Applied Computing and Intelligence

https://www.aimspress.com/journal/aci

Applied Computing and Intelligence
6(2): 194–222.
DOI: 10.3934/aci.2026012
Received: 17 September 2026
Revised: 11 October 2026
Accepted: 21 October 2026
Published: 30 October 2026

Research article

KNN neighborhoods for careful clustering

Libero Nigro*

University of Calabria, DIMES, 87036 Rende, Italy

Correspondence: Email: libero.nigro@unical.it; +393392396819.

Academic Editor: Pasi Fränti

Abstract: Density peaks have proven effective at detecting patterns in datasets with general point distributions. Density peaks are exploited in this work to improve the K-Means and Random Swap algorithms. In particular, a new deterministic seeding method, named Hood, was proposed, which relies on data-point neighborhoods preliminarily built using the k-nearest neighbors (KNN) technique. A corresponding variant of K-Means (Hood-K-Means) was achieved, which is capable of accurate clustering in medium-sized and regularly distributed datasets. To cope with large and multidimensional datasets, an evolutionary version of Hood-K-Means was developed (Evo-Hood-K-Means), which rests on a population of candidate solutions/centroids established through a careful seeding method like K-Means++. The experiments confirmed that applying Hood to such an elitist population favors, in practical cases, the achievement of high-quality clustering. However, to handle both large, complex, and irregularly distributed datasets, a third algorithm based on Hood was developed (Evo-Hood-Random Swap), which replaced K-Means local search refinement of centroids with Random Swap global search on the population. All three algorithms could flexibly work with different objective function costs. They are implemented in parallel Java. In this paper, we described the newly developed algorithms and demonstrated, through a series of clustering experiments on synthetic and realistic datasets, the effectiveness of the achieved tools, which can ensure accurate clustering in limited time. In particular, the clustering results on some known complex and irregular datasets significantly improve upon similar results reported in the literature.

Keywords: density-based clustering; k-nearest neighbors; deterministic seeding; K-Means; Random Swap; genetic/evolutionary clustering

1. Introduction

Unsupervised clustering is an important area of machine learning [1] whose goal is to detect patterns in large-scale data with arbitrary point distributions. More specifically, the aim is to split data into groups called clusters, in such a way that data points assigned to the same cluster are similar to one another, and points of different clusters are dissimilar. Each cluster is represented by its central point called the centroid. In this work, our focus is mainly on partitional clustering [2,3], that is, clusters are disjoint sets of points. Partitional clustering can also benefit from density-based clustering [4], where centroids are identified based on point density, and from genetic/evolutionary clustering [5–8], which can recombine the centroids of a population of candidate solutions. The following highlights basic concepts that underlie the work described in this paper.

1.1. K-means concepts

A dataset $\boldsymbol{X}$ with $\boldsymbol{N}$ points is considered: $\boldsymbol{X} = \{\boldsymbol{x_i}|_{\boldsymbol{i=1}}^{\boldsymbol{N}}\}$, where each point is a vector with $\boldsymbol{D}$ numerical coordinates (features), $\boldsymbol{x_i} \in \boldsymbol{R^D}$. Similarity among points is expressed by their Euclidean distance: $\boldsymbol{d(x_i, x_j)} = \sqrt{(\boldsymbol{x_{i1}} - \boldsymbol{x_{j1}})^2 + \cdots + (\boldsymbol{x_{iD}} - \boldsymbol{x_{jD}})^2}$. A fundamental algorithm in partition clustering is K-Means [9–11], which is very often used due to its simplicity and efficiency, provided the dataset is not very large. K-Means assumes the number $\boldsymbol{K}$ of clusters is an input parameter. A clustering solution consists of a couple $< \boldsymbol{C^A}, \boldsymbol{P^A} >$, where $\boldsymbol{C^A}$ is a vector of $\boldsymbol{K}$ centroids: $\boldsymbol{C^A} = \{\boldsymbol{\mu_j^A}|_{\boldsymbol{j=1}}^{\boldsymbol{K}}\}$, and $\boldsymbol{P^A}$ is a vector of $\boldsymbol{K}$ partitions, that is sets of points: $\boldsymbol{P^A} = \{\boldsymbol{P_j^A}|_{\boldsymbol{j=1}}^{\boldsymbol{K}}\}$, with $\boldsymbol{P_r^A} \cap \boldsymbol{P_s^A} = \emptyset, \boldsymbol{r} \neq \boldsymbol{s}$, and $\cup \left\{\boldsymbol{P_j^A}|_{\boldsymbol{j=1}}^{\boldsymbol{K}}\right\} = \boldsymbol{X}$.

K-Means starts with an initial configuration of centroids, e.g., by randomly choosing $\boldsymbol{K}$ points in the dataset [10]. Then it partitions the $\boldsymbol{N}$ points by assigning them to clusters based on the nearest centroid rule: $\boldsymbol{\mu_j} = \boldsymbol{nc(x_i)}$ **with** $\boldsymbol{j} = \boldsymbol{argmin_h}\{\boldsymbol{d(x_i, \mu_h)}, \boldsymbol{h}$ **in** $[\boldsymbol{1}..\boldsymbol{K}]\}$. After that, centroids are redefined as the mean points of the resultant clusters/partitions: $\boldsymbol{\mu_j'} = \frac{\boldsymbol{1}}{|\boldsymbol{P_j}|}\sum_{\boldsymbol{x_i} \in \boldsymbol{P_j}} \boldsymbol{x_i}$. The two phases of partitioning and centroid update are repeated until a termination condition is met: either the centroids stabilize, that is, they are numerically indistinguishable from previous centroids, or a maximum number of iterations is executed. The modus operandi of K-Means is dictated by the overall goal of minimizing the *Sum-of-Squared Error* (*SSE*) default objective function cost:

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

The more points correctly aggregate around centroids, the smaller the value of the *SSE* internal clustering measure. It is worth noting that K-Means represents a heuristic algorithm, because perfect partitioning with absolute minimal *SSE* is an NP-hard problem, even when *K=2* clusters are used [12,13].

Long-standing studies of K-Means have shown its intrinsic weaknesses. In fact, K-Means clustering strongly depends on the initial centroids (see, e.g., [14–16]). Consequently, several seeding methods, deterministic or stochastic [17], were defined (see also later in this paper) that can improve the K-Means clustering result. However, the basic attitude of K-Means remains unchanged. It acts as a local refiner of centroids and is unable, for example, to move centroids from data space areas where centroids are in excess to areas where centroids are lacking, especially when some overlapping is missing [16]. Therefore, the possibility of getting stuck around a local suboptimal solution is always a reality. A common approach to try to cope with these problems is to repeat the execution of K-Means

(Repeated K-Means), with an independent initialization of centroids at each repetition, and to retain the "best" solution emerging from the executions.

1.2. About seeding methods

The uniform random method adopted by classical Lloyd's K-Means [10] initializes the K centroids as random points selected from the dataset. Consequently, it does not avoid two or more centroids being close to one another (a precondition to split a real cluster into multiple unnatural clusters), nor does it avoid choosing an outlier or noise point as a centroid. The above-mentioned problems can be controlled effectively by careful seeding methods like Maximin [18] and K-Means++ [19]. Other examples of good seeding methods are described, e.g., in [7,8,17]. As usual, Maximin and K-Means++ define the first centroid as a point randomly chosen in the dataset. Subsequent centroids, up to K, are then added by rules that differ between Maximin and K-Means++. Let $D(x_i)$ denote the minimal distance of a point x_i from the currently existing centroids. Maximin defines the next centroid as a point x^* with maximal $D(x^*)$. This rule ensures centroids are chosen far from one another. K-Means++ continues by first assigning to each point a probability of being selected as the next centroid, as follows:

$$\pi(x_i) = \frac{D(x_i)^2}{\sum_{x_j \in X} D(x_j)^2}. \quad (2)$$

Then the next centroid is chosen by a random Bernoulli process. As one can see, a point close to the defined centroids has a small chance of being selected, because it has a low numerator. More careful seeding (see, e.g., the Greedy-K-Means++ variation advocated in [6]) can be achieved by repeating a certain number of times (e.g., Rep=10) K-Means++, and by choosing the centroid configuration, among the Rep attempts, that optimizes the objective function cost, e.g., *SSE*.

