algorithms

MDPI

Article

Genetic Elitist Approach and Density Peaks to Improve K-Means Clustering

Libero Nigro [1,*], Franco Cicirelli [2] and Francesco Pupo [1]

[1] DIMES, University of Calabria, 87036 Rende, Italy; f.pupo@unical.it
[2] Institute for High Performance Computing and Networking (ICAR), CNR—National Research Council of Italy, 87036 Rende, Italy; f.cicirelli@icar.cnr.it
* Correspondence: libero.nigro@unical.it

Abstract

K-Means is a well-known algorithm for unsupervised clustering, very often used due to its simplicity and efficiency. Its long-time widespread use has stimulated researchers to investigate its properties further. A critical property concerns K-Means's strong dependence on the seeding method adopted to initialize centroids. Poor initialization causes K-Means to get stuck in a local sub-optimal solution. This paper proposes DPCCs—Density Peaks of Candidate Centroids—a novel seeding method for K-Means. DPCC rests on genetic concepts and density peaks to define an initialization solution close to the optimal one. First, a population of J elitist candidate solutions, that is, solutions capable of yielding a reduced clustering cost, is built. Although none of these particular solutions can be near the optimal one, candidate centroids, as experimentally confirmed, tend to thicken around ground truth centroids. Therefore, subsequent generations of the population are created by repeating the k-nearest neighbors (kNNs) procedure for different values of the k parameter, and estimating density through the reverse nearest neighbors (RNNs) relationship of each centroid. Centroid density peaks are then exploited to rearrange the population solutions toward extracting a candidate solution, which is finally optimized by K-Means. The paper describes the design and operation of DPCC, which is currently implemented in parallel Java. The clustering effectiveness of DPCC is demonstrated by applications to both benchmark and real-world datasets. Results are compared with those of other competing algorithms.

Keywords: unsupervised clustering; K-Means; seeding methods; genetic clustering; density peaks; k-nearest neighbors; reverse nearest neighbors; benchmark datasets; real-world datasets; Java

Academic Editor: Bogdan Dumitrescu

Received: 13 January 2026
Revised: 30 January 2026
Accepted: 2 February 2026
Published: 5 February 2026

1. Introduction

K-Means [1–4] is a standard algorithm for unsupervised clustering [5,6]. Its popularity stems from its simplicity and efficiency. K-Means aims to partition the samples $x_i \in R^D$, each assumed to have D numerical features, of a dataset $X = \{x_i\}_{i=1}^{N}$, into K groups (clusters) $\{C_j\}_{j=1}^{K}$, in such a way that data points assigned to the same cluster are similar to one another, and data points belonging to different clusters are dissimilar. Its centroid point μ_j represents each cluster C_j. More specifically, K-Means partitioning is regulated by

its goal of minimizing the *Sum-of-Squared-Errors* (SSE) objective function, a sort of internal cluster point distribution variance:

$$SSE = \sum_{j=1}^{K} \sum_{x_i \in X \wedge x_i \in C_j} d(x_i, \mu_j)^2 \quad (1)$$

where $d(x_i, \mu_j)$ is the Euclidean distance between x_i and the centroid μ_j of its belonging cluster. The Euclidean distance acts as the similarity metric. Other functions could be used as well. K-Means is a heuristic, approximate algorithm, because doing optimal clustering by minimizing the SSE is a well-known NP-hard problem [7,8], even when using $K = 2$ clusters.

The widespread use of K-Means has fostered a deep understanding of its behavior and crucial properties. One further reason to prefer K-Means over other clustering algorithms in practice is that its main properties are well-studied and well-understood. K-Means has a strong dependency on the initialization procedure (*seeding method*) of the centroids [9], which are subsequently iteratively refined.

A bad initialization of the centroids causes K-Means to get stuck in a local sub-optimal solution. Centroids should be naturally selected in dense areas of the data space. They would never coincide with outliers or noise points, nor should they be close to one another. Selecting multiple centroids within the same cluster area results in splitting a real cluster into multiple small, unnatural clusters, thereby degrading SSE clustering accuracy. Moreover, centroids in excess in a given area of the data space can be difficult to move (see [9]) to areas where some centroids are missing, especially when the two kinds of areas are neatly separated by intermediate and well-separated clusters. The centroids' movement is instead favored by cluster overlapping situations. All these problems are ignored by the seeding method adopted by standard Lloyd's K-Means [2], where centroids are initialized by simply choosing, by uniform random, K points in the dataset. All of this explains why, in the attempt to improve the clustering quality, K-Means is purposely repeated a certain number of times (Repeated K-Means, RKM), each time with an independent initialization of centroids, and by taking as a "solution" the one that emerges in a repetition and best reduces the SSE.

Several seeding methods, both stochastic and deterministic, have been proposed (see, e.g., [10]), to address the aforementioned problems in centroid initialization. Careful seeding methods (see later in this paper) can generate, albeit at an additional computational cost, a centroids initialization that reduces SSE.

This paper proposes a novel initialization method named Density Peaks of Candidate Centroids (DPCCs), which builds on concepts of genetic algorithms [11,12] and density peaks [13–16]. Similar concepts are exploited, e.g., in the evolutionary Recombinator-K-Means [17] that starts with a population equal to the entire dataset. Subsequent generations, which recombine candidate centroids, are iteratively created. Each generation is achieved by creating J solutions through the careful Greedy-K-Means++ [17] (*GKM*++, see also later in this paper) seeding method, applied to the population. Solutions are refined by a small number of Lloyd's K-Means iterations. The best J solutions (in terms of minimal SSE cost) among previous and new generations are retained to define the new population. The approach tends to collapse toward a single solution of minimal SSE. Population-based K-Means (PB-KM) [18] builds a population of J candidate solutions/centroids defined by careful seeding, e.g., *GKM*++. Candidate centroids are then recombined by applying careful seeding directly to the population. Each new centroid configuration is refined by Lloyd's K-Means. The approach is capable of delivering a good solution after a limited number of iterations.

DPCC distinguishes itself from similar approaches by detecting Density Peaks of Candidate Centroids, which are unique for approaching a solution close to the optimal one. As in Ref. [11], the first step of DPCC consists of building a population $\wp$ with a number J of centroid configurations (solutions with K candidate centroids). Unlike [11], candidate centroids are established using careful seeding methods. Therefore, the J solutions are intrinsically elitist. After that, the population of $J * K$ centroids is viewed as a particular dataset. As experimentally confirmed, candidate solutions have, in general, a small chance of being close to the optimal ground truth solution. Despite this, though, candidate centroids tend to thicken around ground truth centroids. In its second step, DPCC applies the concepts of density peaks [13–16] to candidate centroids. Centroids having greater density and being far from each other are then chosen to compose the K centroids of the emerging solution, which, finally, is optimized by K-Means.

DPCC owes its existence to the ParDP tool [16]. From ParDP is inherited the use of the Principal Component Analysis (PCA) [6,19] technique to deal with the "*curse of dimensionality*", that is, reducing the high number of data point dimensions (coordinates or features) to the essential ones which retain data variations and contribute to keeping the semantics of the Euclidean distance. As in ParDP, DPCC applies the k-nearest neighbors (kNNs) strategy [4,14] to estimate the candidate centroids density. More specifically, first, the k nearest distinct distances of neighbors are determined for each centroid point. Let $NN_k(\mu_j)$ be the distance of the k-th nearest neighbor of μ_j. Then the density of μ_j is computed by evaluating the *reverse nearest neighbors* (RNNs) [20–22] set, similarly to the friendship relationship in human societies:

$$RNN(\mu_j) = \{\mu_h \in \wp | d(\mu_j, \mu_h) \leq NN_k(\mu_h)\} \quad (2)$$

Of course, the particular value of the k parameter of the kNN strategy (lower case k, to be not confounded with the capital letter K indicating the number of clusters) that optimizes density evaluation and then the quality of the resultant solution is a matter of experimental finding. Consequently, the second step of DPCC is repeated for each k value in a given range, like, e.g., [5...150] with step 5 (other values can be used as well), thus observing convergence of K-Means towards the "best" value of the objective cost function. Applying the 2nd step of DPCC to a particular value of k corresponds to creating a new generation of the population, where candidate solutions are crossed with each other according to the selection operations of kNN/RNN and density peaks.

