Decision Making: Applications in Management and Engineering
Vol. 6, Issue 2, 2023, pp. 734-746.
ISSN: 2560-6018
eISSN: 2620-0104
DOI: https://doi.org/10.31181/dmame622023742

RANDOM CENTROID INITIALIZATION FOR IMPROVING CENTROID-BASED CLUSTERING

Vadim V. Romanuke [1*]

[1]Polish Naval Academy, 69 Śmidowicza Street, Gdynia, Poland

Received: 3 April 2023;
Accepted: 9 July 2023;
Available online: 4 August 2023.

Original scientific paper

Abstract: *A method for improving centroid-based clustering is suggested. The improvement is built on diversification of the k-means++ initialization. The k-means++ algorithm claimed to be a better version of k-means is tested by a computational set-up, where the dataset size, the number of features, and the number of clusters are varied. The statistics obtained on the testing have shown that, in roughly 50 % of instances to cluster, k-means++ outputs worse results than k-means with random centroid initialization. The impact of the random centroid initialization solidifies as both the dataset size and the number of features increase. In order to reduce the possible underperformance of k-means++, the k-means algorithm is run on a separate processor core in parallel to running the k-means++ algorithm, whereupon the better result is selected. The number of k-means++ algorithm runs is set not less than that of k-means. By incorporating the seeding method of random centroid initialization, the k-means++ algorithm gains about 0.05 % accuracy in every second instance to cluster.*

Key words: *centroid-based clustering, k-means++, centroid initialization, random initialization, algorithm multiple runs.*

1. Initialization in centroid-based clustering

The centroid-based clustering problem is to partition N data points (observations, objects) into k clusters (groups) by minimizing the sum of within-cluster squared Euclidean distances (Gonzalez, 1985; Hartigan & Wong, 1979; Ikotun et al., 2023). Centroid-based clustering, although being a specific field in cluster analysis, has many practical implementations (Ostrovsky et al., 2006; Phillips, 2002; Mahajan, Nimbhorkar, & Varadarajan, 2009). Clustering flat objects (which have two features) is often perceived as a metric facility location problem (Li, 2011; Megiddo & Tamir, 1982). The task of this problem is to find the best warehouse locations to optimally service a given set of consumers whose locations are taken as the data to be clustered, and warehouses are seen as cluster centers (centroids) (MacQueen, 1967;

* Corresponding author
E-mail address: v.romanuke@amw.gdynia.pl (V. Romanuke)

Romanuke, 2018b; Mahajan, Nimbhorkar, & Varadarajan, 2009; Jafar et al., 2021). For instance, centroid-based clustering is used to rationally assign mobiles (consumers) to base stations (that, in a more rigorous manner, are referred to as centroids) of a wireless communication network (Romanuke, 2019). Mounting locations of base stations also can be determined by centroid-based clustering. It is also invoked to build complex models of decision-making (Jafar & Saeed, 2022).

In practice, the fastest method for centroid-based clustering is the *k*-means algorithm that is an efficient heuristic (Lloyd, 1982; Bottou & Bengio, 1994; Hamerly, 2010; Kanungo et al., 2002). The algorithm quickly converges to a local optimum (an approximate minimum), so it is usually run multiple times and the best approximate minimum is selected (Fränti & Sieranoja, 2019; Celebi et al., 2013; Kanungo et al., 2004). The *k*-means problem is solved using either the Lloyd's or Elkan's algorithm, where the latter is more efficient by using the triangle inequality for dense data (Ostrovsky et al., 2006; Vattani, 2011; Press et al., 2007).

While *k*-means chooses *k* initial cluster centroids at random, the *k*-means++ algorithm specifically initializes the centroids. It uses an heuristic to find centroid seeds for *k*-means clustering. According to Arthur & Vassilvitskii (2007), *k*-means++ improves both the running time of the Lloyd's algorithm and the approximate minimum of the sum of within-cluster squared Euclidean distances. Nevertheless, it is easy to convince that the *k*-means++ advantage does not come true for every centroid-based clustering problem. The advantage works on average. For example, at some random state (determining pseudorandom number generation), a dataset of 2500 points scattered uniformly within a unit square with adding a half of standard normal noise is partitioned into 24 clusters more accurately by initializing 24 centroids at random, where 50 multiples run are used. Indeed, an approximate minimum of 119.0397 is obtained by the random initialization within 677.9 milliseconds on a dual-core processor Intel Core i5-7200U@2.50GHz, whereas the *k*-means++ algorithm takes about 777.8 milliseconds to reach just an approximate minimum of 120.947 (see Figure 1, where the centroids are marked with circles). Thus, in this particular counterexample, *k*-means++ is 1.577% less accurate and 12.84 % slower. This is a quite huge loss in both accuracy and computational speed.