1.3. Random Swap concepts

Unlike K-Means, Random Swap [20,21] performs a global search during the improvement of the centroids. Consequently, it has a high chance of getting a clustering solution close to the optimal one. Random Swap starts with an initialization of centroids through a seeding method and the definition of the corresponding partitions. Then a certain number T of swap iterations are carried out. At each swap, a centroid is randomly chosen in the vector of centroids of the current solution, which gets replaced by a data point randomly selected in the dataset. The new centroid vector is refined by a few K-Means iterations (e.g., 2) and its function cost is evaluated. In the case the new solution improves the current one (e.g., it has a lower *SSE*), it is accepted and becomes current for the next swap. Otherwise, the previous solution is restored and becomes current for the next iteration. As demonstrated in [21], provided a suitable number T of iterations is adopted (e.g., 10^4 or higher), Random Swap is capable of ensuring the achievement of a very good solution. The weak point is the high runtime often required to converge to a solution close to the optimal one.

1.4. Density-peaks concepts

Density-peaks-based clustering [4] identifies centroids as points with a higher-density neighborhood, surrounded by points with lower density, and that are far from points with higher density. The critical issue is the estimation of the point density ρ. In [4], a common cutoff distance d_c is assumed whose value is established by a thumb rule: Within the hypersphere of radius d_c around

each point, many points should fall from 1% to 2% of the dataset size. The density of each point x_i is then estimated by a Gaussian kernel:

$$\rho(x_i) = \sum_{j=1, j!=i}^{N} e^{-(\frac{d(x_i,x_j)^2}{{d_c}^2})}. \quad (3)$$

Density-peaks clustering associates with each point also a second parameter $\delta(x_i)$, which is the minimal distance to its *big brother* [22], that is, a point with greater density than the surrounding points. Clustering is then completed (see also [23] by first sorting the dataset by descending values of the $\gamma = \rho * \delta$ parameter. Centroids are then defined as the first K points in the sorted dataset, which establish the K class labels. Finally, to each remaining point, the cluster label is assigned recursively, as that of its big brother (a centroid is the big brother of its big brother). A lot of work was accomplished, following the seminal paper of Rodriguez and Laio in [4], to improve the estimation of the density of points. Many useful techniques (e.g., [22,24–26]) emerged based on the application of the k-nearest neighbors [27] algorithm (KNN). The first k distinct nearest neighbors of each point p are identified (k being an input parameter, to be assessed by a preliminary sensitivity analysis, not to be confused with upper case K, which denotes the number of clusters), with NNk(p) being the k-th most distant point. From NNk(p), the k-neighborhood *nbKNN(p)* is established: $nbKNN(p) = \{q \in X | d(p,q) \leq NN_k(p)\}$. Du et al. in [24] estimate the density of a point x_i by a Gaussian kernel of the points in the neighborhood of x_i:

$$\rho(x_i) = \mathrm{e}\,(-(\frac{1}{k}\sum_{x_j \in nbKNN(x_i)} d(x_i, x_j)^2)). \quad (4)$$

Other definitions of density can be based on the concepts of Reverse Nearest Neighbors (RNN) or Mutual Nearest Neighbors (MNN): $RNN(p) = \{q \in X |\, d(p,q) \leq NN_k(q)\}$, $MNN(p) = KNN(p) \cup RNN(p)$. $RNN(p)$ is a set composed of the points q that have p within their $NNk(q)$. Two points p and q are said to be natural neighbors if each one is in the RNN of the other. The $MNN(p)$ is the union of the neighborhood of p and its RNN. The performance of RNN and MNN has been explored in depth [8,23].

Application of the KNN algorithm has a high cost of $O(N^2)$ due to the necessity of determining all the pairwise distances among the dataset points. Sieranoja and Franti in [22] proposed a fast technique that builds an approximate KNN graph by a divisive approach. From the graph, the average distance avd of the k-nearest neighbors is evaluated, as well as the distance to the big brother. Point density is then estimated as $1/avd$ (density is mass/volume; mass is assumed unitary; the volume is proportional to the radius avd).

1.5. Evolutionary concepts

To deal with complex and large datasets, the exploitation of genetic/evolutionary [5–8,28] techniques can help. The idea is to prepare a population with a certain number of candidate solutions/centroids, built using some careful seeding method like K-Means++. The candidate solutions can then be recombined using some form of the basic genetic operations of selection, crossover, and (possibly) mutation.

Moreover, that is a property with experimental evidence: Although none of the candidate solutions can be close to the optimal one, the centroids of the solutions tend to thicken around the optimal or ground truth centroids. Exploiting such densities is key for extracting a careful clustering solution. Building a centroid initialization (seeding) from the population acts as a selection operation. A point p in the population that has high density lends itself, during the application of a seeding method, to become a centroid. On the other hand, points around p, which have a smaller density than p, have a

little chance of being selected as the next centroid. This is because seeding methods always try to ensure centroids are far from each other.

Refining such an initial solution by K-Means or Random Swap behaves as a crossover. Finally, (possibly) replacing the selected centroids in the population with the refined ones mutates the population by giving rise to a new generation. Subsequent generations can naturally converge toward an accurate clustering solution.

1.6. Paper contribution

In this paper, we develop a new deterministic seeding method, named Hood, which is based on Maximin. The Hood method relies on density peaks established by KNN neighborhoods and their overlap, and proves effective for careful clustering. More specifically, we propose three clustering algorithms based on Hood: a) Hood-K-Means, suited for handling medium-sized, possibly regularly distributed, datasets; b) Evo-Hood-K-Means, that is, an evolutionary variant of Hood-K-Means, which rests on a population of J candidate solutions achieved through the repeated use of K-Means++. Evo-Hood-K-Means is well suited for clustering larger and complex datasets; and c) Evo-Hood-Random-Swap, which benefits from the Hood deterministic seeding of centroids from the population, and improves them through a limited number of stochastic iteration swaps. Evo-Hood-Random-Swap is capable of accurate clustering in large and irregular datasets. All three algorithms exploit the Principal Component Analysis (PCA) [1] for coping with multidimensional datasets (see [23] for details). The algorithms depend on the k input parameter required for driving the KNN strategy, which is typically the subject of a preliminary study that can suggest a useful k range [k_start..k_end] with a suitable k_step. It can be anticipated that Evo-Hood-K-Means can provide the sensitivity analysis, which can conveniently be exploited to reduce the work of Evo-Hood-Random-Swap. The efficacy of the proposed algorithms is demonstrated through a series of clustering experiments, directed to both benchmark and real-world datasets. The implementation of the three algorithms is based on parallel Java [29–31], which can smooth the $O(N^2)$ KNN initialization cost required by Hood-K-Means and the $O((JK)^2)$ implied when using Evo-Hood-K-Means/Random-Swap.

1.7. Paper structure

In Section 1, we review the basic clustering concepts underlying the work reported in the paper. Both the formal terms, the K-Means operation together with the role of the seeding methods, the behavior of Random Swap, and the basic concepts of density-peaks and evolutionary-based clustering are recapitulated. In addition, the paper's contribution is clarified. In Section 2, we discuss some internal and external measures for the assessment of the clustering quality ensured by a user-provided clustering algorithm. In Section 3, we present the three clustering algorithms proposed in this paper, along with an indication of their application areas. In Section 4, we document the effectiveness of the proposed algorithms by reporting the results of a series of clustering experiments, applying the three proposed algorithms to synthetic/benchmark and real-world datasets, with either regular or irregular data distributions, possibly large and multidimensional. In Section 5, we conclude the paper with some points concerning ongoing and future work.

2. Clustering quality measures

A basic measure that captures the quality of a clustering is the *Sum-of-Squared-Error* (SSE), which belongs to the class of internal measures. The SSE expresses a sort of variance of the point distributions around the centroids. Computing the SSE costs $O(N)$ and $O(ND)$ for high-dimensional datasets. *SSE* is the default objective function to optimize (minimize) on any kind of dataset, synthetic or real-world. Another well-known internal measure is the Silhouette index SI [32], which is more accurate at capturing the internal cohesion of the clusters and their external separation. The SI index of a point x is defined as:

$$SI_x = \frac{b_x - a_x}{\max(b_x, a_x)}, \tag{5}$$

where a_x is the average distance of x from the remaining points of the same cluster, and b_x is the minimum average distance of x from the points of other clusters. The overall SI is the average of the SI coefficients of all the points: $SI = \frac{1}{N}\sum_{x_i \in X} SI_{x_i}$. SI can give a better indication than SSE of the clustering accuracy, but it costs $O(N^2)$, which practically forbids its application to large datasets. SI ranges in [-1..1]. Values close to 1 indicate good clustering, with clusters internally cohesive and externally well separated. SI values toward 0 mirror clusters with high overlap. Negative values indicate an erroneous clustering.

