on each data set. The best results are given by ik_1, ik_2, and Milligan. It is interesting to see that these algorithms lead to similar performance when the number of clusters $K = 2$. When the number of clusters is 5 or 10 Milligan outperforms ik_1 and ik_2. However, when the number of clusters is 20, ik_1 and ik_2 take the lead again. This may be because although very good, Milligan is based on hierarchical clustering. The latter (unlike *k*-means) does not revisit assignments of data points to clusters. The assignment of a data point to a cluster is final (with the obvious exception where clusters are merged), and future clusters are identified based on previous clusters. Figure 6 shows that the major difference between these algorithms happens when the number of features is two. Figures 7 and 8 show somewhat similar performance between these.

### C. COMPUTATIONAL PERFORMANCE

In this section we analyse the computational performance of each of the algorithms we experiment with (for details on the algorithms, see Section II-B) as the amount of data being processed increases. We favoured an empirical approach to do so, following the general theme of this paper. We ran all of our experiments on a CentOS Linux computational cluster. This contains 1097 64-bit processing cores using a balanced mix of Intel E5-2698, Intel Gold 5115, 6152, and 6238L CPUs — with up to 6 Tb of RAM. In terms of software we used Python 3.6. In this type of analysis one must be very careful: our objective is to analyse the algorithms themselves and not their implementations. With this in mind we devised the following methodology, with the intention of producing a fair comparison.

First, we select a particular data set from those we generated. We deliberately choose a data set with rather unexceptional characteristics, in other words, each parameter lies somewhere in the middle of the ranges we explore. In our case we used the configuration whereby $K = 5$, $V = 50$, the clusters' cardinality is uniform, and the within-cluster standard deviation is 1. Second, we run each algorithm on this data set, storing the elapsed time it took to complete. We use this time as a baseline for each algorithm. Third, we merge

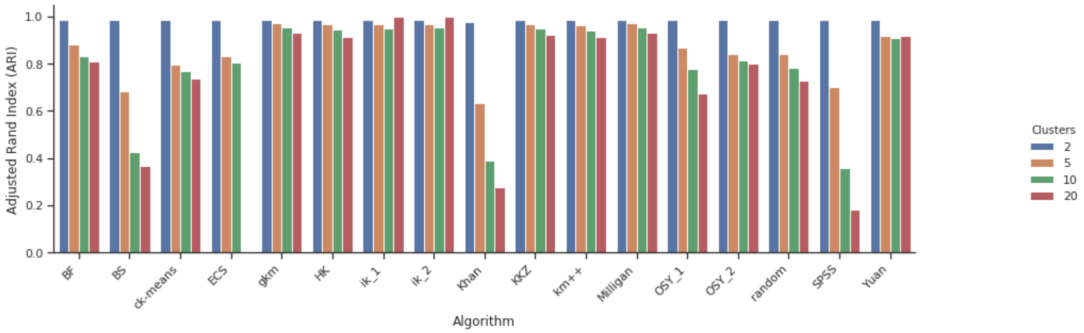

**FIGURE 1.** The average Adjusted Rand Index of each algorithm over all synthetic data sets experimented with, broken down by the number of clusters.

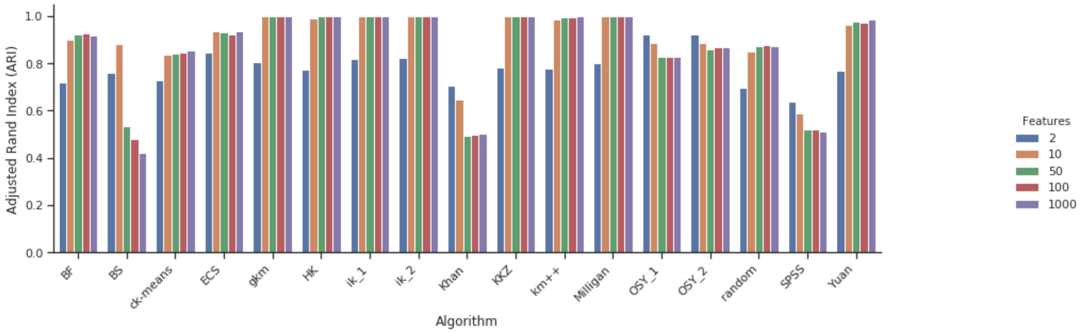

**FIGURE 2.** The average Adjusted Rand Index of each algorithm over all synthetic data sets experimented with, broken down by the number of features.