A preliminary short version of this paper was presented in a conference paper [23]. Differences introduced from the conference paper are the following:

- The DPCC algorithm is now fully detailed.
- The 1st step of population set-up can either use a single seeding method or alternate between different careful seeding methods.
- The 2nd step was extended to optimize either the SSE or the Silhouette index (SI (see later in this paper) as the objective function cost.
- The 2nd step can now detect and handle the problem of empty clusters [24,25], which can occur for certain values of the k parameter.
- The performance assessment of DPCC was significantly extended, using both synthetic and real-world datasets. In particular, DPCC experimental results are compared with two known recent competitor K-Means variations: ImpKmeans [26], which is based on multivariate kernel density estimation [27] as a seeding method; and Nie-KMeans [25], which views cost optimization as a trace transformation, and develops a formal variation in K-Means which is deemed to be both efficient and accurate.

The rest of this paper is structured as follows. Section 2 summarizes the fundamental concepts underlying the design of DPCC. In particular, the classic Lloyd's K-Means algorithm is presented, together with the careful seeding methods and the main clustering measures used in this paper to assess the clustering accuracy. The section also clarifies the basic issues of genetic clustering algorithms and the fundamental concepts of density peaks. Section 3 describes the design of the DPCC algorithm and outlines its current implementation, which is based on Java parallel streams. Section 4 presents a series of experimental results regarding the application of DPCC to benchmark and real-world datasets, organized into four groups. Finally, Section 5 concludes the paper with an indication of further research work.

2. Background

The following describes some basic concepts underlying the proposed DPCC clustering algorithm.

2.1. Lloyd's K-Means

Lloyd's K-Means algorithm [2] represents the classical version of K-Means. Its behavior in pseudo-code is shown in Algorithm 1. Function $nc(x_i)$ denotes the nearest centroid to point x_i.

Algorithm 1. Lloyd's K-Means algorithm.

Input: Dataset $X = \{x_i\}_{i=1}^{N}$, $x_i \in R^D$, number of clusters K, $C_j \leftarrow \varnothing,\ \forall\, j,\ 1 \le j \le K$
Output: X partitioned together with SSE objective function cost.
1. *Initialization*: $C_j = C_j \cup \{\mu_j\}$, $1 \le j \le K$, $\mu_j \leftarrow x_i,\ i = unif\_random(1, N),\ x_i \in X$
2. *Partitioning*. Assign X data points to clusters:
$C_j = C_j \cup \{x_i\},\ x_i \in X,\ j = argmin_{h=1}^{K}(nc(x_i))$
3. *Update centroids*. $\left\{\mu'_j\right\}_{j=1}^{K},\ \mu'_j = \frac{1}{|C_j|}\sum_{x_i \in C_j} x_i$
4. *Check termination*. If new centroids $\left\{\mu'_j\right\}_{j=1}^{K}$ are still not stable, or the maximum number T of iterations has not been reached, re-execute from step 2.

Steps 2 and 3 of Algorithm 1 are intended to be repeated until convergence, that is, new centroids $\left\{\mu'_j\right\}_{j=1}^{K}$ are equal, according to a numerical threshold, to the centroids $\{\mu_j\}_{j=1}^{K}$ of previous iteration, or a maximum number T of iterations has been executed.

2.2. Seeding Methods

The uniform random centroids initialization procedure adopted by Lloyd's K-Means suffers from the limitations discussed in Section 1. Two basic seeding methods that improve uniform random are *Maximin* [10] and *K-Means++* [28]. Both these methods are stochastic and start by defining the first centroid $\mu_L,\ L = 1$, as a point selected by uniform random in the dataset. Subsequent centroids are added incrementally until $L = K$. Let $D(x_i)$ be the minimal distance of a data point x_i to the L existing centroids. *Maximin* defines the next centroid $\mu_{L+1} \leftarrow x^*,\ L = L + 1,$ as the point $x^* \in X$ that has the maximum value of $D(x^*)$. As a consequence, centroids are naturally selected to be far away from each other. *K-Means++* decides the next centroid probabilistically through a random switch process. First, to each point x_i is associated a probability to be selected as the next centroid, which is $\pi(x_i) = \frac{D(x_i)^2}{\sum_{h=1}^{N} D(x_h)^2}$. Therefore, the more a point x_i is near existing centroids, the less it has a chance to be selected. Both *Maximin* and *K-Means++* have proven to provide a useful initialization in practical cases.

Two Careful Seeding Methods

In this work, the Greedy_K-Means++ (*GKM++*) [17,18] and the *Refine* method proposed by Bradley and Fayyad in [29], and adapted by *Pairwise Nearest Neighbors* (*PNNs*) smoothing by Baldassi in [30] ($Refine^{PNN}$), are selected as careful seeding methods for DPCC.

GMK++. This method starts as K-Means++ and extends it by making S attempts in the choice of each next centroid. Among the S K-Means++ attempts, the "best" candidate centroid emerges to be the one that minimizes the SSE in the partitioning of the dataset corresponding to the existing centroids, extended with the prospective one. The value of the parameter S is established as a trade-off between clustering accuracy and the computational cost. As in [17], in this paper, the value $\lfloor 2 + logK \rfloor$ is adopted. Due to its computational cost, $O(KSND)$, *GKM++* can be difficult to apply to large and multidimensional datasets.

$Refine^{PNN}$. This method splits the dataset X into SEG randomly filled segments. As in [30], the value $SEG = \lceil \sqrt{N/2K} \rceil$ is chosen for the experiments. Each segment is then separately clustered by K-Means, e.g., with *Maximin* as the initialization procedure. When all the segments have been clustered, SEG solutions are available. Each solution consists of K centroids and K partitions. Obviously, all the partitions belonging to the same or to different solutions are disjoint by construction. The SEG solutions are reduced to one solution through subsequent PNN merge operations. At any moment, the two (not empty) partitions are chosen, whose merging would increase SSE the least.

2.3. Measures of Clustering Accuracy

The objective function SSE is an example of an internal clustering accuracy measure. Reduced values of SSE indicate good point distributions in clusters. Another internal measure that better expresses the internal cohesion and the good separation of clusters is the *Silhouette* index (SI). Let a_{x_i} be the average distance of a point x_i from all the other points in the same cluster, and b_{x_i} be the minimal average distance of x_i from the points in other clusters. The Silhouette coefficient of a point x_i is defined as follows:

$$SI_{x_i} = \frac{b_{x_i} - a_{x_i}}{\max(a_{x_i}, b_{x_i})} \tag{3}$$

and the global Silhouette index of the clustering is computed as the average of the individual Silhouette coefficients:

$$SI = \frac{1}{N} \sum_{i=1}^{N} SI_{x_i} \tag{4}$$

SI ranges in $[-1, 1]$. Values close to 1 denote internally high cohesive and well-separated clusters. Values towards 0 indicate high overlapping. Values close to -1 denote an erroneous clustering.

Two external indices useful to characterize the similarity degree of a clustering solution achieved by a given algorithm to the ground truth solution prepared by the designer of a benchmark dataset are the Centroid Index (CI) [31] and the Generalized Centroid Index (GCI) [32]. CI can be applied when the dataset comes with ground truth centroids (GTCs). GCI is useful when the synthetic dataset is equipped with ground truth partition (GTP) labels. Let CT be the centroid vector computed by a given algorithm. Each centroid $\mu_j \in CT$, $1 \leq j \leq K$ can be mapped to the nearest ground truth centroid $gtc_h \in GTC$, where, as normally happens, the cardinality of GTC is also K. Then the number of "orphans" in GTC can be computed, that is, the number of ground truth centroids upon which no centroid of CT is mapped. Similarly, ground truth centroids can be mapped to computed centroids of CT, and the number of orphans in CT can be determined. Then, the CI value

of the found clustering solution can be established as the maximum number of orphans in the two-directional mappings:

$$CI = \max\{\text{orphans(CT} \rightarrow \text{GTC)}, \text{orphans(GTC} \rightarrow \text{CT)}\} \quad (5)$$

CI ranges in $[0, K]$. $CI = 0$ characterizes a structurally correct clustering solution, where centroids of CT are very close or coincide with the anticipated GTC. A $CI > 0$ mirrors the number of erroneously computed centroids.