Therefore, the random centroid initialization (which essentially is the *k*-means algorithm) in centroid-based clustering can significantly outperform *k*-means++. But what is the random initialization performance on average? How does it relate to the *k*-means++ performance, i. e. what is the relationship between these two approaches? These questions are still open and need to be answered in order to ascertain a realistic disadvantage of "careful seeding".

2. Motivation and goal

An average advantage in performance has a general problem (that might be punned as a disadvantage) — it does not guarantee that the advantage happens in every single instance (Chakrabarty & Swagatam, 2022; Arthur & Vassilvitskii, 2006). The *k*-means++ algorithm has been believed to have the outperformance with respect to other approaches to centroid-based clustering including the *k*-means with the random centroid initialization. The *k*-means++ underperformance in the particular counterexample in Figure 1 may be just an occasional outlier instance, but it nonetheless pushes to a suggestion that such outliers do exist. Does the pseudorandom number generator state influence it? Maybe the counterexample is just an occurrence appeared at 50 algorithm runs, and it disappears at other numbers of runs? As a matter of fact, it does not.

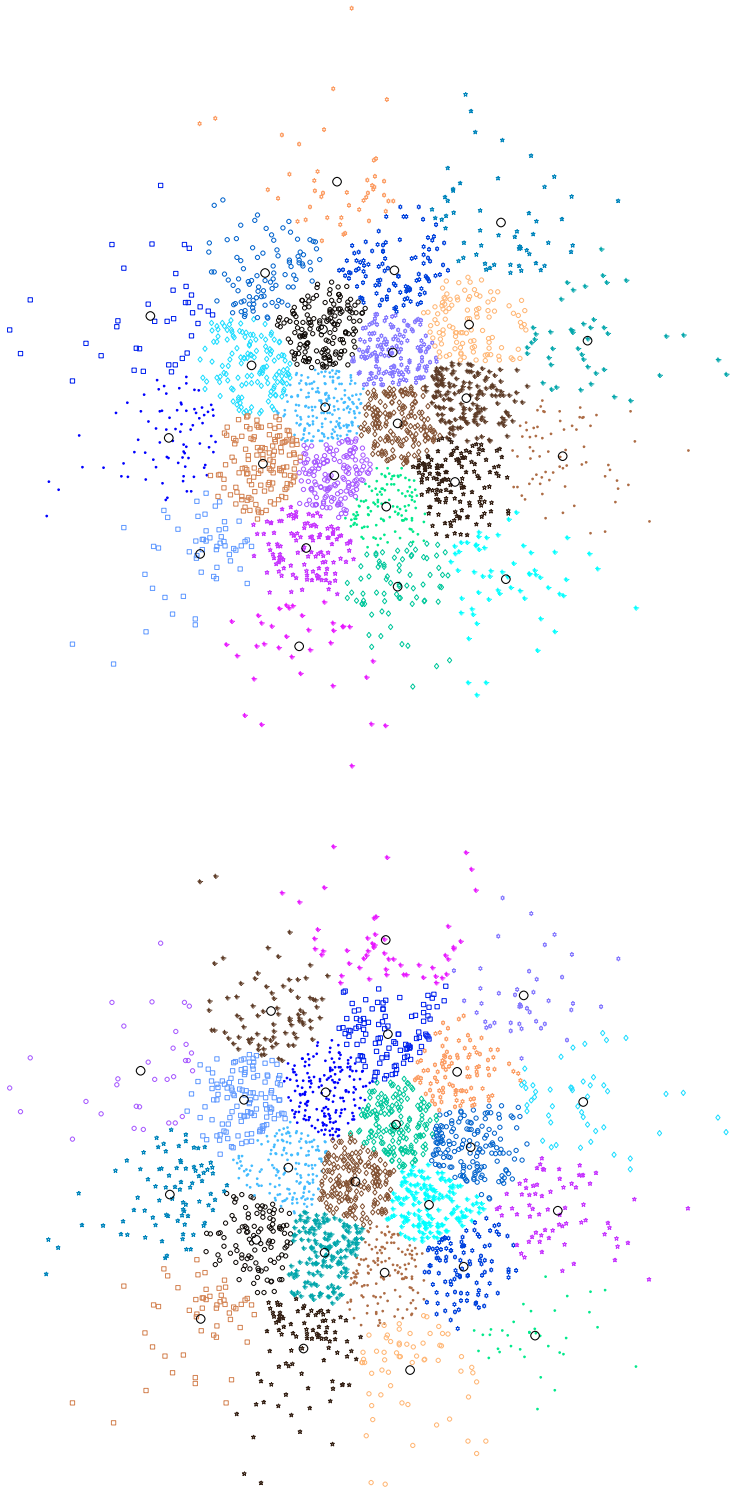

Figure 1. The particular counterexample, in which centroid-based clustering is done by 1.577% more accurately and 12.84 % faster by using the random initialization (the top subplot) than by using *k*-means++ (the bottom subplot)