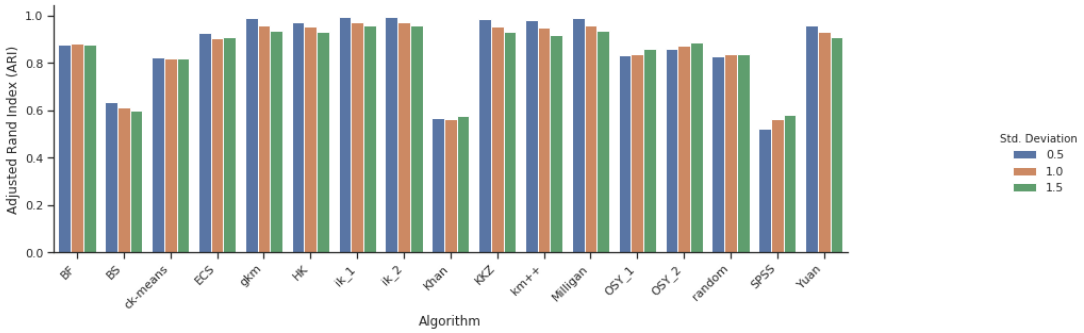

**FIGURE 3.** The average Adjusted Rand Index of each algorithm over all synthetic data sets experimented with, broken down by the within-cluster standard deviation.

as many copies of the original data set as desired to create a larger data set. In our experiments, the original data set has 1,000 points. We merged it enough times as to create versions of this data set with 2,000, 10,000, 50,000, and 100,000 data points. Finally, we run each algorithm on each of the new data sets, computing the elapsed time. We divide this elapsed time by the baseline time.

Figure 9 shows the factors generated by the process above for each of our algorithms plotted on a logarithmic scale. We can see the best computational performance was given by HK, ik_1, ik_2 (with ik_2 producing slightly better results on larger data sets than ik_1), OSY_1, and OSY_2 (with OSY_2 producing slightly better results than OSY_1). It was unsurprising to find gkm among the worst performers, since its original authors acknowledge its limitations in this regard, and even propose methods to improve its computational efficiency, albeit at the expense of its accuracy.

Milligan's initialisation is based on a hierarchical clustering algorithm (see Section II-B3) with a time complexity of $\mathcal{O}(N^2)$ [48]. Hence, its results on this set of experiments is expected. SPSS and Yuan also did not perform particularly

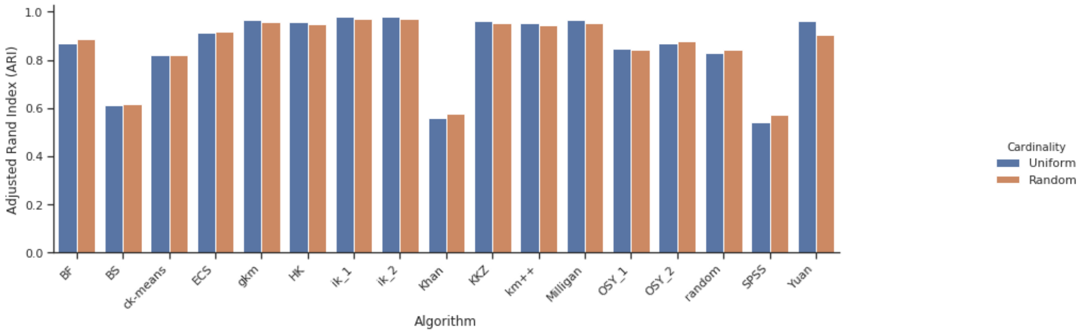

**FIGURE 4.** The average Adjusted Rand Index of each algorithm over all synthetic data sets experimented with, broken down by the cardinality of clusters.

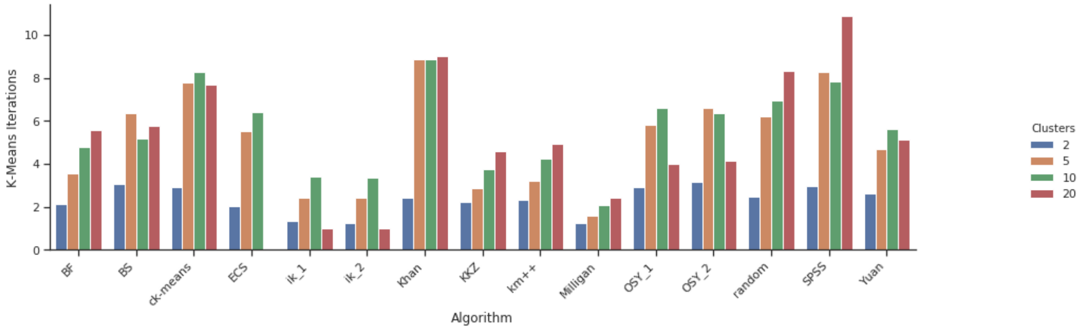

**FIGURE 5.** The average number of iterations *k*-means took to converge when initialised by each algorithm over all synthetic data sets experimented with, broken down by the number of clusters.

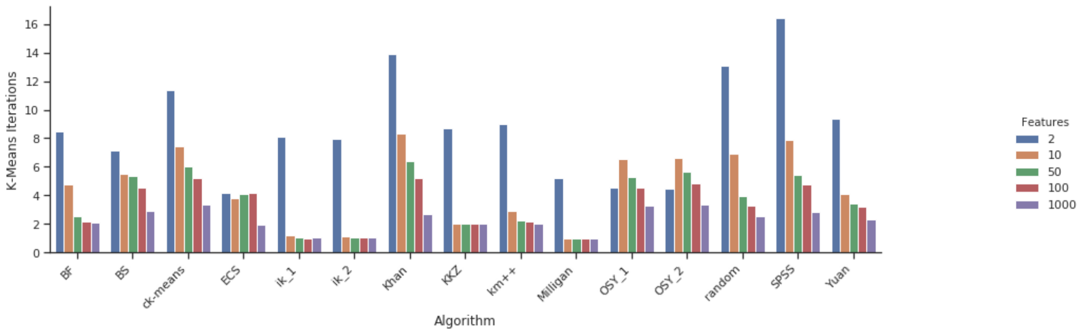

**FIGURE 6.** The average number of iterations *k*-means took to converge when initialised by each algorithm over all synthetic data sets experimented with, broken down by the number of features.