When ground truth partitions are available, the label sets of the K computed clusters/partitions P can be mapped on the provided label sets of the partitions of GTP. In the first mapping direction, a partition $P^j \in P$ is mapped to the nearest partition $GTP^h \in GTP$ by evaluating the Jaccard distance of the two sets: $J_d(P^j, GTP^h) = 1 - \frac{|P^j \cap GTP^h|}{|P^j \cup GTP^h|}$. As for the CI value, the GCI emerges as the maximum number of orphans in the two-direction mappings. The range and meaning of the GCI values are the same as for the CI.

Some well-known external measures of clustering quality include the Adjusted Rand Index (ARI), the Accuracy index (ACC) [33], the Normalized Mutual Information (NMI), F_score, and $Purity$ [34]. They are all implemented in Java and can be used with DPCC. While, for many of the details, the reader is referred, e.g., to [16,34], the following recalls the calculation of the ARI, which is a *pair-matching count* technique and depends on the construction of the *contingency matrix* (see Table 1) of calculated partitions P^1, P^2, ..., P^K, and ground truth partitions, assumed to be both of K sets: GTP^1, GTP^2, ..., GTP^K.

Table 1. Contingency matrix of computed partitions $P^i\big|_{i=1}^{K}$ vs. ground truth partitions $GTP^j\big|_{j=1}^{K}$.

Contingency Matrix	GTP^1	GTP^2	...	GTP^K	Σ
P^1	$n_{11} = \left\|P^1 \cap GTP^1\right\|$	n_{12}		n_{1K}	$n_1 = \sum_{j=1}^{K} n_{1j} = \left\|P^1\right\|$
P^2	n_{21}	n_{22}		n_{2K}	n_2
...	...	...		...	...
P^K	n_{K1}	n_{K2}		n_{KK}	n_K
Σ	$m_1 = \sum_{i=1}^{K} n_{i1} = \left\|GTP^1\right\|$	m_2		m_K	$\sum_i n_i = \sum_j m_j = N$

The ARI can be computed as follows:

$$ARI = \frac{2(ad - bc)}{(a+b)(b+d) + (a+c)(c+d)} \quad (6)$$

where $a = \frac{1}{2}\sum_{i=1}^{K}\sum_{j=1}^{K} n_{ij}(n_{ij} - 1)$ counts the number of pairs of points that belong to the same cluster in P and the same cluster in GTP (true positive); $b = \frac{1}{2}(\sum_{j=1}^{K} m_j^2 - \sum_{i=1}^{K}\sum_{j=1}^{K} n_{ij}^2)$ counts the number of pairs that are in the same cluster in P but in different clusters in GTP (false positive); $c = \frac{1}{2}(\sum_{i=1}^{K} n_i^2 - \sum_{i=1}^{K}\sum_{j=1}^{K} n_{ij}^2)$ counts the number of pairs that are in distinct clusters in P but in the same cluster in GTP (false negative); and $d = \frac{1}{2}(N^2 + \sum_{i=1}^{K}\sum_{j=1}^{K} n_{ij}^2 - (\sum_{i=1}^{K} n_i^2 + \sum_{j=1}^{K} m_j^2))$ counts the number of pairs which belong to different clusters in P and different clusters in GTP (true negative).

The same information from the contingency matrix can be used to compute the *F-score* measure, through the values of *Precision* and *Recall*:

$$Precision = \frac{a}{a+b}, \quad Recall = \frac{a}{a+c} \quad (7)$$

$$F\text{-}score = \frac{2 * (Precision * Recall)}{Precision + Recall} \tag{8}$$

The *Purity* index, which can provide an indication close to the *ACC*, can be computed as follows:

$$Purity = \frac{1}{N}\sum_{i=1}^{K} max_j\left(P^i \cap GTP^j\right) \tag{9}$$

All the above-mentioned external measures admit values in the range $[0, 1]$ (negative values of *ARI*, down to -1, are assumed to be mapped to 0). Values close to 1 mirror high-quality clustering (maximum similarity among P partitions and those of GTP). The value 0 denotes maximal dissimilarity. These measures can be applied when ground truth partition labels are available, e.g., defined by application experts in real-world datasets, or established as part of a "golden solution" achieved by a clustering algorithm deemed accurate.

2.4. Genetic Clustering

Clustering using genetic algorithms (e.g., [11]) has proven to be effective, although it can entail a high computational cost. The starting point is the creation of a population of candidate solutions. Subsequent generations of the population are then built using the basic genetic operations of *selection*, *crossover*, and (possibly) *mutation*. In the elitist approach of Fränti [11], candidate solutions are stored with their corresponding *SSE* values. Pairs of best solutions are then crossed (more specifically, merged by *Pairwise Nearest Neighbors*—*PNN*s—operations) to generate new solutions, which are refined by K-Means. After a certain number of iterations/generations, the emerging best solution is returned. In Ref. [12], the Franti genetic algorithm was prototyped in parallel Java to improve its practical computation time, and some experimental results were retrieved. A key feature of the implementation is an efficient realization of the *PNN* operations [35].

2.5. Density Peaks

A reference clustering technique based on density peaks (DPs) was proposed by Rodriguez and Laio in 2014 [13]. DP demonstrated effectiveness in clustering data with non-uniform distributions. According to DP, centroids are points of higher local density, surrounded by points of smaller density. However, a centroid should have a high distance from other points with greater density. DP associates two quantities with each data point x: its density $\rho(x)$ and the minimal distance, $\delta(x)$, to the nearest point with higher density. The latter point is often referred to as the *big brother* of x [14]. To be chosen as a centroid, a point should have high values of ρ and δ (two centroids are required to be far from each other to avoid splitting a big real cluster into smaller wrong clusters). A third quantity, γ, is associated with x: $\gamma = \rho * \delta$. In brief, candidate centroids should have greater values of γ. DP suggests using the *decision graph* [13], which shows γ vs. ρ or δ vs. ρ of data points, as a help in assessing the possible number K of centroids. Very often, the number of centroids emerges as graphical points neatly separated from all the other points. To complete clustering, DP first sorts the dataset points by descending values of γ, then chooses as centroids the first K points of the sorted dataset, and labels them from 1 to K. Finally, clustering is terminated by propagating, recursively, the label of its big brother to each remaining point.

A critical issue with DP is density estimation. As a "rule of thumb", basic DP in [13] adopts a *cutoff distance*, d_c, as the radius of a hyperball around each point, with the requirement that in the hyperball would fall from 1% to 2% of the dataset points. The implementation of DP [13] exploits a Gaussian kernel to estimate the density of point x_i:

$$\rho(x_i) = \sum_{j=1}^{N} e^{-\left(\frac{d(x_i,x_j)^2}{{d_c}^2}\right)} \tag{10}$$

Several DP-based clustering algorithms have been defined, which use the k-nearest neighbors (kNNs) approach as the basis for estimating point densities. Due to the high cost $O(N^2)$ related to the calculation of all pairwise distances, FastDP [14] builds an approximate kNN tree for estimating ρ and δ of points. In particular, the density is computed as the inverse of the average of the kNN distances of points.

The kNN approach was specialized in [20,22] with the concepts of *reverse* and *natural* nearest neighbors. Let the k-th nearest point of a point p be $NN_k(p)$. The *reverse nearest neighbors* (RNNs) of p, $RNN(p)$, consists of the set of points q which have p within their $NN_k(q)$ distance: $RNN(p) = \{q \in X | \, d(p,q) \leq NN_k(q)\}$. Two points p and q are said to be natural neighbors when each one belongs to the RNN of the partner: $p \in RNN(q) \wedge q \in RNN(p)$. After that, the *Mutual Nearest Neighbors* (MNNs) set of a point p arises: $MNN(p) = kNN_k(p) \cup RNN(p)$. The use of RNN is effective because, as shown in [22], it can automate the definition of the number of clusters K.

ParDP [16] is a recently developed tool based on density peaks. To smooth out the high cost of kNN, ParDP is implemented on top of Java parallel streams and lambda expressions [16,18,36]. ParDP can exploit two techniques for density estimation. The first one is derived from FastDP [14], although the kNN tree is not built. The inverse of the average distance of the k-nearest neighbor distances defines the point density. The second technique uses the average of the kNN distances as a local cutoff distance. Local cutoff distances are then sorted, and the median value is used as global d_c from which a Gaussian kernel computes density. Both techniques tend to limit the influence of outliers on the clustering process. The effectiveness of ParDP was demonstrated in [16] by several experiments on benchmark and real-world datasets. Results were compared to two special ParDP implementations based on the kNN approach of [19] and the MNN approach advocated in [20,22].