Figure 2 presents a plot of how the sum of within-cluster squared Euclidean distances changes versus the number of algorithms runs for the particular pseudorandom number generator state. It is clearly seen that as the algorithm is run more times, *k*-means++ continues underperforming, finally reaching an approximate minimum of 119.2911 after 78 runs. Meanwhile, the randomly initialized centroids here make the sum of within-cluster squared Euclidean distances equal to 120.2943 in the very beginning. It is even better than 120.947 — the *k*-means++ performance after 50 runs (Figure 1). An approximate minimum of 118.984 is reached after 56 runs followed the leap-down from the preceding value of 119.0397 related to Figure 1.

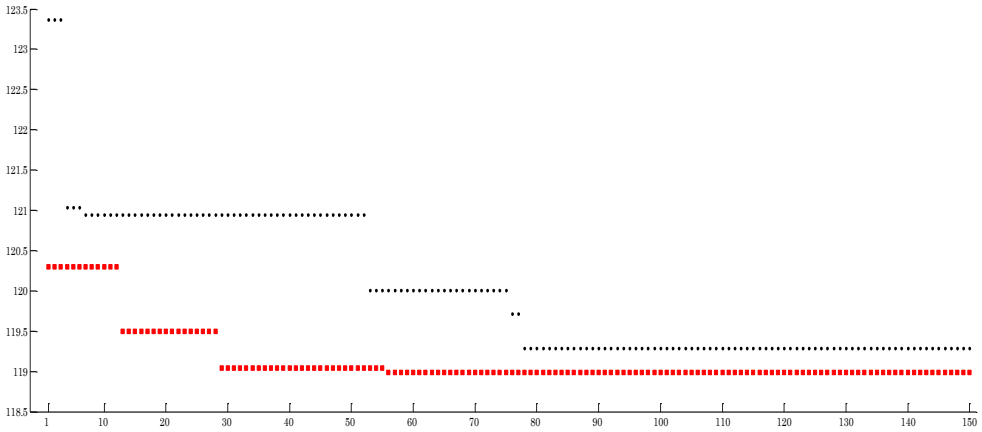

Figure 2. The sum of within-cluster squared Euclidean distances versus the number of algorithm runs for *k*-means++ (black dots) and random (red dotted squares) centroid initialization

By starting with some different pseudorandom number generator state, the same dataset is clustered differently. Unlike the previous state performance visualized in Figure 2, this time the random centroid initialization starts worse than *k*-means++ (Figure 3), but eventually reaches an approximate minimum of 118.8904 after 146 runs. Despite a better start lasting until 69 runs, *k*-means++ still underperforms reaching only an approximate minimum of 119.5583 after 38 runs (it is worse than that in Figure 2). Therefore, this confirms that the starting relationship between the performances can be broken as the algorithm is run more times.

Issuing from the possible underperformance of *k*-means++ in centroid-based clustering, the goal is to suggest a method of how the underperformance could be reduced by using the random centroid initialization. For meeting the goal, the following five tasks are to be accomplished:

1. To formalize a computational set-up to gather statistics of the performance of the *k*-means++ algorithm versus *k*-means with random centroid initialization.

2. To study the statistics and suggest a method of how the random centroid initialization could improve centroid-based clustering to reduce the possible underperformance of *k*-means++.

3. To show the suggested method performance versus *k*-means++.

4. To discuss practical applicability and scientific significance of the suggested method for centroid-based clustering.

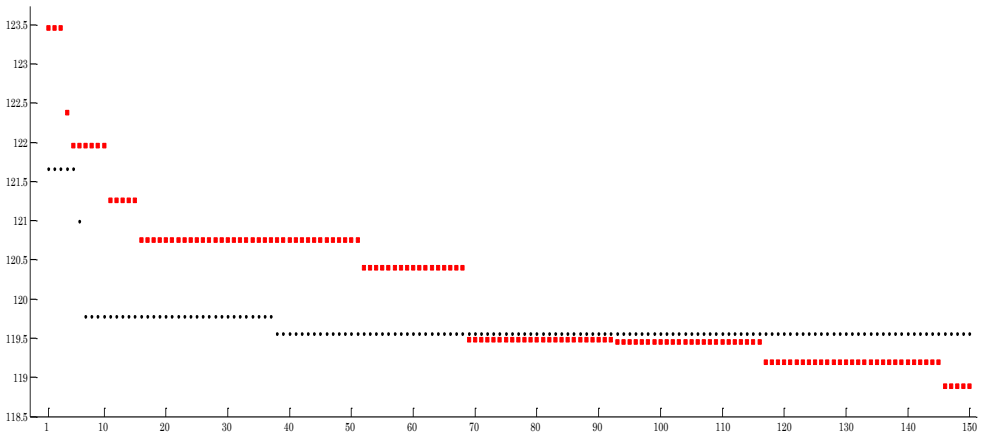

Figure 3. The performance by a pseudorandom number generator state different from that in Figure 2, where *k*-means++ still underperforms despite a better start