Several external measures can be adopted for the assessment of cluster quality [33]. The following recalls some measures, later exploited in this paper, which can be used on datasets equipped with ground truth information (centroids or partition labels). For simplicity, as in [34], ground truth centroids are supposed to be automatically converted into the corresponding ground truth partition labels. The chosen external measures are mostly set-matching and compare the partitions (sets of point indices) of two solutions: one is user-defined through a new algorithm, the other is the available ground truth solution. Such partition vectors will be indicated as P^U and P^G. Although they can have different sizes, the following assumes that the two vectors have the same length K. The definition of the selected external measures can be expressed by first building the so-called contingency matrix CM: P^UxP^G. The item $CM_{i,j}$ is defined as $CM_{i,j} = \left|P_i^U \cap P_j^G\right|$, that is, the intersection of the partition (row) $P_i^U|_{i=1}^K$, and the partition (column) $P_j^G|_{j=1}^K$. Of course: $\sum_{j=1}^K CM_{i,j} = n_i = \left|P_i^U\right|$, $|_{i=1}^K$, and similarly: $\sum_{i=1}^K CM_{i,j} = n_j = \left|P_j^G\right|$. A first measure, which can be defined on the CM, is the so-called $Purity$:

$$Purity = \frac{1}{N}\sum_{i=1}^K \text{argmax}_{j|_1^K}\{CM_{i,j}\}. \tag{6}$$

$Purity$ sums the max elements of the various CM rows; then the sum is averaged over the number N of dataset points. $Purity$ ranges from [0..1]. Values close to 1 mirror good clustering. Intuitively, Purity counts how many points of the user-defined partitions get correctly placed on corresponding ground truth partitions. The complement of Purity denotes the number of dataset points that were incorrectly clustered, a sort of incompleteness degree.

In this paper, as in [34,35], the clustering accuracy index (ACC) is adopted as one of the most indicative clustering measures. In particular, the definition of ACC proposed by Franti and Sieranoja in [36] is adopted. Algorithm 1 briefly recalls, in pseudocode, the definition of ACC.

Algorithm 1. Definition of the ACC accuracy measure [36].

```
ACC( P^U, P^G ){
    acc1 ← matchOneWay( P^U, P^G )
    acc2 ← matchOneWay( P^G, P^U )
    return min( acc1, acc2 )
}
matchOneWay( P^A, P^B ){
    CM ← contingencyMatrix( P^A, P^B )
    mapA2B ← {
        ∀ i ∈ [1..|P^A|], mapA2B[i] <-argmax_j ( CM[i][j] ), j ∈ [1..|P^B|]
    }
    return countMatches( P^A, P^B, mapA2B );
}
countMatches( P^A, P^B, mapA2B )<-{
    count ← 0
    ∀ i ∈ [1..N], i index of a data point in X
        let j be the P^B[j] partition containing i
        let h be the P^A[h] partition containing i
        if( mapA2B[h]==j ) ++count
    return count/N;
}
```

ACC is evaluated in the two mapping directions: $U \rightarrow G$ and $G \rightarrow U$. In each direction, the matchOneWay(P^A, P^B) procedure builds the corresponding CM and initializes the vector mapA2B in such a way that mapA2B[i]=j if the j-th (column) partition of CM holds the maximum value of the i-th partition (row). From the countMatches(…) procedure, one can retrieve the $Purity$ definition between the two partition solutions P^A and P^B. Consequently, the ACC value can equivalently be computed as:

$$ACC = \min\left(Purity(P^U, P^G), Purity(P^G, P^U) \right). \tag{7}$$

The use of the two mapping directions can be understood as a way to detect the worst degree of incompleteness: of the data points when placed by the user algorithm on the ground truth partitions, and of the ground truth partition points as located on the user-provided partitions. ACC values range in [0..1], with values close to 1 which denote accurate clustering.

As a further measure, we consider the Cluster Index (CI) as proposed in [37], when ground truth centroids (GTC) are given, and generalized in [38], when ground truth partition labels (GTP) are furnished. As anticipated in this section, for the sake of simplicity, the same name, CI, is uniformly adopted in both cases, GTC and GTP. In addition, since GTC can immediately be transformed into GTP by applying the nearest centroid rule to dataset points with respect to the ground truth centroids, CI will tacitly be assumed to be the Generalized Centroid Index advocated in [38].

Additionally, the CI measure can be expressed in terms of the contingency matrix CM. In particular, CI can be summarized as:

$$CI = \max\left(\#orphans(P^U \rightarrow P^G), \#orphans(P^G \rightarrow P^U) \right), \tag{8}$$

where the two possible mapping directions are considered. In the first mapping direction, each partition $P_i^U|_{i=1}^K$ (i-th row of CM) is mapped onto the partition $P_j^G|_{j=1}^K$ (j-th column of CM) where the maximum $CM_{i,j}$ occurs. After that, it is possible to compute the Jaccard distance between P_i^U and P_j^G as:

$$Jd\left(P_i^U, P_j^G\right) = 1 - \frac{\max\left(CM_{i,j}\right)}{\left|P_i^U \cup P_j^G\right|}. \quad (9)$$

Of course, the mapping P_i^U onto P_j^G is mirrored by the minimal Jaccard distance between the two partitions. The number of orphans in the $P^U \rightarrow P^G$ direction coincides with the number of ground truth partitions (columns of CM) to which no user-found partition is mapped. Similarly, the number of orphans is computed in the opposite direction. CI ranges in [0..K-1]. A $CI = 0$ characterizes correct clustering. A $CI > 0$ mirrors the number of incorrectly detected clusters.

The algorithms proposed in this paper can adopt either the SSE or (when the dataset comes with ground truth information) the ACC as the objective function cost to optimize (SSE to minimize, ACC to maximize).

Other clustering measures often used include (see [33]) the Adjusted Rand Index (ARI), the Normalized Mutual Information (NMI), and the *F-score*. Details about how to compute them are clarified in [8,39]. These indices also have values that range in [0..1], with values close to 1 indicating good clustering.

3. New proposed algorithms

The following first describes the new Hood seeding method. Then, three clustering algorithms are presented that use Hood for centroid initialization: Hood-K-Means, Evo-Hood-K-Means, and Evo-Hood-Random-Swap.

3.1. The Hood seeding method

The operation of Hood is summarized in Algorithm 2. Hood deterministically generates a vector of K centroids from a source of data points, which can be either the whole dataset or a smaller population used to drive an evolutionary algorithm. Important is the k input parameter, which guides the KNN algorithm. Two phases can be recognized. In the first one, the KNN algorithm is applied to detect, for each point p of the source, the first k distinct distances to the nearest neighbors of p. After that, the KNN neighborhood ($nbKNN$) of p is established, together with the average distance (avd) of the first k nearest neighbor distances, which is stored in p. The second step identifies the K centroids through a deterministic adaptation of the Maximin seeding method (see Section 1.2). The first centroid is identified as the point in the source with minimal avd, which in turn ensures the point has maximal density (see Section 1.4). Then L is set to 1. To proceed with the next centroid, the Jaccard distance (see also Eq (9)) between point neighborhoods ($nbKNN$) is used to quantify neighborhood overlap:

$$JD(p_1, p_2) = 1 - \frac{|p_1.nbKNN \cap p_2.nbKNN|}{|p_1.nbKNN \cup p_2.nbKNN|}. \quad (10)$$

For each point p, its minimum JD to the L existing centroids is computed and saved in p. Then, the next centroid is defined as the point with the maximum JD. After that, L is incremented. The operations continue until all K centroids are in place.

Algorithm 2. Operation summary of the Hood seeding method.

Input: source, k, K

1. apply k-nearest neighbors (KNN) algorithm to each point of source, and determine the neighborhood nbKNN and average distance of the point (avd)
2. define the 1st centroid as the first point, in source, with minimal avd

L←1

do{

for each point p in source, determine the minimal Jaccard distance (JD) of p from the L existing centroids, and save JD in p

choose the next centroid as the first point with maximal JD

L←L+1

}while(L<K)

Output: vector of K centroids extracted from source.