well. In fact, we were unable to complete the experiment with the latter on 100,000 data points within a practical timeframe (we ran it for three weeks).Still, it is clear from the results at 50,000 samples that this algorithm does not scale well. This behaviour is also expected, given Yuan computes all pairwise distances in a data set.

### D. RESULTS ON REAL-WORLD DATA SETS

The results on real-world data sets are certainly more mixed and difficult to read. This is expected as we lose the ability to control the properties of each data set. Having said that, we were still somewhat surprised to see all algorithms performed about the same in terms of cluster recovery (measured with the ARI), on average (see Figure 10).

In our experiments, we run each non-deterministic algorithm 50 times. Hence, we were able to calculate the standard deviation of an algorithm when applied to a particular data set. Figure 10 also shows the average of these standard deviations, which were not particularly high. Additionally, we show the proportion of incorrect number of clusters. This relates to the number of times an initialisation algorithm returned a clustering $S$ so that $|S| \neq K$. In our experiments, we considered $K$ to be known so we ignored such cases in our other results.

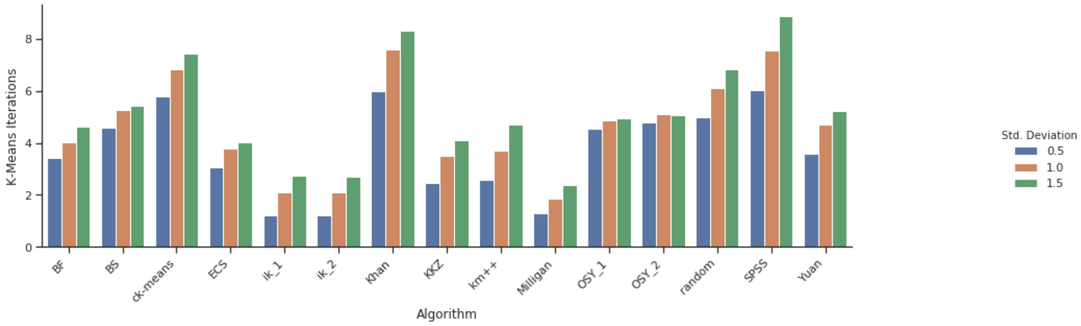

**FIGURE 7.** The average number of iterations *k*-means took to converge when initialised by each algorithm over all synthetic data sets experimented with, broken down by the within-cluster standard deviation.

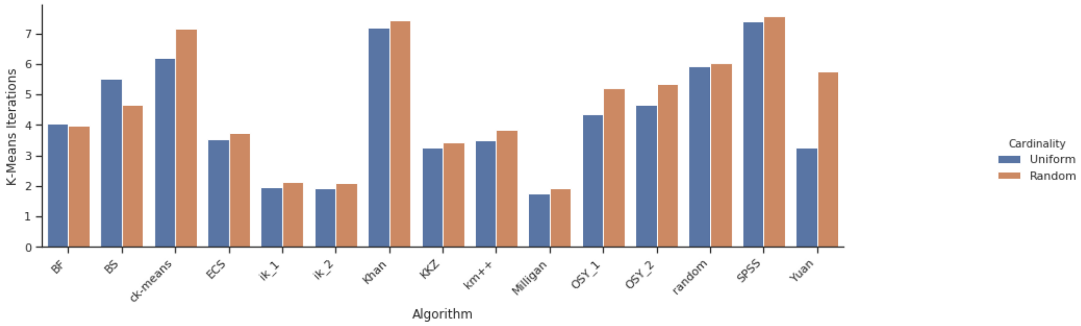

**FIGURE 8.** The average number of iterations *k*-means took to converge when initialised by each algorithm over all synthetic data sets experimented with, broken down by the cardinality of clusters.

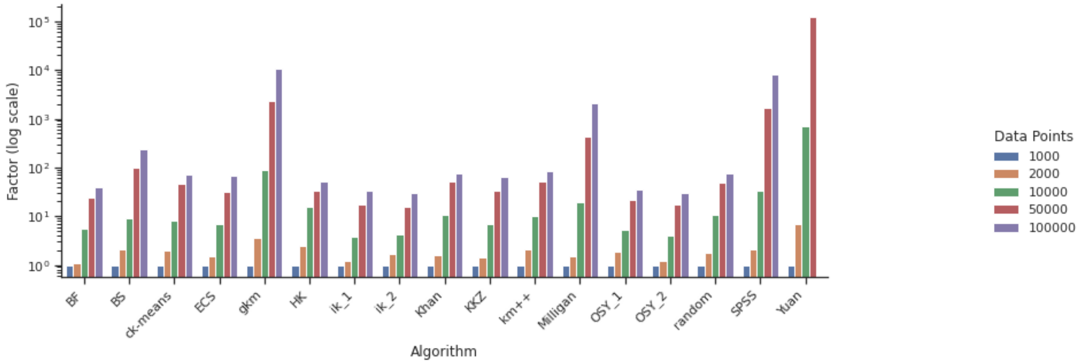

**FIGURE 9.** Computational performance of each algorithm plotted on a logarithmic scale. The $y$ axis represents how many times more CPU time an algorithm requires to converge in relation to an original data set with 1,000 data points.