5. To make an appropriate conclusion and an outlook of how the research might be developed further.

3. Computational set-up

To gather statistics of the performance of the *k*-means++ algorithm versus *k*-means with random centroid initialization, a computational simulation is set up as follows. The number of algorithms runs denoted by A_{runs} is 200. The maximal number of iterations (for every run) is set at 300. The number of the dataset points to be clustered denoted by N is selected from a range

$$N_{\text{Range}} = \{10^3, 10^4, 2.5\cdot 10^4, 5\cdot 10^4, 10^5, 2\cdot 10^5\}. \tag{1}$$

The number of clusters denoted by k is selected from a range

$$K_{\text{Range}} = \{4, 8, 16, 32, 64\}. \tag{2}$$

The number of object features denoted by m is selected from a range

$$M_{\text{Range}} = \{2, 3, 4, 5\}. \tag{3}$$

For every $\{N, m\}$ the dataset is generated as points

$$\{\mathbf{X}_i\}_{i=1}^{N} = \{[x_{li}]_{1\times m}\}_{i=1}^{N}, \tag{4}$$

where the value of feature l is

$$x_{li} = \theta_{li} + 0.5\cdot\xi_{li} \text{ by } l = \overline{1, m} \text{ and } i = \overline{1, N} \tag{5}$$

for a value θ_{li} of a random variable Θ_{li} distributed uniformly on the open interval $(0;1)$, and a value ξ_{li} of a random variable Ξ_{li} distributed normally with the zero mean and unit variance. The dataset is partitioned into k clusters, $k \in K_{\text{Range}}$. The performance includes both accuracy and computational (running) time. The accuracy is meant by the sum of within-cluster squared Euclidean distances: for centroids

$$\{\mathbf{c}_j\}_{j=1}^{k} = \{[c_{lj}]_{1\times m}\}_{j=1}^{k} \tag{6}$$

belonging to respective clusters $\{S_j\}_{j=1}^{k}$ the sum is

$$D = \sum_{j=1}^{k}\sum_{\mathbf{x}_i\in S_j}\sum_{l=1}^{m}(x_{li}-c_{li})^2 . \tag{7}$$

Denote sum (7), which can be referred to as a score, and the respective computational time for *k*-means and *k*-means++ respectively by D_{rand}, t_{rand}, and D_{++}, t_{++}. The relative tolerance regarding sum (7) is set at 0.0001 to declare convergence.

The relative difference between the *k*-means and *k*-means++ performances for number A_{runs} is calculated as a percentage

$$p(A_{\text{runs}}) = 100\cdot\frac{D_{++}-D_{\text{rand}}}{D_{++}} . \tag{8}$$

The relative difference between the *k*-means and *k*-means++ running times for number A_{runs} is calculated as a percentage

$$\tau(A_{\text{runs}}) = 100\cdot\frac{t_{++}-t_{\text{rand}}}{t_{++}} . \tag{9}$$

Meanwhile, both the algorithms can be run (in any order, but not in parallel) for fewer runs A^*_{runs}, whereupon the minimum

$$D^*(A^*_{\text{runs}}) = \min\{D^*_{\text{rand}}, D^*_{++}\} \tag{10}$$

is found for the respective solution (herein, both the algorithms are used in a mix), where D^*_{rand} and D^*_{++} are the respective "internal" values of sum (7). The relative difference percentage

$$p^*(A_{\text{runs}}, A^*_{\text{runs}}) = 100\cdot\frac{D_{++}-D^*(A^*_{\text{runs}})}{D_{++}} \tag{11}$$

is calculated to see how much minimum (10) improves the solution with respect to the *k*-means++ solution. In this set-up, $A^*_{\text{runs}} = 80$ and thus the mix algorithm is of 160 runs. This is done intentionally not to increase its running time with respect to *k*-means++. The relative difference between the mix algorithm and *k*-means++ running times is calculated as a percentage

$$\tau^*(A_{\text{runs}}, A^*_{\text{runs}}) = 100\cdot\frac{t_{++}-t^*(A^*_{\text{runs}})}{t_{++}} , \tag{12}$$

where $t^*(A^*_{\text{runs}})$ is the computational time taken to find (10) by A^*_{runs} runs for each of *k*-means and *k*-means++.