Algorithm 3 reports some details about the implementation of the second step of Hood (see Algorithm 2) in parallel Java, that is, by using parallel streams and lambda expressions [21,29–31]. Source point neighborhoods are kept in the global nbKNN[] array, which is accessed by the Jaccard distance function JD(i1,i2) through the indices of two points. The first part of Algorithm 3 determines the point with minimal avd (maximal density), which serves as the first centroid. This is done in parallel (the global PARALLEL parameter is assumed to be true) on a stream opened on the source and processed by the terminal reduce() operation. A lambda expression is used for making the comparison of point distances (avd). Then, the Jaccard distance from the first centroid is annotated within each point as the minimal distance from the existing centroids. The next centroid is established as the point with the maximal minimal distance. After that, minimal distances from the current centroids are updated. Purposely, the distance field of each point is possibly updated by considering only the distance from the newly added centroid.

The above-described basic Hood seeding method proves effective in many practical cases (see, e.g., the datasets in Sections 4.1 to 4.3) and is applied by default. A more robust version of Hood that guarantees the next centroids are chosen far from existing centroids and with high density, and avoids outliers from being considered as potential centroids, rests on a modified Jaccard distance as follows:

$$mJD(p_1, p_2) = \rho(p_1) * JD(p_1, p_2), \tag{11}$$

where $\rho(p_1)$ is the density of point p_1 estimated as $1/avd(p_1)$, and $JD(p_1, p_2)$ is that of Eq (10). It is worth noting that an outlier has a higher Jaccard distance from current centroids, but it also has a lower density. The modified Jaccard distance has a minor influence on the evolutionary algorithms based on a population of elitist solutions (see later in this paper).

Algorithm 3. Centroid initialization of the Hood seeding method (see step 2 of Algorithm 2).

```
//find point with minimal average distance avd
Stream<DataPoint> pStream=Stream.of( source );
if( PARALLEL ) pStream=pStream.parallel();
DataPoint neutral=new DataPoint(); neutral.setDist( Double.MAX_VALUE );
DataPoint first=pStream
    .reduce( neutral, (p1,p2)->{ return p1.getDist()<p2.getDist()?p1:p2; } );
centroids[0]=new DataPoint( first );
centroids[0].setID( first.getID() );
//set current (minimal) distances from 1st centroid
pStream=Stream.of( source );
if( PARALLEL ) pStream=pStream.parallel();
pStream
    .map( p->{
        p.setDist( JD( p.getID(), centroids[0].getID() ) );
        return p;
    })
    .forEach( p->{} );
//define remaining centroids
int L=1;
do{
    //next centroid as point with maximum of minimal distances
    pStream=Stream.of(source);
    if( PARALLEL ) pStream=pStream.parallel();
    neutral=new DataPoint(); neutral.setDist( Double.MIN_VALUE );
    DataPoint next=pStream
        .reduce( neutral, (p1,p2)->{ return p1.getDist()>p2.getDist()?p1:p2; } );
    centroids[L]=new DataPoint( next );
    centroids[L].setID( next.getID() );
    //update minimal distances by considering the newly added centroid
    final int l=L;
    pStream=Stream.of(source);
    if( PARALLEL ) pStream=pStream.parallel();
    pStream
        .map( p->{
            double jd=JD( p.getID(),centroids[l].getID() );
            if( jd<p.getDist() ) p.setDist( jd );
            return p;
        })
        .forEach( p->{} );
    ++L;
}while( L<K );
```

3.2. Hood-K-means algorithm

This variant of K-Means is reproduced in pseudocode in Algorithm 4. The k range [k_start..k_end] with step k_step is a matter of preliminary analysis, which depends on the dataset. As an example, [2..150] with a step of 1 is often used in practical cases. Hood-K-Means can optimize either the *SSE* or (better in many cases) the *ACC* clustering measure, as in [34,35]. Due to the $O(N^2)$ cost of the Hood seeding, the algorithm is effective in small and medium-sized datasets. The effective Java application also performs, at the time of dataset loading, the required PCA analysis [1,23].

Algorithm 4. Modus operandi of Hood-K-Means algorithm

```
Input: dataset X of N points, K number of clusters, k range [k_start..k_end] with k_step
regulating the KNN technique, T maximum number of iterations of any K-Means execution,
cost SSE or ACC
k←k_start, tot_it←0
do{
    best_cost←?, best_solution←?, best_k←?
    it←0, termination←false
    centroids←Hood(X,k,K) //initialize centroids
    do{
        create-partitions: assign X data points to clusters according to the nearest centroid
            rule nc(p)
        update-centroids: define centroids as the mean points of the resultant clusters
        if( centroids are numerically stable w.r.t. previous centroids or it==T )
            termination←true
        if( !termination ) ++it
    }while( !termination )
    current_solution←<centroids,partitions>
    tot_it←tot_it+it
    current_cost←cost( current_solution )
    if( current_cost improves best_cost ){
        best_cost←current_cost, best_solution←current_solution, best_k←k
    }
    k←k+k_step
}while( k<k_end )
Output: best_solution, best_k, best_cost, other clustering measures, average number of K-Means
iterations
```

Hood-K-Means assumes, as usual, that the number K of centroids is known. In cases where K is unknown, it can be estimated by executing the algorithm for various values of K and by observing the value that minimizes the *SSE* or, better, the one that maximizes the Silhouette index *SI*.

3.3. Evolutionary-Hood-K-means

Evo-Hood-K-Means (see Algorithm 5) is organized in two steps: In step 1), the population $\wp$ consisting of *J* candidate solutions (and J*K centroids) is created; and in step 2), the evolutionary computation is carried out. Both steps can work with either the *SSE* or the *ACC* as the objective cost. The evolutionary part applies Hood-K-Means, which directs seeding to the population $\wp$. For its design,

Evo-Hood-K-Means is adequate for medium-to-large, not irregular, datasets. Due to the behavior of Hood seeding, Evo-Hood-K-Means also ensures deterministic clustering. Re-applying the same k range always generates the same sequence of solutions.

Algorithm 5. Operation of Evo-Hood-K-Means.

```
1) Population setup
Input: dataset X, number of clusters K, number J of population solutions, number Rep of K-Means++ repetitions for defining a population solution, cost SSE or ACC
℘←∅
repeat J times{
    best_cost←?, best_solution←?
    repeat Rep times {
        centroids←K-Means++( X, cost )
        current_solution←K-Means()
        current_cost←cost( current_solution, X )
        if( current_cost improves best_cost ){
            best_cost←current_cost, best_solution←current_solution
        }
    }
    ℘←℘ ∪ { best_solution }
}
2) Evolutionary computation
Input: dataset X, population ℘ of J*K centroids, k range [k_start,k_end] by k_step, cost SSE or ACC
best_cost← ?, best_solution←?, best_k←?
for( k=k_start; k<=k_end; k=k+k_step ){
    current_solution←Hood-K-Means( ℘,k,X )
    current_cost←cost( current_solution,X )
    if( current_cost improves best_cost ){
        best_cost←current_cost, best_solution←current_solution, best_k←k
    }
}
Output: best_solution, best_cost, best_k, other clustering quality measures
```

3.4. Evolutionary-Hood-Random Swap

Complex datasets, with irregular point distributions, are difficult to cluster with either Hood-K-Means or Evo-Hood-K-Means. To cope with these cases, a third algorithm was developed, in which K-Means local search is replaced by Random Swap [20,21] global search. However, the evolutionary organization underlying Evo-Hood-K-Means remains the key to reducing the computational burden of Random Swap by restricting centroid swaps to occur with population points, and not dataset points. Of course, the use of Random Swap forces the clustering to behave stochastically, and multiple runs can be required for the statistical assessment of the quality measures of a clustering solution. Hood is used again for seeding from the population. The good centroid initialization, though, ensured by Hood, is capable of reducing the number T of Random Swap iterations required to converge toward a good solution. Often, a $T=10^3$ value or smaller can be used.