Figure 11 shows the average number of iterations $k$-means took to converge when initialised by each of the algorithms we experiment with. It was interesting to see BF and ECS among the best given they did not perform well in our other experiments. Good results were obtained by OSY_2. This Figure (as well as Figure 10) also shows a small advantage of ik_1 over ik_2.

## V. RECOMMENDATIONS FOR PRACTICAL APPLICATIONS

In this section we introduce a small number of brief example use cases exploring how the results presented in this paper may be used to inform the selection of an appropriate $k$-means initialisation algorithm. The use cases presented are not exhaustive, nor are they intended to be. Rather, we hope they provide insight into some ways in which our outcomes might be of value to the wider research community, and the thought processes that may be involved in making use of our results.

### A. VECTOR QUANTISATION

Vector quantisation is a data compression technique based on clustering. Essentially, one applies a clustering algorithm to a

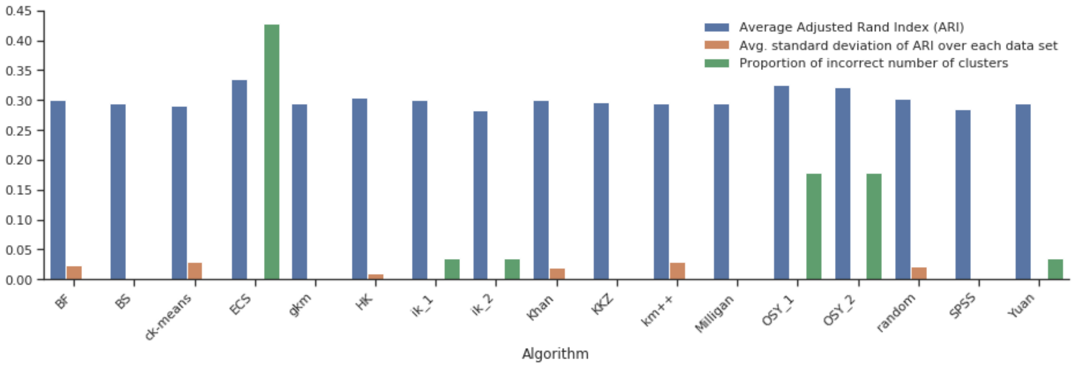

**FIGURE 10.** **The average Adjusted Rand Index (ARI), the average of the standard deviation of ARI over each data set, and the proportion of incorrect number of clusters found of each algorithm over all real-world data sets experimented with.**

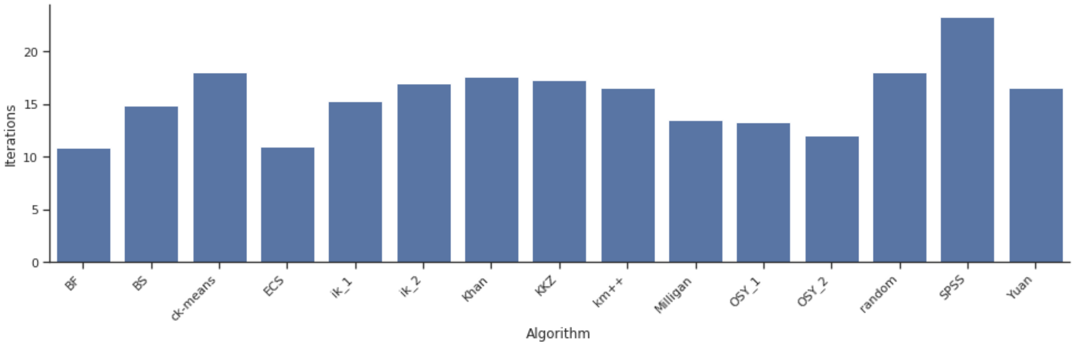

**FIGURE 11.** **The average number of iterations *k*-means took to converge when initialised by each algorithm over all real-world data sets experimented with.**

data set, so that each data point is represented by a centroid. This technique is particularly suitable for data sets consisting of many points, in which the distances between points and respective centroids are not particularly high.

Given the above one might wish to select a *k*-means initialisation algorithm providing good cluster recovery performance on data sets with low within-cluster standard deviation. From Figure 3 we see that Intelligent *k*-means (ikm_1 and ikm_2) consistently shows excellent performance in terms of ARI where within-cluster standard deviation is low, as do gkm, HK, KKZ, km++, Milligan, and Yuan.