3. DPCC Algorithm

The design of DPCC emerged from experimental evidence that, by collecting multiple K-Means initialization solutions (candidate centroid configurations) generated by a careful seeding method, it is typically observed that no particular solution approaches the optimal one. However, centroids of the various solutions tend to thicken around the ground truth centroids. The aim of DPCC is then to identify Density Peaks of Candidate Centroids and to exploit the concepts of DP to identify and extract K final centroids, which serve as the initial solution for K-Means to optimize. It turns out that, in many cases, a few iterations, and sometimes just one, are required to converge to the final accurate solution.

The logic of DPCC is summarized in Algorithm 2. DPCC borrows from ParDP [16] the *Principal Component Analysis* (PCA) technique [6,19] to handle multidimensional data points, and the basic mechanisms of density peaks. DPCC, though, is characterized by its genetic volition of relying on a population of J elitist candidate solutions/centroids, with subsequent generations that are built by applying kNN in a given range of the k parameter. For each particular value of k, Density Peaks of Candidate Centroids are determined, along with the three quantities ρ (density), δ (minimal distance to big brother), and $\gamma = \rho * \delta$, for each point. The *Reverse Natural Neighbors* (RNN) strategy is used to estimate centroid densities. At each new generation, solution centroids are crossed with each other, with the population that is sorted by descending values of γ. The first K centroids in the sorted generation are then extracted and used as the initialization of K-Means. The best solution, among the various generations, is identified, which optimizes the objective cost (minimization of the SSE or maximization of the SI). Finally, clustering quality measures such as CI/GCI, ARI, ACC, NMI, and so forth, of the resultant solution are evaluated.

Algorithm 2. DPCC operational behavior.

Input: dataset $X(N, D, K)$, seeding method SM, J, Rep, k_{start}, k_{end}, k_{step}
Step 1. $Population\ set - up$
load X, possibly scale point coordinates, apply Principal Component Analysis (PCA)
$\wp \leftarrow \varnothing$
repeat J times{
    $bestCost \leftarrow \infty$, $candidateSolution \leftarrow ?$
    repeat Rep times{
        $solution = SM(X)$
        $cost \leftarrow SSE(solution)$
        $if(\ cost < bestCost\ )\{$
            $cost \leftarrow bestCost$
            $candidateSolution \leftarrow solution$
        }
    }
    $\wp \leftarrow \wp \cup \{candidateSolution\}$
}
Step 2. $Creation\ of\ generations$
$bestCost \leftarrow \infty$, $bestSolution \leftarrow ?$
$for(\ k = k_{start}; k \leq k_{end}; k = k + k_{step}\ )\{$
    //modify $\wp$ by crossing candidate centroids $\{\mu_j\}_{j=1}^{J*K}$ according to kNN/RNN
    apply kNN with k to $\wp$ and determine the density $rho(\mu), \forall\ \mu \in \wp$, by RNN
    sort $\wp$ by descending rho values
    compute $delta(\mu)$ and $gamma(\mu), \forall\ \mu \in \wp$
    sort $\wp$ by descending $gamma$ values
    extract first K centroids in sorted $\wp$ and define the target solution
    refine target solution by K-Means iterations until convergence of centroids
    if(empty clusters) stop iteration and start next one with new k
    $cost \leftarrow ObjectiveFunctionCost(refined\ target\ solution, X)$
    $if(\ cost\ improves\ bestCost\ )\{$
        $bestCost \leftarrow cost$
        $bestSolution \leftarrow refined\ target\ solution$
    }
}
Output: Clustering indexes on the resultant solution:
$SSE/SI,\ CI/GCI,\ ARI,\ ACC,\ NMI, \ldots$

DPCC consists of two steps. The first one is devoted to building the population of J candidate solutions. Solutions can be generated (see Section 2.2) by GKM++, $Refine^{PNN}$, or by alternating the use of these two seeding methods. Due to the stochastic character of these methods, each candidate solution is selected as the one that minimizes the SSE cost, after Rep repetitions (e.g., $Rep = 5$) of the seeding procedure.

The second step creates different generations of the population by varying the k parameter in the range from $[k_{start}, k_{end}]$ with a step k_{step}. For each k value, the kNN technique is applied to find the first k distances to neighbors for each centroid μ. After that, the k-neighborhood of μ is identified. The density of μ is estimated by deriving the RNN of μ from the built k-neighborhoods. To facilitate the determination of the $delta$ parameter of each centroid, the population vector is preliminarily sorted by descending density values. This way, centroid points with greater density are easily searched on the left of the target

centroid. From the values of *rho* (density) and *delta* (minimal distance to the big brother), the *gamma* quantity is computed. Finally, the population vector is sorted by descending *gamma* values, and the first K centroids of the sorted population are selected to compose the target solution for K-Means to optimize. The goal of the second step is to automatically infer the k value that optimizes the assumed objective function, that is, the SSE or the Silhouette index SI measure.

It is worth noting that, for certain k values, some clusters of the target solution can be empty. In this case, the corresponding generation is abandoned. To improve the various sorting operations of the population, candidate centroids are never moved. Rather, the indirection of an index vector is used to mirror the actual centroid order.

As for ParDP [16], DPCC is implemented on top of Java parallel streams (which depend on the underlying fork/join mechanism) with lambda expressions used to functionally capture data manipulation. Algorithm 3 shows a Java code fragment of the method used to calculate RNN.

Algorithm 3. An excerpt of the method used to calculate RNN.

```
void rnn(){
    Stream<DataPoint> pStream=Stream.of(population);
    if(PARALLEL) pStream=pStream.parallel();
    pStream
        .map(p->{
            int i=p.getID();
            detect the first k distinct distances of p nearest neighbors
            fill the k-neighborhood of p in KNN[i]
        })
        .forEach(p->{});
    //determine RNN of each centroid point
    pStream=Stream.of(population);
    if(PARALLEL) pStream=pStream.parallel();
    pStream
    .map(p->{
        int i=p.getID();
        Set<Integer> RNN=new HashSet<>();
        for(int j=0; j<J*K; ++j)
            if(KNN[j].contains(i)) RNN.add(j);
        p.setRho(RNN.size());
        return p;
    })
    .forEach(p->{});
}//rnn
```

The first part of Algorithm 3 focuses on the kNN calculations (see also [16]) of the k-nearest neighbor distances and the composition of the k-nearest neighborhood of the centroid point. The last part of the method computes the RNN and terminates with the density estimation. k-neighborhoods are held in the global array data structure KNN[] of $J * K$ items.

A notable point in Algorithm 3 concerns population centroid points that can be processed in parallel (the global PARALLEL field is true by default). However, to avoid concurrent/parallel inconsistencies (e.g., race conditions), each point only modifies itself. For instance, during the RNN computation, the global k-neighborhoods KNN[.] are si-

multaneously accessed, but point p is restricted to update its local RNN set only. The terminal operation for each() serves the purpose of starting the various map() intermediate operations in parallel. Parallel streams are also used, for example, in the K-Means implementation and in the computations of clustering quality measures.

4. Experimental Results

DPCC clustering accuracy and runtime efficiency were assessed by applying it to several benchmark and real-world datasets. All the experiments were conducted on a Windows 11 Pro desktop platform, Dell XPS 8940, Intel i7-10700 (8 physical+8 virtual cores), CPU@2.90 GHz, 32 GB RAM, using Java 25.

4.1. First Group of Experiments

As a starting point, DPCC was applied to clustering the 2-d benchmark datasets reported in Table 2, which were downloaded from [37]. These datasets come with ground truth centroids, and are often used to evaluate the effectiveness of a proposed algorithm. The selected datasets can have spherical clusters with the same size and regularly placed in the data space (i.e., Birch2, which has clusters organized along a sine curve), or Gaussian distributed clusters having different sizes, overlap degrees and positions in the data space (i.e., S3, Asymmetric, Unbalance2).

Table 2. First group of 2-dimensional benchmark datasets [37].