The logical behavior of Evo-Hood-Random-Swap is sketched in Algorithm 6. The population setup phase coincides with step 1 of the Evo-Hood-K-Means in Algorithm 5. The synergy due to the use of the same evolutionary background makes it possible to exploit Evo-Hood-K-Means for a preliminary assessment of the sensitivity analysis of the population to a suitable k range of values, which can then be perfected by Evo-Hood-Random-Swap.

Algorithm 6. Operation of the Evo-Hood-Random-Swap.

2) *Evolutionary computation*

```
Input: dataset X, population ℘ of J*K centroids, k range [k_start,k_end] by k_step, cost SSE or
ACC, number T of Random Swap iterations
best_cost←?, best_solution←?, best_k←?
for( k=k_start; k<=k_end; k=k+k_step ){
    centroids←Hood( ℘,k )
    //assign X points to clusters according to nearest centroid rule
    partitions←partitioning( centroids,X )
    best_rs_solution←<centroids,partitions>, best_rs_cost←cost( best_rs_solution )
    accepted←true
    for( T times ){
        if( accepted ) save( best_rs_solution )
        j←uniform_random(1..K), i←uniform_random(1..J*K)
        replace j-th centroid with i-th point of population //make a swap
        //refine current_rs_solution with 2 K-Means iterations
        current_rs_solution←K-Means(2)
        current_rs_cost←cost( current_rs_solution )
        if( current_rs_cost improves best_rs_cost ){
            accepted←true
            best_rs_solution←current_rs_solution
            best_rs_cost←current_rs_cost
        }
        else{
            accepted←false
            //restore previous centroids & partitions
            best_rs_solution←restore( previous_rs_solution )
            best_rs_cost←cost( best_rs_solution)
        }
    }
    current_solution←best_rs_solution
    current_cost←best_rs_cost
    if( current_cost improves best_cost ){
        best_solution←current_solution, best_cost←current_cost, best_k←k
    }
}
Output: best_solution, best_cost, best_k, other clustering quality measures
```

Random Swap starts with a Hood centroid initialization built from the population. Such an initial solution is completed by associating the corresponding partition labels with the centroids. As shown in Algorithm 6, two loops are nested. The outer loop is regulated by the assumed range [k_start..k_end]

of values, where possibly k_start can be equal to k_end. The inner loop realizes the *T* Random Swap iterations. The two loops are, separately, responsible for maintaining and improving, according to the *SSE* or *ACC* adopted cost, a best solution. Random Swap elaborates its best_rs_solution, which is initialized with the Hood seeding, and hopefully improved by swap iterations. The resultant best_rs_solution becomes the current_solution at the time the higher k-level loop is resumed. The k-level loop is responsible for keeping the global best_solution updated.

For each k value, T Random Swap iterations are executed. At each swap iteration, the last accepted best_rs_solution (complete with centroids and partitions) is saved before defining, through a swap, the next current_rs_solution. The index j of a randomly chosen centroid and the index i of a randomly selected point in the population are used to replace the j-th centroid with the i-th population point, thus creating a new centroid configuration which gets refined by a few K-Means iterations, responsible for associating with centroids the corresponding partition labels. This way, the current_rs_solution gets defined. The cost of the current_rs_solution is then compared with that of the existing best_rs_solution. In the case of improvement, the current_rs_solution is accepted and replaces the best_rs_solution. Otherwise, the saved best_rs_solution is restored in view of the next swap iteration.

Although not explicit in Algorithm 6, the population points that were used to compose the Hood initial centroids are skipped during the swap operations.

As is the case with the other algorithms, the *ACC* cost can be preferable when studying complex datasets. However, even when the *SSE* cost is adopted, in the k-level loop, the *ACC* cost is used to assess an improvement of the current_solution vs. the best_solution. This provision permits observing a property verified by some irregular datasets: good clustering does not always occur at the minimal *SSE* value.

A subtle point affecting the population setup and the execution, e.g., of the Evo-Hood-Random-Swap, is concerned with the handling of wrong clustering solutions, made up of some empty clusters. When one such solution is sensed, it is abandoned and rebuilt with the same value of k.

4. Clustering experiments

The Hood-based clustering algorithms were thoroughly tested on several benchmark datasets. The first experiments were conducted on synthetic datasets from the Franti repository [40]. The selected datasets come with ground truth information. Ground truth centroids are automatically converted to ground truth partition labels by assigning dataset points to the nearest centroids. The clustering experiments are executed on a Windows 11 Pro desktop, a Dell XPS 8940, with an Intel i7-10700 (8 physical + 8 virtual cores), CPU @ 2.90 GHz, 32 GB RAM, using Java 25.

4.1. The 2-d A and S families of datasets [40]

The A family (see Figures 1 to 3) consists of three datasets that are in the relationship: $A_1 \subset A_2 \subset A_3$. Such datasets are characterized by the varying number of spherical clusters: K=20, 35, and 50. The S family (see Figures 4 to 7) admits four datasets S1 to S4, all having N=5000 data points, with K=15 Gaussian clusters with different degrees of cluster overlap and spatial complexity. The A and S datasets were clustered by first scaling the point coordinates by the overall maximum.



Figure 1. The A1 dataset, N=3000, K=30.



Figure 2. The A2 dataset, N=5250, K=35.

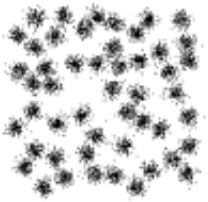


Figure 3. The A3 dataset, N=7500, K=50.

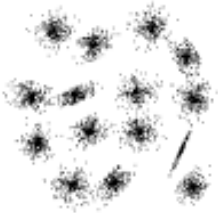


Figure 4. The S1 dataset.



Figure 5. The S2 dataset.

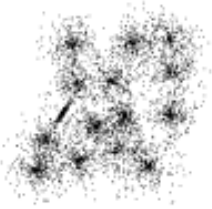


Figure 6. The S3 dataset.



Figure 7. The S4 dataset.

The A datasets are rather regular. The S datasets, instead, were more challenging due to the Gaussian clusters. Both kinds of datasets were clustered using the Hood-K-Means algorithm. Each clustering experiment was split into two runs: The first was devoted to detecting the k value that optimized the cost function. A k range of [2..200] with a step of 1 was assumed, with the ACC cost to maximize. The second run was driven by the optimal k value and aimed to provide various clustering measures, like the ACC, SSE, SI, CI, $Purity$, and so forth. Table 1 reports the collected results, including the number of refinement iterations executed by K-Means (KM_it). The Elapsed Time (ET) in sec refers to the second run only. The $Purity$ column coincides with that of the ACC, testifying to their close bond. The achieved $CI = 0$ mirrors correct clustering.

Table 1. Hood-K-Means clustering results of A and S datasets [40].

Dataset	k_opt	ACC	SSE	SI	CI	Purity	KM_it	ET(sec)
A1	118	0.9990	2.8281	0.5950	0	0.9990	6	0.48
A2	88	0.9989	4.7235	0.5975	0	0.9989	7	0.68
A3	118	0.9995	6.7377	0.6008	0	0.9995	8	1.25
S1	123	0.9990	9.4630	0.7113	0	0.9990	4	0.49
S2	180	0.9968	13.6991	0.6262	0	0.9968	8	0.67
S3	111	0.9804	18.8209	0.4924	0	0.9804	17	0.48
S4	183	0.9714	16.4507	0.4800	0	0.9714	16	0.74

Besides the clustering measures reported in Table 1, good clustering can also be observed in Figures 8 and 9, which portray the centroids (in red, the ground truth, in black, those generated by Hood-K-Means) and the partitions of the A1 and the S4 datasets, respectively.

It is worth noting that, to overcome the optimistic cluster order of the points of S1–S4, each such dataset was clustered by first generating a shuffled version of the dataset. Hood-K-Means, though, was able to produce very similar results (as in Table 1) in the optimistic and shuffled order of the points.

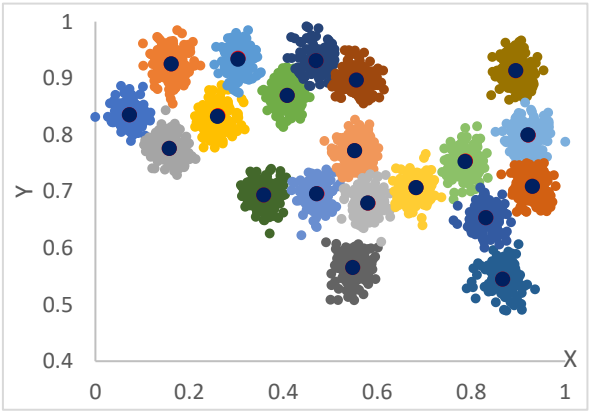


Figure 8. Partitions and centroids of A1.