Data compression is most likely to be desirable when a data set is very large. Hence, the most suitable initialisation must scale well and lead to a low number of *k*-means iterations. Figure 7 deals with the latter. In this we can see that from the algorithms previously identified, gkm and HK show excellent performance by this measure, typically requiring a single *k*-means iteration for convergence. This is very closely followed by ik_1, ik_2, and Milligan. We can filter our choice of algorithm further by analysing how these algorithms scale. Figure 9 helps us to do so. In this we can see that gkm and Milligan do not scale particularly well.

Finally, all other factors being equal, we might consider a deterministic algorithm to be preferable due to the reproducibility of its results, and the lack of a requirement for an arbitrary parameter to be set, specifically the number of restarts to be performed. Furthermore, only having to perform one execution of an algorithm rather than many is clearly beneficial for performance reasons. HK is non-deterministic, so we lean away from recommending it. Hence, in this scenario one could well consider ik_1 or ik_2.

### *B. REGRESSION ANALYSIS*

Regression analysis is a quite popular set of techniques used to estimate the relationship between dependent and independent variables. Unfortunately, regression can be prohibitive for large data sets. A popular approach to to deal with this issue involves clustering: once the data set has been clustered, the regression analysis may be performed upon the resulting centroids. In general, the accuracy of the regression is directly proportional to the number of centroids available, and how good the actual clustering is. Hence, a suitable initialisation would display strong cluster recovery performance where the number of clusters $K$ is high.

Figure 1 shows we can immediately disregard certain initialisations for this scenario. For instance, ECS was unable to successfully cluster the data with $K = 20$. Further, the ARI values returned by BS, Khan and SPSS degrade markedly as $K$ increases. The remaining algorithms showing relatively strong cluster recovery performance for higher values

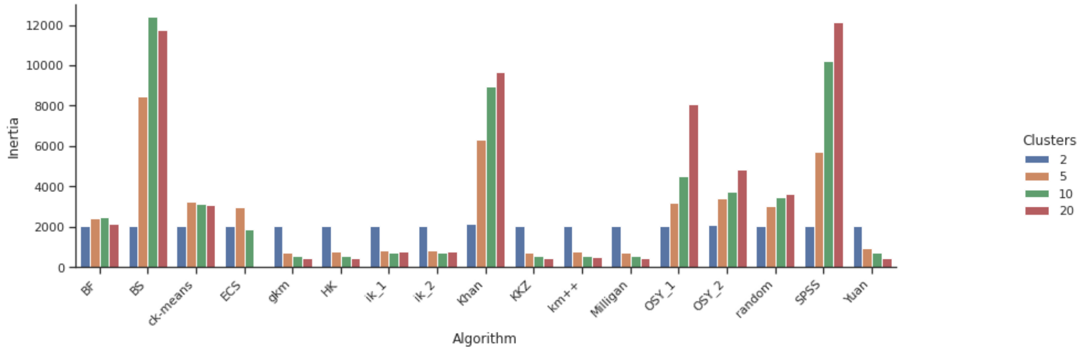

**FIGURE 12.** The average inertia of each algorithm over all synthetic data sets experimented with, broken down by the number of clusters.

of $K$ are ik_1, ik_2, gkm, HK, KKZ, km++, Milligan, and Yuan, so one might consider these to be an initial shortlist for recommendation.

We note that both ik_1, and ik_2 actually show better cluster recovery performance at $K = 20$ than at lower values, however, the results in those cases may not be truly representative (for details see Sections II-B9 and IV-A). In some experimental runs of these algorithms we were not able to produce clusterings with the correct $K$. Given the above, the results we show for $K = 20$ are an approximation, hence, we cannot be sure this is the most suitable algorithm.

This scenario concerns solely large data sets—so that regression may require clustering in the first place. Thus, we must now consider computational performance. Section V-A discusses the fact gkm and Milligan are computationally prohibitive for large data sets. Hence, one cannot recommend their use in this scenario as well. Of the remaining algorithms, Figure 5 shows KKZ, *k*-means++, and Yuan, to perform well in terms of the number of iterations these take to converge at a higher number of clusters. However, one should note HK converges in a single iteration. Figure 9 shows Yuan is not a particularly scalable initialisation, hence, one cannot recommend it for this scenario. Finally, in this scenario one could consider using km++, HK, or KKZ.

## VI. CONCLUSION AND FURTHER WORK

In this paper we considered the highly popular *k*-means clustering algorithm, and its shortcomings (for details, see Section I). Here, our primary concern was that the clusterings generated by *k*-means are sensitive to its initial set of centroids. In other words, initial centroids that are not well-aligned with the data structure are likely to drive *k*-means to a sub-optimal clustering.

Many algorithms have been proposed with the intention of providing good initial centroids for *k*-means (see Section II). We performed an extensive side-by-side study comparing the performance of 17 such algorithms by using them to initialise *k*-means on 6,000 synthetic data sets (under different configurations), and 28 real-world data sets. We measured the cluster recovery of each algorithm in two ways: (i) calculating the Adjusted Rand Index (ARI) between each clustering produced by *k*-means on each data set, and the respective true labels—the use of this particular measure means that our results are corrected-for-chance; (ii) calculating the inertia of each clustering. We also investigated computational performance in two ways: (i) we computed the number of iterations *k*-means itself required to converge when initialised with the centroids generated by each of the algorithms we experimented with; (ii) we performed experiments increasing the amount of data to be processed, and analysed the performance of each algorithm.