Dataset	N	K
Birch2	100,000	100
S3	5000	15
Asymmetric	1000	5
Unbalance2	6500	8

All datasets of Table 2 were studied by first scaling the point coordinates to the overall maximum. In addition, a population of $J = 20$ solutions, each achieved by taking the best of five repetitions of the $Refine^{PNN}$ seeding method, was prepared in the first step of DPCC.

Table 3 reports the k parameter value that minimizes the SSE, and the corresponding Silhouette index SI, Centroid Index CI, average number of K-Means iterations ($avgIT$) needed for refining the proposed solution, number (PCA) of essential coordinates, and cumulative elapsed time (ET), in seconds, for the execution of the two steps of DPCC. The results in Table 3 confirm that all the datasets were clustered correctly ($CI = 0$), and with a small number of K-Means iterations (in three cases, just one iteration was sufficient). Most of the execution time is spent on population setup.

Table 3. Experimental results for the datasets of Table 2.

Dataset	*k*	*SSE*	*SI*	*CI*	*avgIT*	*PCA*	*ET* (sec)
Birch2	5	0.23	0.78	0	1	1	73.0
S3	40	18.82	0.49	0	1.57	2	1.52
Asymmetric	5	0.98	0.64	0	1	2	0.42
Unbalance2	5	1.09	0.85	0	1	2	1.17

Figures 1–4 confirm the accurate clustering achieved. In almost all cases, the ground truth centroids (gt, red) and the centroids found by DPCC (ct, black) are very close

to each other or coincide. The smaller value of SI for S3 witnesses a higher degree of cluster overlapping.

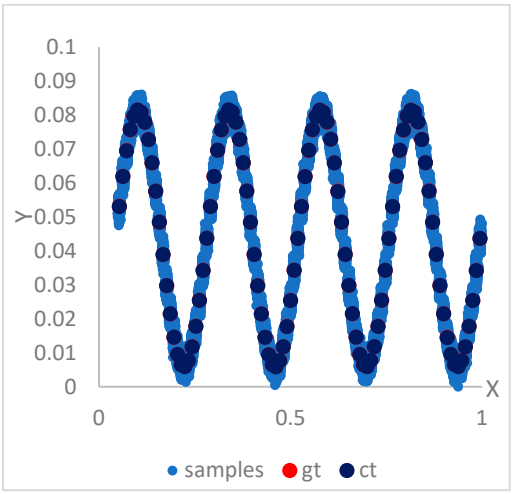

Figure 1. Centroids of Birch2 proposed by DPCC.

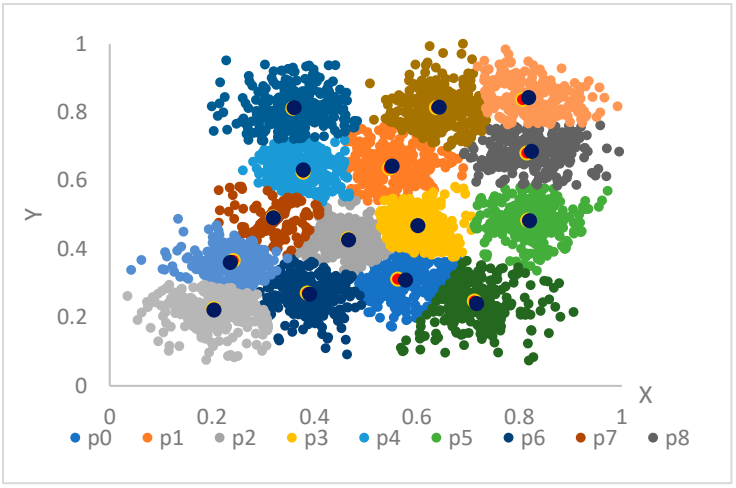

Figure 2. Partitions and centroids of S3 dataset found by DPCC.

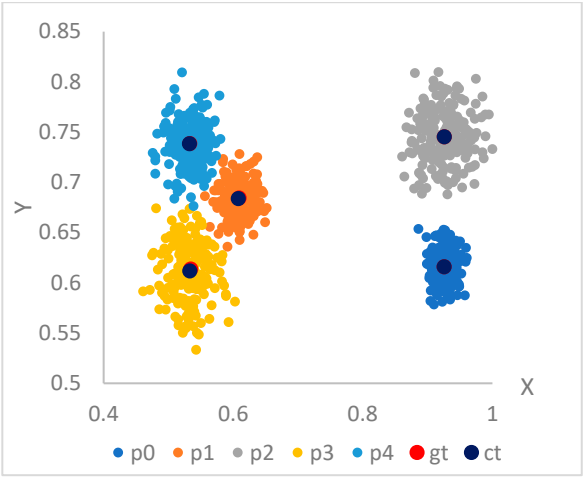

Figure 3. Partitions and centroids of Asymmetric dataset found by DPCC.

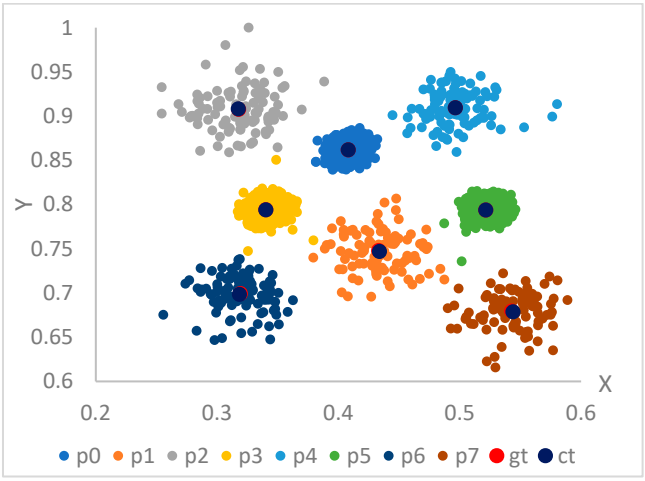

Figure 4. Partitions and centroids of Unbalance2 dataset found by DPCC.

4.2. Second Group of Experiments

A second series of experiments was devoted to evaluating some synthetic and real-world datasets used by Senol in Ref. [26] to assess the clustering abilities of ImpKmeans, which relies on multivariate kernel density estimation to initialize centroids. The 19 datasets used are listed in Table 4. They are all provided with ground truth partition labels. For comparison purposes, the datasets of Table 4 were processed by first scaling the point coordinates by min–max normalization.

Table 4. Second group of benchmark and real-world datasets [26].

Dataset	N	D	K
2d-3c	715	2	3
2d-4c	1261	2	4
2d-9c	2990	2	9
2d-20c	1517	2	20
Breast Cancer	699	10	2
Corners	2000	2	4
Diamond9	3000	2	9
D31	3100	2	31
Ds-850	850	2	5
Forty	1000	2	40
Iris	150	4	3
Outliers	1000	2	4
R15	600	2	15
Size1	1000	2	4
S-Set1	5000	2	15
ST-900	900	2	9
Thyroid	215	5	3
Twenty	1000	2	20
Xclara	3000	2	3

In the 1st step of DPCC, a population of $J = 20$ candidate solutions generated by the $Refine^{PNN}$ seeding method, repeated five times per solution, was prepared for each dataset. In the 2nd step, the goal was to minimize the SSE in the various k, $k \in [5, 150]$ generations. Unlike the other cases, the populations of Corners and Outliers were created by alternating solutions from *GKM++* and $Refine^{PNN}$ methods (see Section 2.2), and in the 2nd step, the Silhouette index SI, instead of the SSE, was used as the objective function to maximize.

Table 5 collects the clustering results achieved by DPCC. For each dataset, the observed values of the external measures GCI, ARI, ACC, and NMI are also reported.

Table 5. DPCC experimental results for the datasets of Table 4.