4. Statistics of the performance

As every instance to cluster has been generated for a triple $\{N, m, k\}$ by having repeated it for five times, there are 10 $6\times4\times5\times5$ arrays of the following data by

(8) — (12): D_{rand}, t_{rand}, D_{++}, t_{++}, $D^*(80)$, $t^*(80)$, $p(200)$, $\tau(200)$, $p^*(200,80)$, $\tau^*(200,80)$. These arrays are averaged over the last dimension. Then the resulting $6\times4\times5$ averages can be averaged over the second dimension getting rid of the number of features. So, the relative difference between the *k*-means and *k*-means++ performances by (8) becomes a 6×5 matrix (Table 1). So do the relative difference between the *k*-means and *k*-means++ running times by (9) presented in Table 2, the relative difference between the mix algorithm and *k*-means++ performances by (11) presented in Table 3, and the relative difference between the mix algorithm and *k*-means++ running times by (12) presented in Table 4. Positive percentage values highlighted bold in Tables 1 — 4 imply that the *k*-means++ algorithm is worse. As the number of points increases, *k*-means may perform even better, and the difference becomes more solid (Table 1). However, there is no solid advantage in its running time (Table 2). The only exception is the dataset of 1000 points, for which *k*-means performs far faster as the number of clusters is increased. On average, *k*-means is 0.1248 % less accurate and 5.5815 % faster than *k*-means++. As the number of features increases from 2 up to 5, the accuracy loss drops from 0.2897 down to 0.051 %, whereas the speedup grows from 1.9388 up to 6.536 %.

Table 1. The averaged relative difference $p(200)$

		k				
		4	8	16	32	64
N, thousand points	1	**0.0038**	–0.0332	**0.0499**	–0.5858	–2.9195
	10	**0.0028**	–0.0029	**0.0194**	–0.0093	–0.1851
	25	–0.0032	–0.0044	**0.0018**	–0.0084	–0.071
	50	–0.0017	**0.0003**	–0.0136	–0.011	–0.0422
	100	**0.003**	**0.002**	**0.0023**	**0.005**	**0.0301**
	200	–0.0026	**0.0089**	0	–0.012	**0.0326**

Table 2. The averaged relative difference $\tau(200)$

		k				
		4	8	16	32	64
N, thousand points	1	**15.9466**	**20.2476**	**26.0419**	**40.4715**	**56.623**
	10	–11.7908	–5.8843	–2.2911	–0.8097	–2.1711
	25	–10.2717	–2.9196	**0.6984**	**2.4742**	–2.7903
	50	–1.514	**6.0402**	**6.5657**	**2.6504**	–4.1854
	100	**4.5527**	**6.7761**	**5.5678**	**1.7482**	–3.7013
	200	**5.9929**	**7.0517**	**6.3528**	**2.5478**	–2.574

On average, the mix does not outperform *k*-means++ (Table 3) by losing about 0.0265 % in accuracy. However, the loss drops down to 0.0037 % as the dataset becomes larger (for instance, it is clearly seen by the column for 64 clusters).

Moreover, the mix is 22.2059 % faster (see Table 4, in which only cells are highlighted bold which correspond to those in Table 3).

Table 3. The averaged relative difference $p^*(200, 80)$

		k				
		4	8	16	32	64
N, thousand points	1	–0.0018	–0.0339	**0.0444**	–0.0898	–0.4169
	10	–0.0015	–0.0119	–0.0037	–0.0171	–0.0753
	25	–0.0057	–0.004	–0.0031	–0.0198	–0.0386
	50	–0.009	**0.001**	–0.0148	–0.0131	–0.0432
	100	**0.0007**	**0.0011**	–0.0042	–0.0073	–0.0097
	200	–0.0062	–0.0005	–0.0003	–0.0161	**0.0048**

Table 4. The averaged relative difference $\tau^*(200, 80)$

		k				
		4	8	16	32	64
N, thousand points	1	26.5908	27.7943	**29.0979**	35.8353	41.6295
	10	15.5539	17.9377	18.7276	18.7803	19.0342
	25	15.4327	18.8305	20.167	21.3691	19.0413
	50	21.136	**22.0985**	23.0131	21.1487	18.1065
	100	**22.4393**	**22.2942**	22.8851	21.0932	18.6342
	200	22.458	22.5046	22.7475	20.8289	**18.9677**

So, the main inference from the statistics is that, as both the number of points and the number of features increase, the impact of the random centroid initialization solidifies. As the number of clusters is increased, no such or other distinct pattern is observed. However, there is another distinct property of the statistics that is as solid as the mentioned inference is. This is about the rate of instances for which *k*-means performs better than *k*-means++ (based on the data for Table 1). In fact, about a half of 600 instances (this is the total number of instances) are clustered more accurately by *k*-means (Table 5). The dependence on the number of features, on the dataset size, and on the number of clusters is hardly perceivable. Nearly the same quality of the performance is seen for the mix algorithm (Table 6), where the tie on the same performance is broken by the shorter running time of the mix algorithm.