Our experiments show there is no single algorithm that outperforms all others in all cases. Often algorithms fared better or worse when compared over data sets with different characteristics. Section IV shows which algorithms work better under each data set configuration. For example, whilst in general algorithms showed a decline in ARI when recovering a higher number of clusters, some showed markedly more consistent performance than others. We also found that certain algorithms simply were not suitable for certain data set configurations, for example often finding an incorrect number of clusters (even if feed with the correct $K$), or being unable to produce a suitable clustering where the number of features is lower than the number of clusters. The results we discuss in Section IV are certainly of interest to anybody considering *k*-means for a non-trivial clustering scenario.

With regard to possible future work, we see the topic we deal with here to be ongoing. It is not possible to implement and analyse every single *k*-means initialisation algorithm thus far developed, and there will be more such algorithms published in the future. Also, it would be interesting to perform controlled experiments to investigate the performance of these algorithms on synthetic data sets containing outliers or non-Gaussian clusters, as well as analyse the performance of these algorithms under different numbers of *k*-means repetitions (here fixed at 50). Other potential directions include experiments on data sets with parameters that are even higher to those we use here, or sparse data sets (e.g. textual data).

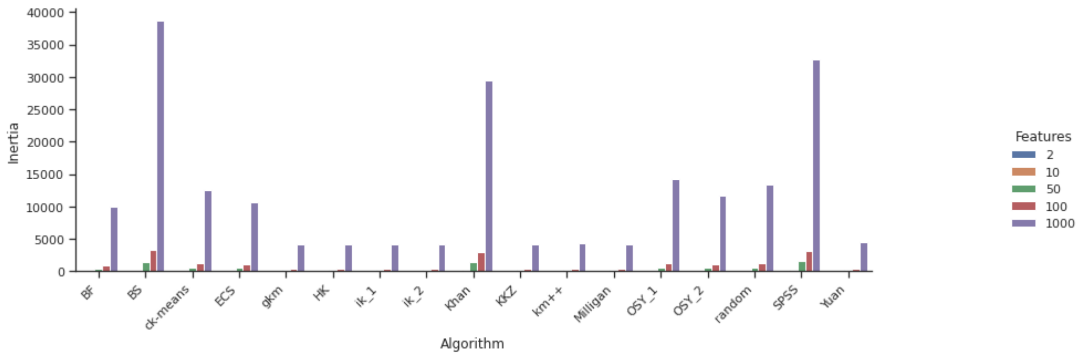

**FIGURE 13.** The average inertia of each algorithm over all synthetic data sets experimented with, broken down by the number of features.

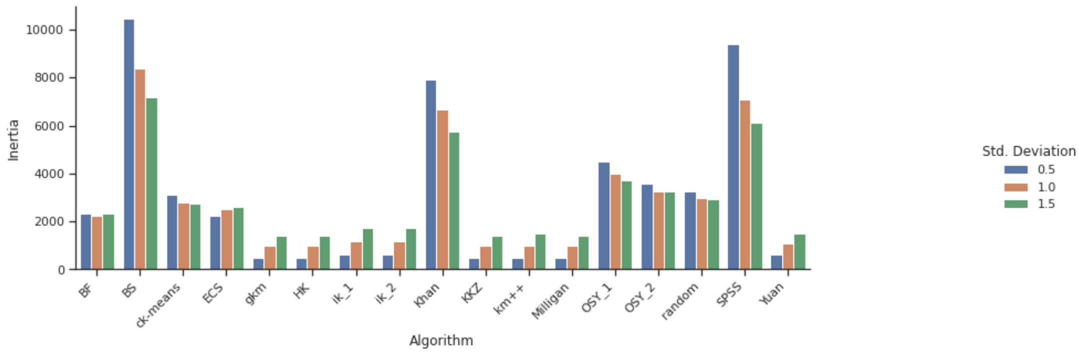

**FIGURE 14.** The average inertia of each algorithm over all synthetic data sets experimented with, broken down by the within-cluster standard deviation.

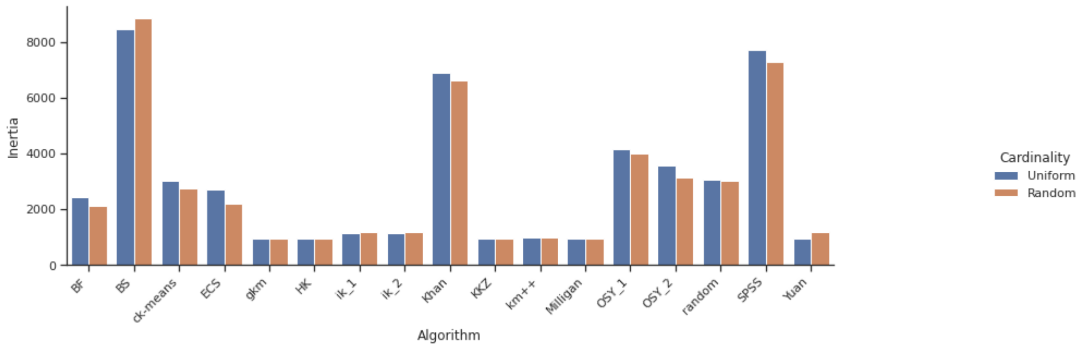

**FIGURE 15.** The average inertia of each algorithm over all synthetic data sets experimented with, broken down by the cardinality of clusters.