Dataset	k	SSE	SI	GCI	avgIT	PCA	ACC	NMI	ARI	ET (sec)
2d-3c	5	15.83	0.50	0	1.0	2	0.817	0.680	0.631	0.32
2d-4c	5	2.76	0.87	0	1.0	2	0.999	0.996	0.998	0.38
2d-9c	5	4.10	0.84	0	1.0	2	0.998	0.995	0.997	0.78
2d-20c	10	2.94	0.66	0	1.47	2	0.985	0.980	0.975	0.38
Breast Cancer	5	243.44	0.60	0	1.0	9	0.956	0.736	0.839	0.47
Corners	10	56.20	0.46	0	3.07	2	1.0	1.0	1.0	0.53
Diamond9	5	29.01	0.55	0	1.0	2	1.0	1.0	1.0	0.74
D31	5	4.88	0.58	0	1.07	2	0.977	0.968	0.954	1.93
Ds-850	5	13.06	0.56	0	1.0	2	0.954	0.886	0.890	0.44
Forty	5	0.98	0.69	0	1.0	2	1.0	1.0	1.0	1.58
Iris	5	7.00	0.50	0	1.0	4	0.887	0.742	0.716	0.21
Outliers	20	29.44	0.57	0	5.47	2	1.0	1.0	1.0	0.50
R15	5	0.57	0.75	0	1.0	2	0.997	0.994	0.993	0.55
Size1	5	17.18	0.59	0	1.0	2	0.982	0.926	0.956	0.46
S-Set1	5	10.29	0.71	0	1.13	2	0.998	0.995	0.995	1.68
ST-900	5	11.25	0.44	0	1.07	2	0.921	0.855	0.831	0.57
Thyroid	5	10.65	0.56	0	1.0	5	0.888	0.597	0.628	0.23
Twenty	5	1.72	0.74	0	1.0	2	1.0	1.0	1.0	0.80
Xclara	5	38.22	0.69	0	1.0	2	0.998	0.987	0.993	0.50

All the datasets were correctly clustered by DPCC, as confirmed by the $GCI = 0$ value that emerged after a minimal number of K-Means iterations. Careful clustering is also confirmed by Figures 5–8 that show the partitions and the centroids generated by DPCC in four representative cases.

The results of Table 5 were compared to those obtained by ImpKmeans in Ref. [26]. It emerged that, in 15 cases out of 19, DPCC was able to generate a clustering solution with the identical ARI to that in [26]. In the two cases of 2d-3c and Thyroid, ImpKmeans produced a better result than DPCC: $ARI^{DPCC}_{2d-3c} = 0.631$ vs. $ARI^{ImpKmeans}_{2d-3c} = 0.764$, $ARI^{DPCC}_{Thyroid} = 0.628$ vs. $ARI^{ImpKmeans}_{Thyroid} = 0.843$. Finally, in the two cases of 2d-20c and D31, DPCC outperformed ImpKmeans: $ARI^{DPCC}_{2d-20c} = 0.975$ vs. $ARI^{ImpKmeans}_{2d-20c} = 0.838$, and $ARI^{DPCC}_{D31} = 0.954$ vs. $ARI^{ImpKmeans}_{D31} = 0.549$.

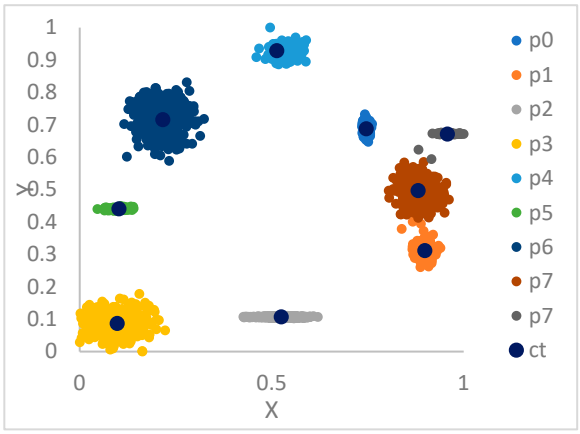

Figure 5. DPCC-generated partitions and centroids for the 2d-9c dataset.

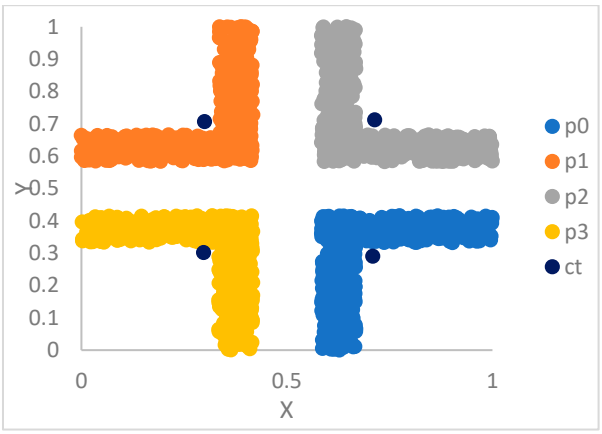

Figure 6. DPCC-generated partitions and centroids for the Corners dataset.

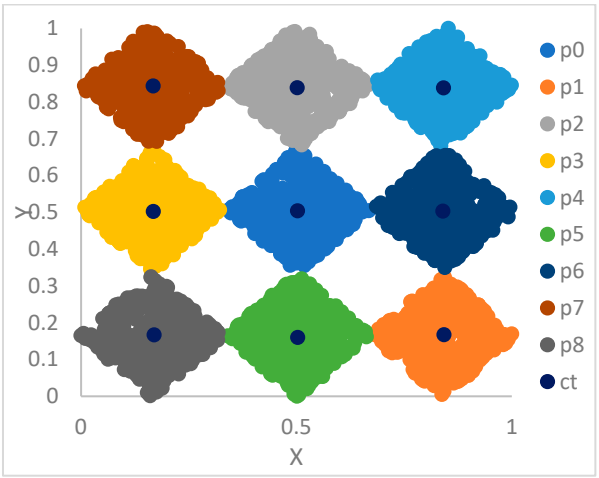

Figure 7. DPCC-generated partitions and centroids for the Diamond9 dataset.

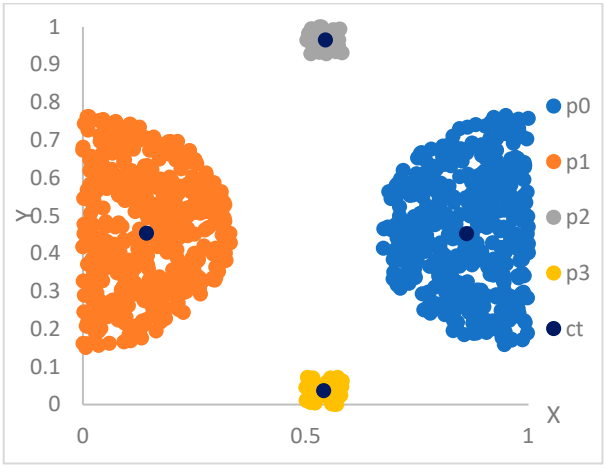

Figure 8. DPCC-generated partitions and centroids for the Outliers dataset.

4.3. Third Group of Experiments

This series of experiments aimed to analyze the DPCC behavior when clustering the 15 real-world challenging datasets chosen in [25] to test the performance of a reformulated K-Means algorithm with a new objective function based on trace transformation. This new algorithm will be referred to as Nie-KMeans. The datasets are specified in Table 6, sorted by the ascending number of point dimensions D (i.e., the number of coordinates or features). Data points are processed without scaling.

Table 6. Third group of real-world datasets [25].

Dataset	N	D	K
Balance	625	4	3
Dermatology	366	34	6
uspst	2007	256	10
USPSdata_20	1854	256	10
USPSdata	9298	256	10
MSRA25	1799	256	12
PalmData25	2000	256	100
Binalpha	1404	320	36
Ecoli	336	343	8
Corel_5k	5000	423	50
MnistData_05	3495	784	10
MnistData_10	6996	784	10
Coil20Data_25	1440	1024	20
Mpeg7	1400	6000	70
TDT2_10	653	36,771	10

For comparison purposes with the results in [25], the *ACC*, *NMI*, *F-score*, and *ARI* clustering measures will be evaluated. The DPCC clustering results are shown in Table 7. As one can see, all datasets receive a $GCI > 0$. Therefore, all the solutions proposed by DPCC (and by Nie-KMeans in [25]) are imperfect. However, the degree of similarity between the found partitions and the ground truth partitions is reflected by the values of the external measures.

Table 7. DPCC experimental results for the datasets of Table 6.