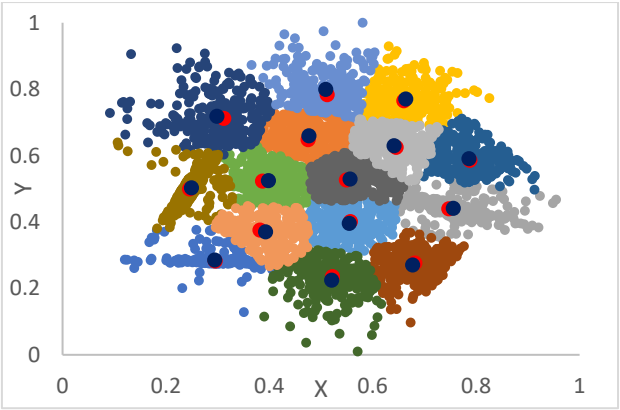


Figure 9. Partitions and centroids of S4.

For comparison, the A and (shuffled) S datasets were also clustered using, separately, K-Means with K-Means++ seeding and Repeated K-Means, with 100 repetitions, each seeded independently using the uniform random seeding method. Results are collected in Tables 2 and 3, respectively.

Table 2. Clustering results of A and S datasets [40] using K-Means with K-Means++ seeding.

Dataset	ACC	SSE	SI	CI	Purity	KM_it	ET(sec)
A1	0.899	3.9389	0.5330	2	0.9157	14	0.025
A2	0.908	6.9479	0.5365	3	0.9389	14	0.052
A3	0.8975	9.8243	0.5360	5	0.9321	35	0.070
S1	0.9360	14.0452	0.6641	1	0.9780	24	0.049
S2	0.9254	18.0833	0.5737	1	0.9618	12	0.029
S3	0.9046	21.0204	0.4706	1	0.9318	13	0.029
S4	0.7654	21.9755	0.4259	3	0.8460	25	0.049

Table 3. Clustering results of A and S datasets [40] using 100 repetitions of K-Means with uniform random seeding.

Dataset	ACC	SSE	SI	CI	Purity	KM_it	SR	ET(sec)
A1	0.9507	3.2748	0.5677	1	0.964	22.05	0	0.754
A2	0.9688	5.2093	0.5785	1	0.9770	26.85	0	1.762
A3	0.9363	8.5931	0.5534	3	0.9628	28.18	0	3.318
S1	0.9992	9.4630	0.7113	0	0.9992	19.22	6%	1.159
S2	0.9962	13.6990	0.6261	0	0.9962	24.77	11%	1.157
S3	0.9818	18.8204	0.4925	0	0.9818	26.30	13%	1.280
S4	0.9624	16.4445	0.4799	0	0.9624	37.18	27%	1.367

As shown in Table 2, no dataset was correctly clustered by K-Means++; CI is always greater than 0, and all the other clustering measures registered lower values with respect to the Hood-K-Means results in Table 1. Better behavior was observed with Repeated K-Means, where only the S datasets were correctly clustered. In Table 3, the column KM_it refers to the number of iterations executed by K-Means, averaged over the 100 repetitions. The column SR (success rate) indicates the number of repetitions that terminated with $CI = 0$. Better results in Table 3 are a consequence of the greater cluster overlap degree in the S datasets, particularly for the S4 dataset. However, Hood-K-Means could outperform, in most cases, K-Means++ and Repeated K-Means, while requiring a small execution time.

4.2. G2 datasets

These datasets (see Figure 10) were made up of Gaussian clusters with a varying degree of overlap (variance) and number of dimensions (point coordinates), D. The family was characterized by N=2048, D=2 or 1024, K=2, and variance in [10..100], indicated at the end of the dataset name. The datasets were clustered, without scaling, by Hood-K-Means, and the results are shown in Table 4. In all cases, the PCA confirmed the number of coordinates as 2 or 1024.

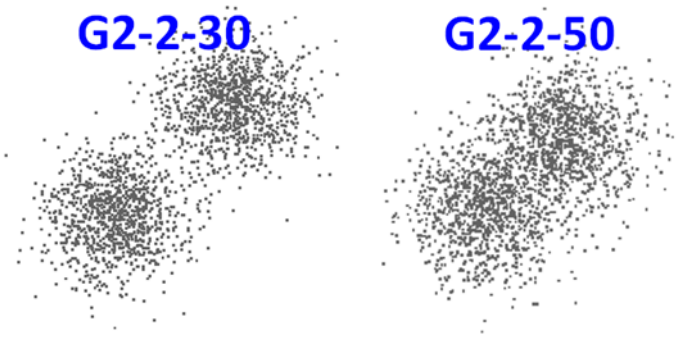


Figure 10. Two example datasets of the G2 family [40].

Table 4. Hood-K-Means clustering results of the G2 dataset family [40].

Dataset	k_opt	ACC	SSE	SI	CI	Purity	ET(sec)
G2-2-10	2	1.0	418803.1689	0.8734	0	1.0	0.12
G2-2-30	2	1.0	3652317.4556	0.6290	0	1.0	0.12
G2-2-50	2	0.9985	8874807.3844	0.4815	0	0.9985	0.13
G2-2-70	2	0.9702	1.6169×10^{7}	0.3977	0	0.9702	0.12
G2-1024-10	2	1.0	8.2834×10^{8}	0.7282	0	1.0	0.56
G2-1024-100	2	1.0	2.0730×10^{10}	0.1831	0	1.0	0.52

4.3. DIM datasets

As another example of the application of basic Hood-K-Means, the DIM set of datasets was chosen. Each dataset had N=1024 points, D=32 to 1024 dimensions, and K=16 well-separated Gaussian clusters. The results, achieved with no data scaling, are reported in Table 5. The *SI* values confirmed well-separated clusters. Table 5 also shows the number of point coordinates that are principal components and are involved in the Euclidean distance calculation.

Table 5. Hood-K-Means clustering results of the DIM dataset family [40].

Dataset	k_opt	ACC	SSE	SI	CI	Purity	PCA	ET(sec)
DIM032	2	1.0	232535.1094	0.9456	0	1.0	32	0.12
DIM064	12	1.0	213247.4219	0.9665	0	1.0	63	0.13
DIM128	10	1.0	270884.4688	0.9747	0	1.0	126	0.14
DIM256	7	1.0	236583.8750	0.9830	0	1.0	251	0.12
DIM512	7	1.0	291114.0156	0.9868	0	1.0	502	0.14
DIM1024	9	1.0	269777.3594	0.9909	0	1.0	1003	0.21

4.4. Birch datasets

There were three datasets (see Figures 11 to 13), all with N=105, D=2, and K=100 clusters. Birch1 and Birch2 admitted spherical clusters regularly distributed, respectively, over a 10x10 grid and along a sine curve. Birch3, instead, was composed of clusters with random sizes, randomly located in the data space. Birch3 was the most difficult to cluster. Despite their regularity, Birch1 and Birch2 were better handled by the evolutionary versions of the Hood-based algorithms, which depended on a small-sized population of elitist candidate solutions. In particular, a population of J=20 solutions, each selected as the best one (according to ACC) in a batch of Rep=10 repetitions of K-Means++, was created. Birch datasets were clustered by first scaling the point coordinates by the overall maximum. Birch1 was processed by Evo-Hood-K-Means, whereas Birch2 was handled by Evo-Hood-Random-Swap with the SSE function cost and T=1000 swaps. The population setup took a runtime of about 7 min for Birch1 and about 3.3 min for Birch2. A range of [2..30] was used to detect the k value that optimizes the ACC cost. The results reported in Table 6 comply with similar results documented, e.g., in [8,21,38].

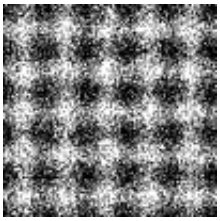


Figure 11. Birch1 dataset [40].



Figure 12. Birch2 dataset [40].

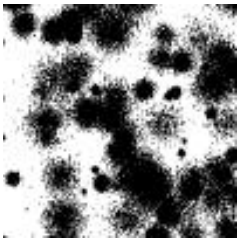


Figure 13. Birch3 dataset [40].