## APPENDIX

This section presents the plots for all our experiments measuring inertia on synthetic data sets. For details, see Figures 12, 13, 14, and 15.

## REFERENCES

[1] V. Y. Kiselev, K. Kirschner, M. T. Schaub, T. Andrews, A. Yiu, T. Chandra, K. N. Natarajan, W. Reik, M. Barahona, and A. R. Green, "SC3: Consensus clustering of single-cell RNA-seq data," *Nature Methods*, vol. 14, no. 5, pp. 483–486, 2017.

[2] M. Caron, P. Bojanowski, A. Joulin, and M. Douze, "Deep clustering for unsupervised learning of visual features," in *Proc. Eur. Conf. Comput. Vis. (ECCV)*, 2018, pp. 132–149.

[3] R. C. de Amorim, "Unsupervised feature selection for large data sets," *Pattern Recognit. Lett.*, vol. 128, pp. 183–189, Dec. 2019.

[4] A. Saxena, M. Prasad, A. Gupta, N. Bharill, O. P. Patel, A. Tiwari, M. J. Er, W. Ding, and C.-T. Lin, "A review of clustering techniques and developments," *Neurocomputing*, vol. 267, pp. 664–681, Dec. 2017.

[5] J. C. Bezdek, W. Full, and R. Ehrlich, "FCM: The fuzzy c-means clustering algorithm," *Comput. Geosci.*, vol. 10, nos. 2–3, pp. 191–203, 1984.

[6] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: An overview," *Wiley Interdiscipl. Rev. Data Mining Knowl. Discovery*, vol. 2, no. 1, pp. 86–97, 2012.

[7] H.-P. Kriegel, P. Kröger, J. Sander, and A. Zimek, ''Density-based clustering,'' *Wiley Interdiscipl. Rev., Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 231–240, May/Jun. 2011.

[8] J. MacQueen, ''Some methods for classification and analysis of multivariate observations,'' in *Proc. 5th Berkeley Symp. Math. Statist, Prob.*, Oakland, CA, USA, vol. 1, Jun. 1967, pp. 281–297.

[9] E. W. Forgy, ''Cluster analysis of multivariate data: Efficiency versus interpretability of classifications,'' *Biometrics*, vol. 21, no. 3, pp. 768–769, 1965.

[10] A. K. Jain, ''Data clustering: 50 years beyond K-means,'' *Pattern Recognit. Lett.*, vol. 31, no. 8, pp. 651–666, 2010.

[11] D. Steinley, ''K-means clustering: A half-century synthesis,'' *Brit. J. Math. Stat. Psychol.*, vol. 59, no. 1, pp. 1–34, May 2006.

[12] M. Mahajan, P. Nimbhorkar, and K. Varadarajan, ''The planar k-means problem is NP-hard,'' *Theor. Comput. Sci.*, vol. 442, pp. 13–21, Jul. 2012.

[13] D. J. Hand and W. J. Krzanowski, ''Optimising-means clustering results with standard software packages,'' *Comput. Statist. Data Anal.*, vol. 49, no. 4, pp. 969–973, Jun. 2005.

[14] M. Capo, A. Perez, and J. A. A. Lozano, ''An efficient split-merge re-start for the K-means algorithm,'' *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 4, pp. 1618–1627, Apr. 2020.

[15] P. Fränti and S. Sieranoja, ''How much can k-means be improved by using better initialization and repeats?'' *Pattern Recognit.*, vol. 93, pp. 95–112, Sep. 2019.

[16] J. M. Pena, J. A. Lozano, and P. Larranaga, ''An empirical comparison of four initialization methods for the k-means algorithm,'' *Pattern Recognit. Lett.*, vol. 20, no. 10, pp. 1027–1040, 1999.

[17] J. He, M. Lan, C.-L. Tan, S.-Y. Sung, and H.-B. Low, ''Initialization of cluster refinement algorithms: A review and comparative study,'' in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 1, Jul. 2004, pp. 297–302.

[18] D. Steinley and M. J. Brusco, ''Initializing K-means batch clustering: A critical evaluation of several techniques,'' *J. Classification*, vol. 24, no. 1, pp. 99–121, Jun. 2007.

[19] *MATLAB, Version 7.10.0 (R2010a)*, The MathWorks, Natick, MA, USA, 2010.

[20] *R: A Language and Environment for Statistical Computing*, R Core Team, R Found. Stat. Comput., Vienna, Austria, 2020. [Online]. Available: https://www.R-project.org

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, ''Scikit-learn: Machine learning in Python,'' *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[22] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, ''The weka data mining software: An update,'' *SIGKDD Explor.*, vol. 11, no. 1, pp. 10–18, 2009.