On average, the *k*-means performance is better, the same, and worse in 49.8333 %, 0.5 %, and 49.6667 % of all instances generated. The respective averaged rates of the mix performance are 36.6667 %, 13 %, and 50.3333 %. Obviously, the mix algorithm configured for $A^*_{\text{runs}} = 80$ by 200 runs of *k*-means++ is always faster (Table 4). Meanwhile, *k*-means does not always outrun *k*-means++ (Table 7), which is also inferred from Table 2. An exception occurs when the dataset is of a medium to small size, although there are two cells in Table 7 for 10000-point and 25000-point datasets, where *k*-means++ is likelier to be faster. As the dataset size increases,

k-means++ becomes slower, and the slowness builds up. On average, *k*-means is faster at 68.3333 % rate.

Table 5. Percentage of instances on which the *k*-means performance is better or worse

By the number of features				By the dataset size (in thousands of points)				By the number of clusters			
m	better	equal	worse	Size	better	equal	worse	*k*	better	equal	worse
2	47.3333	2	50.6667	1	34	3	63	4	55	1.6667	43.3333
3	55.3333	0	44.6667	10	51	0	49	8	53.333	0.833	45.833
4	46	0	54	25	45	0	55	16	55.8333	0	44.1667
5	50.6667	0	49.3333	50	47	0	53	32	45.8333	0	54.1667
				100	63	0	37	64	39.1667	0	60.8333
				200	59	0	41				

Table 6. Percentage of instances on which the mix performance is better or worse

By the number of features				By the dataset size (in thousands of points)				By the number of clusters			
m	better	equal	worse	Size	better	equal	worse	*k*	better	equal	worse
2	42	8.6667	49.3333	1	27	22	51	4	35.833	15.833	48.333
3	42.667	13.333	44	10	44	10	46	8	39.167	14.167	46.667
4	34.6667	12	53.3333	25	32	11	57	16	35.833	14.167	50
5	27.3333	18	54.6667	50	32	11	57	32	39.1667	12.5	48.3333
				100	44	14	42	64	33.333	8.333	58.333
				200	41	10	49				

The obtained data remain nearly the same if the computational simulation is repeated (starting from different pseudorandom number generator states). In general, the abovementioned main inference from the statistics remains the same. Moreover, the advantage of the mix algorithm solidifies further if the number of *k*-means and *k*-means++ runs (inside this algorithm) is increased from 80 up to 115.

Table 7. Percentage of instances on which *k*-means performs faster or slower

By the number of features				By the dataset size (in thousands of points)				By the number of clusters			
m	faster	equal	slower	Size	faster	equal	slower	*k*	faster	equal	slower
2	53.3333	0	46.6667	1	100	0	0	4	55	0	45
3	73.3333	0	26.6667	10	43	0	57	8	73.3333	0	26.6667
4	71.3333	0	28.6667	25	42	0	58	16	80.8333	0	19.1667
5	75.3333	0	24.6667	50	69	0	31	32	73.3333	0	26.6667
				100	76	0	24	64	59.1667	0	40.8333
				200	80	0	20				

5. The improved performance

Whichever number A_{runs} is, number A^*_{runs} cannot be selected such that *k*-means would outperform *k*-means++. Nevertheless, by ignoring the seeding method of random centroid initialization, *k*-means++ loses about 0.05 % accuracy in every second instance to cluster. All the more that much higher accuracy losses are likely. For instance, a 1000-point dataset with 4 features has been partitioned into 16 clusters by *k*-means++ with a score of 417.222, whereas *k*-means has performed on the dataset with a score of 414.5155 (that is 0.6487 % more accurate). At the same time, *k*-means may perform too poorly on smaller-sized problems. Thus, another 1000-point dataset with 2 features has been partitioned into 64 clusters by *k*-means++ with a score of 13.6557, whereas the *k*-means score is 14.8535 (that is 8.7712 % less accurate).