Table 6. Birch1 and Birch2 clustering results.

Dataset	k_opt	ACC	SSE	SI	CI	Purity	PCA	ET(sec)
Birch1	9	0.9950	92.7729	0.4598	0	0.9950	2	1.47
Birch2	5	0.9984	0.2284	0.7827	0	0.9984	1	56.96

Table 7 shows the *ACC* and *SSE* observed values when using Evo-Hood-K-Means with the Birch3 dataset. A k range [50..150], with a step of 10, was used, with ACC as the cost to optimize. Table 7 confirms that in irregular datasets such as Birch3, best clustering (max *ACC*) does not necessarily

correspond to the minimal SSE value. For completeness, Evo-Hood-K-Means suggested a $CI = 17$ for k=60. To improve clustering results, we ran Evo-Hood-Random-Swap 10 times with T=1000 swaps, k=60, and the SSE cost. Table 8 reports confidence intervals (based on a 95% confidence level and Student's T-distribution) for the clustering measures. A CI of 11 is documented in several papers in the literature (see [8,21,39]). The average value $CI = 10.5$ is an improvement.

Table 7. Birch3 ACC/SSE values by using Evo-Hood-K-Means with cost ACC and k range [50..150] by step 10.

k	ACC	SSE
50	0.7752	54.5263
60	0.8176	52.7404
70	0.7636	52.7404
80	0.8076	54.0941
90	0.8015	49.8327
100	0.7832	54.8643
110	0.7785	53.9515
120	0.7983	53.6778
130	0.8134	47.7530
140	0.8051	50.7811
150	0.8021	49.3026

Table 8. Birch3 clustering results of 10 runs of Evo-Hood-Random-Swap with k=60.

ACC	SSE	SI	CI	Purity	ET(sec)
0.8679±0.0019	37.9079±0.0354	0.5247±0.0010	10.5000±0.4099	0.8684±0.0020	626.19

We studied the Birch datasets using a population with J=20 candidate solutions, i.e., a reduced dataset of J*K centroid points. A good J value distributed the population centroids to create significant density peaks around the ground-truth centroids. Table 9 reports the population setup, for J varying from 5 to 25, for the Birch2 dataset. For each J value, Evo-Hood-K-Means was executed on the population using a k range of [2..50] with step 1, and the k value that optimized ACC was recorded. Table 9 also shows the CI corresponding to the optimal k. The Elapsed Time (ET), in sec, refers to the creation of the population. Table 9 confirms that J=20 is adequate for the evolutionary study of Birch2.

J=20 proves suited for all the evolutionary clustering experiments reported later in this paper.

Table 9. Sensitivity analysis of the Birch2 dataset to the J parameter.

J	k_opt	ACC	CI	ET (sec)
5	23	0.9162	8	46.04
10	4	0.9381	6	96.56
15	2	0.9662	3	130.96
20	5	0.9878	1	192.98
25	4	0.9877	1	260.20

4.5. The shape set of datasets [40]

This set of 2-d datasets (see Table 10), although small in size, was characterized by clusters with particular shapes and positions. They were often used to check density-peak special-purpose algorithms (see, e.g., ParDP in [23]). They were studied using Evo-Hood-K-Means without scaling, by preparing a population of J=20 solutions, each being the best of 10 repetitions of K-Means++. The

ACC cost was used during the population setup and for driving the executions of Evo-Hood-K-Means. The collected results are presented in Table 11. The CI column, with all 0s, confirms the structurally correct clustering for all the datasets.

Table 10. Shape set of datasets [40].

Dataset	N	D	K
Aggregation	788	2	7
PathBased	300	2	3
Spiral	312	2	3
D31	3100	2	31
R15	600	2	15
Jain	373	2	2
Flame	240	2	2

Table 11. Evo-Hood-K-Means results about the Shape datasets of Table 10.

Dataset	k_opt	ACC	SSE	SI	CI	Purity	ET(sec)
Aggregation	10	0.8794	11266.8561	0.4917	0	0.8794	0.03
PathBased	2	0.7467	8980.9873	0.5406	0	0.7633	0.02
Spiral	2	0.3558	13310.0339	0.3581	0	0.3590	0.02
D31	4	0.9777	3393.3917	0.5755	0	0.9777	0.12
R15	2	0.9967	108.6190	0.7527	0	0.9967	0.06
Jain	2	0.7855	22209.2456	0.4952	0	0.7855	0.02
Flame	3	0.8542	3147.5389	0.3768	0	0.8542	0.02

For brevity, Figures 14 to 17 depict graphically the Evo-Hood-K-Means clustering achieved for four shape datasets. Detected centroids are black colored, and (when available) ground truth centroids are red colored.

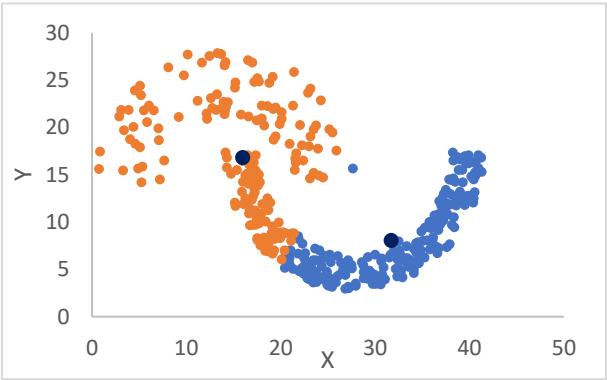


Figure 14. Partitions and centroids of Jain.

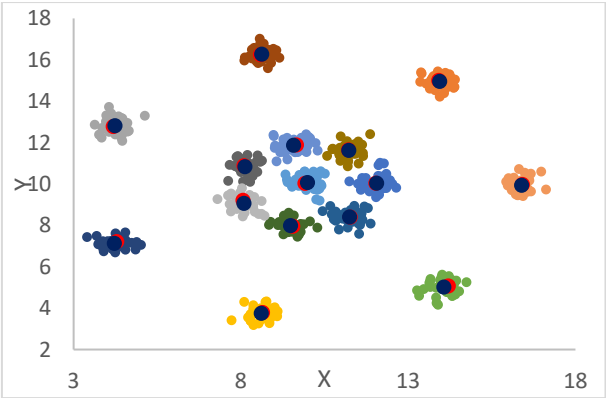


Figure 15. Partitions and centroids of R15.

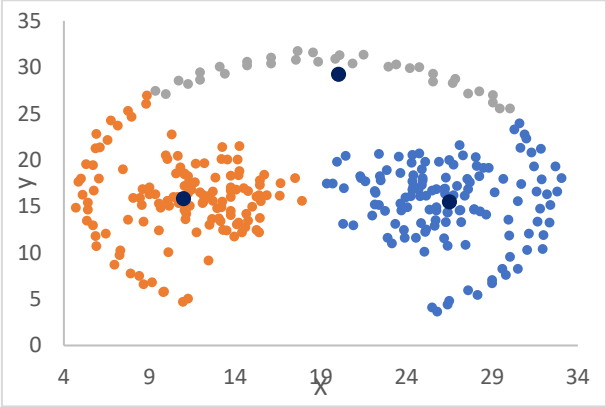


Figure 16. Partitions and centroids of PathBased.

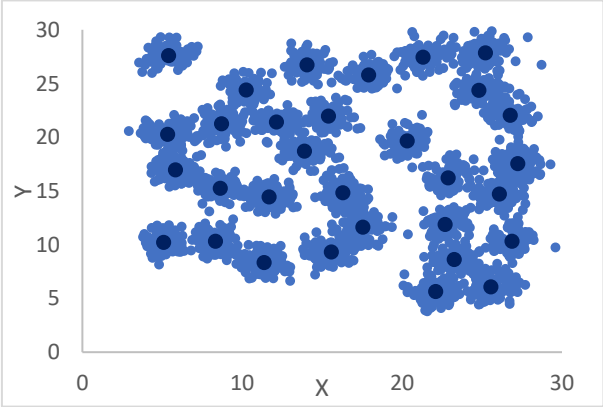


Figure 17. Detected centroids of D31.