Dataset	k	SSE	SI	GCI	avgIT	PCA	ACC	NMI	F-score	ARI	ET (sec)
Balance	10	3473.68	0.17	1	4.47	4	**0.5456**	**0.1547**	**0.4853**	**0.1725**	0.42
Dermatology	25	5585.84	0.23	1	2.27	34	0.8607	0.8633	0.7722	0.7153	0.51
uspst	25	63,342.64	0.16	1	14.23	226	**0.7035**	**0.6180**	**0.5843**	**0.5351**	3.66
USPSdata_20	5	66,403.88	0.16	1	10.93	226	**0.7006**	**0.6304**	**0.5901**	**0.5416**	3.61
USPSdata	30	333,918.72	0.16	1	13.87	226	**0.7120**	**0.6164**	**0.5894**	**0.5405**	17.60
MSRA25	5	1.53×10^8	0.18	2	11.13	240	0.5453	0.5956	0.4062	0.3448	3.72
PalmData25	5	5.12×10^8	0.30	15	12.70	252	**0.7950**	**0.9269**	**0.7107**	**0.7076**	54.07
Binalpha	5	67,133.64	0.06	9	14.93	315	**0.4822**	**0.5944**	**0.3258**	**0.3064**	10.46
Ecoli	35	340.73	0.02	2	3.90	339	**0.7619**	**0.6343**	**0.7699**	**0.6912**	0.98
Corel_5k	5	4.36×10^6	0.09	19	48.77	183	**0.1888**	**0.2684**	**0.0844**	**0.0627**	58.59
MnistData_05	40	0.75×10^9	0.06	1	24.77	434	0.5568	0.4848	0.4137	0.3471	12.96
MnistData_10	5	1.76×10^{10}	0.06	2	37.83	434	0.5812	0.4988	0.4290	0.3628	29.74
Coil20Data_25	20	2.39×10^9	0.23	4	24.30	893	**0.7**	**0.7918**	**0.6315**	**0.6108**	15.22
Mpeg7	25	5864.37	0.09	21	12.03	5248	**0.5079**	**0.7173**	**0.3690**	**0.3585**	858.65
TDT2_10	10	1.96×10^5	0.26	3	6.97	11,028	0.4150	0.4186	0.2429	0.0960	117.25

Following a careful comparison with the average clustering measures documented in [25], results where DPCC outperformed Nie-KMeans, in all four measures *ACC*, *NMI*, *F-score* and *ARI*, are highlighted in bold in Table 7. This occurs in 10 out of 15 datasets. In three of the remaining datasets (Dermatology, MnistData_10, and TDT2_10), DPCC proposed better values for both the *ACC* and the *NMI*, whereas Nie-KMeans was a little more accurate for the *F-score* and *ARI*. For the MSRA25 dataset, DPCC showed better values for the *ACC*, *NMI*, and *ARI*. Finally, for the MnistData_05 dataset, DPCC proposed only a better *ACC* value than Nie-KMeans.

It is worth noting that the elapsed runtime (ET) in Table 7, and also in Tables 3 and 5, does not include the time required to apply the PCA technique to reduce the number of dimensions. For example, PCA required about 12 min for the TDT2_10 dataset, and 47 s for Mpeg7.

4.4. Fourth Group of Experiments

The irregular, very challenging synthetic worm-like datasets [37] described in Table 8 were used to compare the DPCC clustering results with those achieved, e.g., by the evolutionary K-Means/Random Swap algorithms proposed in [18]. The datasets were clustered after a preliminary scaling of the data point features by the overall maximum. Both the worms datasets come with ground truth partition labels.

Table 8. Fourth group of worm-like datasets [37].

Dataset	N	D	K
Worms_2D	105,600	2	35
Worms_64D	105,000	64	25

Worms_2D has 35 clusters in 2-dimensional space, and Worms_64D contains 25 clusters in 64-dimensional data space. The worm geometry is established by starting from a random position and moving along a random direction. At each subsequent step, points follow a Gaussian distribution whose variance is gradually increased step-by-step. In addition, the

movement direction is continually changed according to an orthogonal direction. In the 64D scenario, the orthogonal direction is randomly established at each new step.

Table 9 presents the DPCC-obtained clustering results. Despite the higher number of dimensions, and confirming results reported in the literature, e.g., in [14,18,38], Worms_64D is more amenable to clustering than Worms_2D. To the best of our knowledge, no clustering algorithm, e.g., based on K-Means, reported in the literature was able to correctly cluster Worms_2D (GCI = 0). Similar to tools described in [14,18,38], a GCI measure of about 8 was achieved by DPCC, meaning that 8 out of 35 clusters were incorrectly determined. For the Worms_64d, DPCC achieved a minimal SSE significantly smaller than that in [18]: about 12,555 against 18,060. All of this was confirmed by the good values of the observed ACC, NMI, and ARI measures. Low values of the SI indicate a high degree of cluster overlapping.

Table 9. DPCC experimental results for the datasets of Table 8.

Dataset	k	SSE	SI	GCI	avgIT	PCA	ACC	NMI	ARI	ET (sec)
Worms_2D	45	80.50	0.36	8	104.87	2	0.4975	0.6010	0.4644	50.39
Worms_64D	10	12,155.28	0.09	0	40.57	64	0.8696	0.7541	0.8036	194.12

Figures 9 and 10 show the registered SSE vs. k, respectively, for the Worms_2D and Worms_64D datasets. Figure 10, in particular, indicates that the optimal value of the SSE was reached for $k = 10$.

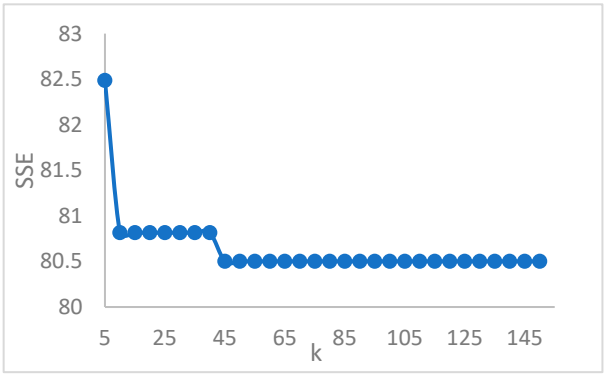

Figure 9. SSE vs. k for the Worms_2d dataset.

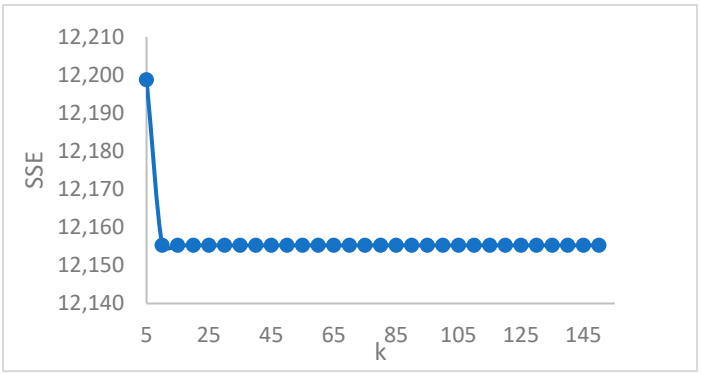

Figure 10. SSE vs. k for the Worms_64d dataset.

5. Conclusions

This paper proposes a novel variant of the K-Means unsupervised clustering algorithm [1–4,9], which is named Density Peaks of Candidate Centroids (DPCC). DPCC is

characterized by the development of a new seeding method for centroid initialization. The new seeding method rests on genetic concepts [11,12] and density peaks [13–15]. First, a population of candidate elitist solutions, each with K centroids, is built by using some existing careful seeding methods. Although none of these solutions is close to the optimal one, the centroids of the various solutions tend to thicken around the ground truth centroids. Then, subsequent generations of the population are generated by using the k-nearest neighbors (kNNs) [14,19] strategy, for different values of the k parameter. For each value of k, the k-neighborhood of each population centroid is determined, and its density is inferred by constructing the *reverse nearest neighbors* (RNNs) set. After that, density peaks [13,16] of candidate centroids are identified, and the population is reorganized accordingly to define a new generation. Finally, a target solution is extracted for K-Means by selecting K centroids that are both of higher density and are far from each other. The target solution is optimized over a few K-Means iterations.