To prevent potential accuracy losses, the random centroid initialization is incorporated into *k*-means++. Inasmuch as *k*-means cannot be used standalone, the mix algorithm is a simple solution. However, minimum (10) still can occasionally exceed D_{++} if number A^*_{runs} is a fraction of A_{runs} (i. e., when $A^*_{\text{runs}} < A_{\text{runs}}$). If both *k*-means and *k*-means++ are initiated by the same pseudorandom number generator state, then

$$D^*\left(A^*_{\text{runs}}\right) \leqslant D_{++} \text{ by } A^*_{\text{runs}} = A_{\text{runs}}. \tag{13}$$

If not, then inequality (13) does not always hold. Meanwhile, it is not about the competition between the two approaches. It is rather about the diversity of scores among which the minimum should be selected. It is similar to the hybridization and fuzzification used for decision-making in multi-attributes and multi-objective problems (Jafar et al., 2022; Jafar et al., 2020a; Romanuke, 2018a). The best solution is to run both *k*-means and *k*-means++ on parallel cores, whichever number A^*_{runs} is, whereupon to conclude on minimization operation (10). Besides, inasmuch as *k*-means++ is more robust, it can be run for more times than *k*-means. For example, in overall 160 runs, *k*-means is run for 40 times and *k*-means++ is run for 120 times, where four processor cores may be used to run the algorithms in parallel. This improves the performance in every second instance to cluster.

6. Discussion and conclusion

In nearly a half of instances to cluster, using *k*-means is seemingly redundant, but the other half really needs the random centroid initialization for improving centroid-based clustering. It is worth remembering that the clustering result score is a value of a random variable, so the improved performance can be guaranteed only as an expectance. Such an improvement makes sense for applications in management and engineering featuring repeatability of decision-making events (Romanuke, 2018a; Fränti & Sieranoja, 2019; Romanuke, 2021). Non-repeatable decision-making problems will nonetheless benefit from the improved centroid-based clustering while, e. g., subproblems of measuring similarity are solved (Jafar et al., 2020b; Jafar et al., 2021; Romanuke, 2018b; Romanuke, 2019).

After all, why does *k*-means++ underperform? The answer is quite simple: the *k*-means++ sometimes underperforms because the first centroid is chosen at random, so it is likely that a poor choice of the first centroid leads to poorer accuracy.

Obviously, equal running times of *k*-means and *k*-means++ are very unlikely, but the occurrence when *k*-means++ converges faster than *k*-means (by the same

number of the algorithm runs and the same pseudorandom number generator state) is not excluded. Due to the *k*-means++ centroid initialization is a sequential process, the *k*-means++ slowdown deepens as the problem size increases.

Based on a series of computational simulations, it must be concluded that, by incorporating the seeding method of random centroid initialization, the *k*-means++ algorithm gains about 0.05 % accuracy in every second instance to cluster. In its best non-sequential way to maintain the same running time, the incorporation is practically done by running the *k*-means algorithm on a separate processor core in parallel to running the *k*-means++ algorithm, whereupon the better result is selected. The impact of the random centroid initialization solidifies as both the dataset size and the number of features increase. The approach can be developed further by substituting the sequential process of centroid initialization in the *k*-means++ algorithm with the random centroid initialization as an alternative choice of the first centroid.

Author Contributions: Conceptualization, methodology, validation, formal analysis, writing—original draft preparation, writing—review and editing, visualization, V.R.

Funding: This research received no external funding.

Acknowledgments: This work was technically supported by the Faculty of Mechanical and Electrical Engineering, Polish Naval Academy, Poland.

Conflicts of Interest: The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Arthur, D., & Vassilvitskii, S. (2006). How Slow is the *k*-means Method?, in: Proceedings of the Twenty-Second Annual Symposium on Computational Geometry (SCG'06), pp. 144–153.

Arthur, D., & Vassilvitskii, S. (2007). *K*-means++: The Advantages of Careful Seeding, in: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '07), pp. 1027–1035.

Bottou, L., & Bengio, Y. (1994). Convergence properties of the *K*-means algorithms, in: Proceedings of the 7th International Conference on Neural Information Processing Systems (NIPS'94), pp. 585–592.

Celebi, M. E., Kingravi, H. A., & Vela, P. A. (2013). A comparative study of efficient initialization methods for the *k*-means clustering algorithm. Expert Systems with Applications, 40 (1), 200–210.

Chakrabarty, A., & Swagatam, D. (2022). On strong consistency of kernel *k*-means: A Rademacher complexity approach. Statistics & Probability Letters, 182, Article ID 109291.

Fränti, P., & Sieranoja, S. (2019). How much can *k*-means be improved by using better initialization and repeats? Pattern Recognition, 93, 95–112.

Gonzalez, T. (1985). Clustering to minimize the maximum intercluster distance. Theoretical Computer Science, 38, 293–306.

Hamerly, G. (2010). Making *k*-means even faster, in: Proceedings of the 2010 SIAM International Conference on Data Mining, pp. 130–140.