[23] M. E. Celebi, H. A. Kingravi, and P. A. Vela, ''A comparative study of efficient initialization methods for the k-means clustering algorithm,'' *Expert Syst. Appl.*, vol. 40, no. 1, pp. 200–210, Jan. 2013.

[24] V. Faber, ''Clustering and the continuous K-means algorithm,'' *Los Alamos Sci.*, no. 22, p. 67, 1994.

[25] G. W. Milligan, ''An examination of the effect of six types of error perturbation on fifteen clustering algorithms,'' *Psychometrika*, vol. 45, no. 3, pp. 325–342, Sep. 1980.

[26] G. W. Milligan and P. D. Isaac, ''The validation of four ultrametric clustering algorithms,'' *Pattern Recognit.*, vol. 12, no. 2, pp. 41–50, Jan. 1980.

[27] J. H. Ward, ''Hierarchical grouping to optimize an objective function,'' *J. Amer. Statist. Assoc.*, vol. 58, no. 301, pp. 236–244, 1963.

[28] I. Katsavounidis, C.-C. J. Kuo, and Z. Zhang, ''A new initialization technique for generalized Lloyd iteration,'' *IEEE Signal Process. Lett.*, vol. 1, no. 10, pp. 144–146, Oct. 1994.

[29] T. Onoda, M. Sakai, and S. Yamada, ''Careful seeding method based on independent components analysis for K-means clustering,'' *J. Emerg. Technol. Web Intell.*, vol. 4, no. 1, pp. 51–59, Feb. 2012.

[30] S.-Q. Wang and D.-M. Zhu, ''Research on selecting initial points for k-means clustering,'' in *Proc. Int. Conf. Mach. Learn. Cybern.*, Jul. 2008, pp. 91–99.

[31] A. Likas, N. Vlassis, and J. J. Verbeek, ''The global K-means clustering algorithm,'' *Pattern Recognit.*, vol. 36, no. 2, pp. 451–461, Feb. 2003.

[32] F. Yuan, Z.-H. Meng, H.-X. Zhang, and C.-R. Dong, ''A new algorithm to get the initial centroids,'' in *Proc. Int. Conf. Mach. Learn. Cybern.*, 2004, pp. 1191–1193.

[33] B. Mirkin, *Clustering for Data Mining: A Data Recovery Approach*. Boca Raton, FL, USA: CRC Press, 2005.

[34] M. M.-T. Chiang and B. Mirkin, ''Intelligent choice of the number of clusters in K-means clustering: An experimental study with different cluster spreads,'' *J. Classification*, vol. 27, no. 1, pp. 3–40, Mar. 2010.

[35] R. C. de Amorim and C. D. L. Ruiz, ''Identifying meaningful clusters in malware data,'' *Expert Syst. Appl.*, vol. 177, Sep. 2021, Art. no. 114971.

[36] D. Arthur and S. Vassilvitskii, ''K-means++: The advantages of careful seeding,'' in *Proc. 18th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2007, pp. 1027–1035.

[37] K. K. Pavan, A. A. Rao, A. Rao, and G. Sridhar, ''Single pass seed selection algorithm for K-means,'' *J. Comput. Sci.*, vol. 6, no. 1, pp. 60–66, 2010.

[38] M. Erisoglu, N. Calis, and S. Sakallioglu, ''A new algorithm for initial cluster centers in K-means algorithm,'' *Pattern Recognit. Lett.*, vol. 32, no. 14, pp. 1701–1705, 2011.

[39] A. Hatamlou, ''In search of optimal centroids on data clustering using a binary search algorithm,'' *Pattern Recognit. Lett.*, vol. 33, no. 13, pp. 1756–1760, Oct. 2012.

[40] F. Khan, ''An initial seed selection algorithm for K-means clustering of georeferenced data to improve replicability of cluster assignments for mapping application,'' *Appl. Soft Comput.*, vol. 12, no. 11, pp. 3698–3700, Nov. 2012.

[41] A. Hyvärinen and E. Oja, ''Independent component analysis: Algorithms and applications,'' *Neural Netw.*, vol. 13, nos. 4–5, pp. 411–430, Jun. 2000.

[42] A. Hyvärinen, ''Fast and robust fixed-point algorithms for independent component analysis,'' *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 626–634, May 1999.

[43] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml

[44] H. Chernoff, ''The use of faces to represent points in K-dimensional space graphically,'' *J. Amer. Statist. Assoc.*, vol. 68, no. 342, pp. 361–368, 1973.

[45] L. Hubert and P. Arabie, ''Comparing partitions,'' *J. Classification*, vol. 2, no. 1, pp. 193–218, Jan. 1985.

[46] W. M. Rand, ''Objective criteria for the evaluation of clustering methods,'' *J. Amer. Statist. Assoc.*, vol. 66, no. 336, pp. 846–850, 1971.

[47] R. C. D. Amorim, ''An empirical evaluation of different initializations on the number of K-means iterations,'' in *Proc. Mex. Int. Conf. Artif. Intell.* Springer, 2012, pp. 15–26.

[48] F. Murtagh and P. Legendre, ''Ward's hierarchical agglomerative clustering method: Which algorithms implement ward's criterion?'' *J. Classification*, vol. 31, no. 3, pp. 274–295, 2014.