The reliability and accuracy of DPCC are assessed by applying it to several benchmark and real-world datasets. Clustering results are compared with those achieved by competitor algorithms, including Nie-KMeans [25] and ImpKmeans [26]. DPCC qualifies as being able to furnish more accurate and often better results than those of its competitors.

Advancement of the work is geared to the following points: first, extending DPCC with other seeding methods to set up the population, including the use of Random Swap [38,39] for improving a candidate solution by using a limited number of iterations; second, experimenting with different techniques for the estimation of centroid point density, e.g., as in ParDP [16]; third, extending the availability of DPCC by implementing it in other languages like Python and MATLAB.

Author Contributions: Conceptualization, L.N.; methodology, L.N.; software, L.N.; validation, all authors; investigation, all authors; data curation, all authors.; writing—original draft preparation, L.N.; writing—review and editing, all authors; visualization, all authors; supervision, L.N. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data available by request to authors.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability 1967*; University of California: Berkeley, CA, USA, 1967; pp. 281–297.
2. Lloyd, S.P. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [CrossRef]
3. Jain, A.K. Data clustering: 50 years beyond k-means. *Pattern Recognit. Lett.* **2010**, *31*, 651–666. [CrossRef]
4. Wu, X.; Kumar, V.; Ross Quinlan, J.; Ghosh, J.; Yang, Q.; Motoda, H.; McLachlan, G.J.; Ng, A.; Liu, B.; Yu, P.S.; et al. Top 10 algorithms in data mining. *Knowl. Inf. Syst.* **2008**, *14*, 1–37. [CrossRef]
5. Everitt, B.S.; Landau, S.; Leese, M.; Stahl, D. *Cluster Analysis*, 5th ed.; Wiley: Hoboken, NJ, USA, 2011.
6. Wang, R. *Introduction to Machine Learning—From Math to Code*; Cambridge University Press: Cambridge, UK, 2026.
7. Mahajan, M.; Nimbhorkar, P.; Varadarajan, K. The planar kmeans problem is NP-Hard. *Theor. Comput. Sci.* **2012**, *442*, 13–21. [CrossRef]
8. Aloise, D.; Deshpande, A.; Hansen, P.; Popat, P. NP-hardness of Euclidean sum-of-squares clustering. *Mach. Learn.* **2009**, *75*, 245–248. [CrossRef]
9. Fränti, P.; Sieranoja, S. How much can k-means be improved by using better initialization and repeats? *Pattern Recognit.* **2019**, *93*, 95–112. [CrossRef]
10. Vouros, A.; Langdell, S.; Croucher, M.; Vasilaki, E. An empirical comparison between stochastic and deterministic centroid initialization for K-Means variations. *Mach. Learn.* **2021**, *110*, 1975–2003. [CrossRef]
11. Fränti, P. Genetic algorithm with deterministic crossover for vector quantization. *Pattern Recognit. Lett.* **2000**, *21*, 61–68. [CrossRef]

12. Nigro, L.; Cicirelli, F. Fast clustering convergence by genetic algorithm. In *International Conference on WorldS4*; Springer Nature: Singapore, 2024; pp. 331–342.
13. Rodriguez, R.; Laio, A. Clustering by fast search and find of density peaks. *Science* **2014**, *344*, 1492–1496. [CrossRef]
14. Sieranoja, S.; Fränti, P. Fast and general density peaks clustering. *Pattern Recognit. Lett.* **2019**, *128*, 551–558. [CrossRef]
15. Li, Z.; Tang, Y. Comparative density peaks clustering. *Expert Syst. Appl.* **2018**, *95*, 236–247. [CrossRef]
16. Nigro, L.; Cicirelli, F. ParDP: A parallel density peaks-based clustering algorithm. *Mathematics* **2025**, *13*, 1285. [CrossRef]
17. Baldassi, C. Recombinator-k-means: An evolutionary algorithm that exploits k-means++ for recombination. *arXiv* **2022**, arXiv:1905.00531. [CrossRef]
18. Nigro, L.; Cicirelli, F. Improving clustering accuracy of K-Means and Random Swap by an evolutionary technique based on careful seeding. *Algorithms* **2023**, *16*, 572. [CrossRef]
19. Du, M.; Ding, S.; Jia, H. Study on density peaks clustering based on k-nearest neighbors and principal component analysis. *Knowl.-Based Syst.* **2016**, *99*, 135–145. [CrossRef]
20. Dai, Q.Z.; Xiong, Z.Y.; Xie, J.; Wang, X.X.; Zhang, Y.F.; Shang, J.X. A novel clustering algorithm based on the natural reverse nearest neighbor structure. *Inf. Syst.* **2019**, *84*, 1–16. [CrossRef]
21. Ahmed, A.H.; Ashour, W. An initialization method for the K-Means algorithm using RNN and coupling degree. *Int. J. Comput. Appl.* **2011**, *25*, 1–6.
22. Zhu, Q.; Feng, J.; Huang, J. Natural neighbor: A self-adaptive neighborhood method without parameter k. *Pattern Recognit. Lett.* **2016**, *80*, 30–36. [CrossRef]
23. Nigro, L.; Cicirelli, F.; Pupo, F. Clustering algorithm based on density peaks of candidate centroids. In *Proceedings of the 11th International Congress ICICT 2026, London, UK, 24–27 February 2026*; Springer: Berlin/Heidelberg, Germany, 2026.
24. Pakhira, M.K. A modified K-Means algorithm to avoid empty clusters. *Int. J. Recent Trends Eng.* **2009**, *1*, 220.
25. Nie, F.; Li, Z.; Wang, R.; Li, X. An effective and efficient algorithm for K-means clustering with new formulation. *IEEE Trans. Knowl. Data Eng.* **2023**, *35*, 3433–3443. [CrossRef]
26. Şenol, A. Impkmeans: An improved version of the K-Means algorithm, by determining optimum initial centroids, based on multivariate kernel density estimation and kd-tree. *Acta Polytech. Hung.* **2024**, *21*, 111–131. [CrossRef]
27. Węglarczyk, S. Kernel density estimation and its application. In *Proceedings of the ITM Web of Conferences 2018*; EDP Sciences: Paris, France, 2018; Volume 23, p. 00037. [CrossRef]
28. Arthur, D.; Vassilvitskii, S. k-means++: The advantages of careful seeding. In *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2007; pp. 1027–1035.
29. Bradley, P.S.; Fayyad, U.M. Refining initial points for K-Means clustering. In Proceedings of the 15th International Conference on Machine Learning (ICML), Madison, WI, USA, 24–27 July 1998; pp. 91–99.
30. Baldassi, C. Systematically and efficiently improving K-Means initialization by Pairwise-Nearest-Neighbor smoothing. *arXiv* **2023**, arXiv:2202.03949.
31. Fränti, P.; Rezaei, M.; Zhao, Q. Centroid index: Cluster level similarity measure. *Pattern Recognit.* **2014**, *47*, 3034–3045. [CrossRef]
32. Fränti, P.; Rezaei, M. Generalized centroid index to different clustering models. In *Proceedings of the Joint IAPR International Workshop on Structural Syntactic, and Statistical Pattern Recognition, S+SSPR*; Springer: Cham, Switzerland, 2016; Volume 10029, pp. 285–296.
33. Franti, P.; Sieranoja, S. Clustering accuracy. *Appl. Comput. Intell.* **2024**, *4*, 24–44. [CrossRef]
34. Rezaei, M.; Franti, P. Set matching measures for external cluster validity. *IEEE Trans. Know. Data Eng.* **2016**, *28*, 2173–2186. [CrossRef]
35. Fränti, P.; Kaukoranta, T. Fast implementation of the optimal PNN method. In Proceedings of the 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269), Chicago, IL, USA, 7 October 1998; Volume 3, pp. 104–108.
36. Nigro, L. Parallel K-Means Algorithms in Java. *Algorithms* **2022**, *15*, 117. [CrossRef]
37. Fränti, P. Benchmark Datasets. Available online: http://cs.uef.fi/sipu/datasets/ (accessed on 30 January 2026).
38. Nigro, L.; Cicirelli, F.; Fränti, P. Parallel random swap: An efficient and reliable clustering algorithm in Java. *Simul. Model. Pract. Theory* **2023**, *124*, 102712. [CrossRef]
39. Fränti, P. Efficiency of random swap clustering. *J. Big Data* **2018**, *5*, 13. [CrossRef]