Hartigan, J. A., & Wong, M. A. (1979). Algorithm AS 136: A *k*-Means Clustering Algorithm. Journal of the Royal Statistical Society. Series C, 28 (1), 100–108.

Ikotun, A. M., Ezugwu, A. E., Abualigah, L., Abuhaija, B., & Heming, J. (2023). *K*-means clustering algorithms: A comprehensive review, variants analysis, and advances in the era of big data. Information Sciences, 622, 178–210.

Jafar, M. N., Khan, M. R., Sultan, H., & Ahmed, N. (2020a). Interval valued fuzzy soft sets and algorithm of IVFSS applied to the risk analysis of prostate cancer. International Journal of Computer Applications, 177 (38), 18–26.

Jafar, M. N., Farooq, A., Javed, K., & Nawaz, N. (2020b). Similarity measures of tangent, cotangent and cosines in neutrosophic environment and their application in selection of academic programs. International Journal of Computer Applications, 177 (46), 17–24.

Jafar, M. N., Saeed, M., Saqlain, M., & Yang, M.-S. (2021). Trigonometric similarity measures for neutrosophic hypersoft sets with application to renewable energy source selection. IEEE Access, 9, 129178–129187.

Jafar, M. N., Saeed, M. H., Khan, K. M., Alamri, F. S., & Khalifa, H. A. E. W. (2022). Distance and similarity measures using max-min operators of neutrosophic hypersoft sets with application in site selection for solid waste management systems. IEEE Access, 10.1109/ACCESS.2022.3144306.

Jafar, M. N., & Saeed, M. (2022). Matrix theory for neutrosophic hypersoft set and applications in multiattributive multicriteria decision-making problems. Journal of Mathematics, 2022, Article ID 6666408.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., & Wu, A. Y. (2002). An efficient *k*-means clustering algorithm: Analysis and implementation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24 (7), 881–892.

Kanungo, T., Mount, D., Netanyahu, N., Piatko, C., Silverman, R., & Wu, A. (2004). A local search approximation algorithm for *k*-means clustering. Computational Geometry: Theory and Applications, 28 (2–3), 89–112.

Li, S. (2011). A 1.488 Approximation Algorithm for the Uncapacitated Facility Location Problem, in: Automata, Languages and Programming. Lecture Notes in Computer Science, vol. 6756. Springer, pp. 77–88.

Lloyd, S. P. (1982). Least squares quantization in PCM. IEEE Transactions on Information Theory, 28, 129–137.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations, in: Proceedings of the Fifth Berkeley Symposium on Mathematics, Statistics and Probability, vol. 1, pp. 281–296.

Mahajan, M., Nimbhorkar, P., & Varadarajan, K. (2009). The Planar *k*-Means Problem is NP-Hard, in: Lecture Notes in Computer Science, vol. 5431. Springer, pp. 274–285.

Megiddo, N., & Tamir, A. (1982). On the complexity of locating linear facilities in the plane. Operations Research Letters, 1 (5), 194–197.

Ostrovsky, R., Rabani, Y., Schulman, L. J., & Swamy, C. (2006). The effectiveness of Lloyd-type methods for the *k*-means problem, in: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06), pp. 165–174.

Phillips, S. J. (2002). Acceleration of *K*-Means and Related Clustering Algorithms, in: Mount, D. M., & Stein, C. (eds.), Lecture Notes in Computer Science, vol. 2409. Springer, pp. 166–177.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). Section 16.1. Gaussian Mixture Models and *k*-Means Clustering, in: Numerical Recipes: The Art of Scientific Computing (3rd edition). Cambridge, New York, NY: Cambridge University Press.

Romanuke, V. V. (2018a). Decision making criteria hybridization for finding optimal decisions' subset regarding changes of the decision function. Journal of Uncertain Systems, 12 (4), 279–291.

Romanuke, V. V. (2018b). Optimization of a dataset for a machine learning task by clustering and selecting closest-to-the-centroid objects. Herald of Khmelnytskyi national university. Technical sciences, 6 (1), 263–265.

Romanuke, V. V. (2019). Fast-and-smoother uplink power control algorithm based on distance ratios for wireless data transfer systems. Studies in Informatics and Control, 28 (2), 147–156.

Romanuke, V. V. (2021). Refinement of acyclic-and-asymmetric payoff aggregates of pure strategy efficient Nash equilibria in finite noncooperative games by maximultimin and superoptimality. Decision Making: Applications in Management and Engineering, 4 (2), 178–199.

Vattani, A. (2011). *k*-means requires exponentially many iterations even in the plane. Discrete and Computational Geometry, 45 (4), 596–616.