4.6. The Worms datasets [40]

There were two synthetic datasets: Worms_2d (see Figure 18) and Worms_64d, whose complete descriptions were: N=105600, D=2, and K=35 for Worms_2d, and N=105000, D=64, and K=25 for Worms_64d. Worms_2d contained 35 shapes that were movement paths in the 2-D space. Worm shapes began at a random position and moved in a random direction. At each step, points followed a Gaussian distribution, which generated a cloud around the current position. Variance increased gradually at each step; the movement direction changed according to an orthogonal direction. In the 64D case, the orthogonal direction was chosen randomly at each step.

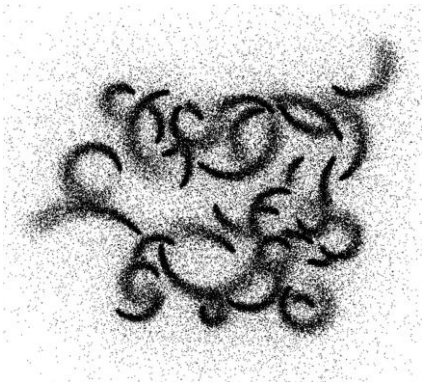


Figure 18. The Worms_2d dataset [40].

The worm datasets are often considered due to the difficulty of clustering them. As confirmed in the literature (see, e.g., [21]), Worms_64d, despite its 64 features per point, is more amenable to clustering. Worms_2d, instead, is the hardest. No full solution, one with $CI = 0$, is documented in the literature. For example, in [22], a solution with an average CI of 7.5 is reported. The two worm datasets were studied in this paper by first scaling (as in [39]) point coordinates by the overall maximum, using Evo-Hood-K-Means to determine the optimal k value where the highest *ACC* occurs, and, ultimately, by Evo-Hood-Random-Swap, which improves a solution at the optimal k. In particular, 10 runs of each dataset were executed, using the SSE cost function and T=1000 swaps. Table 12 reports the observed results. The average value $CI = 7.3$ for Worms_2D is better than the value $CI = 8$ reported in [8] and the average value $CI = 7.5$ documented in [22]. In addition, the average value $CI = 0.2$ for Worms_64D confirms that the recurrent value of CI in the various runs is 0, thus witnessing correct clustering as in, e.g., [8]. The time required for preparing the population (J=20, Rep=10 of K-Means++) was 155.54 sec for Worms_2d, and 341.32 sec for Worms_64d. The PCA analysis confirmed the point coordinates.

Table 12. Evo-Hood-Random-Swap results of 10 runs of the Worms datasets [40].

Dataset	k_opt	ACC	SSE	SI	CI	Purity	ET(sec)
Worms_2d	20	0.5273±0.0062	79.4814±0.1004	0.3603±0.0014	7.3000±0.2800	0.5293±0.0074	357.54
Worms_64	5	0.8627±0.0085	12162.2421±8.5162	0.0862±0.0072	0.2000±0.2444	0.8692±0.0004	2855.33

4.7. The Olivetti dataset [41]

The Olivetti realistic dataset (N=400, D=4096, K=40), from AT&T Laboratories Cambridge and available as in the reference [41], is a challenging multidimensional facial recognition problem, where 40 human subjects, each represented in 10 different photos, have to be identified from the images. Each photo consists of 64×64 = 4096 pixels. In [39], a $CI = 6$ was achieved; that is, 6 subjects were not recognized by the clustering algorithm.

Olivetti was clustered by Evo-Hood-Random-Swap, without scaling the point coordinates, with T=1000 swap iterations, and *ACC* as the cost to optimize. The ground truth information provided in [6] was used. The PCA analysis detected 4010 of 4096 as principal components. The population setup was completed in about 38 sec.

The Evo-Hood-K-Means, executed with the range [2..100], suggested k=7 as the value that optimizes *ACC* to the value of 0.6675. Thirty runs of Evo-Hood-Random-Swap were executed with k=7 and ACC as the cost to improve. The observed confidence intervals (Normal-distribution) are reported in Table 13.

An average value of $CI = 3.73$ for Olivetti, that is, at most 4 subjects were wrongly recognized, significantly improves similar results reported in the literature.

Table 13. Clustering results of 30 runs of Evo-Hood-Random-Swap on the Olivetti dataset.

ACC	SSE	SI	CI	Purity	ET(sec)
0.7677±0.0052	12135.9493±32.2088	0.1360±0.0021	3.7333±0.2962	0.7693±0.0055	532.58

4.8. Realistic datasets [42]

The sixteen realistic datasets (see Table 14) proposed in [42] to demonstrate the efficiency and accuracy of a new formulation of the K-Means objective function as a trace maximization problem, here referred to as Nie-K-Means, and explored, e.g., in [39], were clustered by the Hood algorithms for comparison with [42] and [39]. Some datasets came from the UCI repository [43]. The chosen multidimensional datasets, listed in Table 14 by ascending number of point coordinates, could be challenging to cluster. The datasets were studied by first creating a population with J=20 solutions, each emerging as the best one (from the *ACC* viewpoint) from Rep=10 repetitions of K-Means++. Then, Evo-Hood-K-Means was used to check the best k value (k_opt) in [2..100] that maximized the *ACC*. Finally, Evo-Hood-Random-Swap was used to improve the solution corresponding to k_opt. The clustering results reported in Table 15 were achieved using, in almost all cases, 30 executions of Evo-Hood-Random-Swap using the ACC function cost. Only for the more expensive PalmData25, Binalpha, Corel_5k, and Mpeg7 datasets, 10 executions were used. The number T of swaps was chosen to be close to J*K. For the heavier four datasets, T=1000 was adopted. For comparison with the results in [42], the SI, ACC, NMI, $F-score$, and the ARI measures are shown. In addition, even though the authors in [42] do not show them, Table 15 includes the Centroid Index (CI) and the number of principal components detected by PCA analysis [1,23].

Table 14. Sixteen realistic datasets [42].

Dataset	N	D	K
Iris	150	4	3
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	36771	10

The results of Table 15 significantly outperform the results reported in [42] in all the clustering measures. In addition, the Evo-Hood-Random-Swap-generated results also improve the good results documented in [39]. In particular, whereas in [39] a $CI = 0$ was registered only for the Iris dataset,

correct clustering, as expressed by an average CI less than 1, is ensured in Table 15, also for the Dermatology, Uspst, USPSdata_20, USPSdata, and MSRA25 datasets. Furthermore, in cases where an average value $CI > 1$ was observed, a smaller value occurred with respect to the results in [39].

In light of the presented experimental work, Hood-based clustering algorithms appeared to be capable of generating good clustering on synthetic and real-world datasets, also in the presence of general, irregular point distributions.

Table 15. Clustering results of the datasets in Table 14 obtained by Evo-Hood-Random-Swap.

Dataset	k_opt	SI	CI	ACC	NMI	F-score	ARI	PCA	ET(sec)
Iris	2	0.5520± 0.0007	0.0000± 0.0000	0.8911± 0.0011	0.7528± 0.0028	0.8175± 0.0016	0.7256± 0.0024	4	0.38
Balance	25	0.1727± 0.0011	1.0000± 0.0000	0.5522± 0.0108	0.1577± 0.0185	0.4883± 0.0112	0.1776± 0.0184	4	0.84
Dermatology	38	0.2205± 0.0002	0.0000± 0.0000	0.9471± 0.0006	0.9004± 0.0007	0.9207± 0.0007	0.9011± 0.0008	34	1.31
Uspst	95	0.1637± 0.0025	0.5667± 0.2034	0.7395± 0.0067	0.6325± 0.0030	0.6109± 0.0060	0.5640± 0.0067	226	25.25
USPSdata_20	2	0.1599± 0.0012	0.3333± 0.1716	0.7267± 0.0069	0.6411± 0.0046	0.6071± 0.0065	0.5601± 0.0072	226	19.01
USPSdata	63	0.1607± 0.0011	0.3667± 0.1754	0.7421± 0.0072	0.6319± 0.0039	0.6150± 0.0066	0.5687±0. 0073	226	324.70
MSRA25	44	0.1812± 0.0024	0.8000± 0.1456	0.6180± 0.0019	0.6461± 0.0028	0.4698± 0.0043	0.4170± 0.0051	240	22.01
PalmData25	8	0.3104± 0.0016	4.1000± 0.5075	0.9053± 0.0034	0.9596± 0.0020	0.8563± 0.0053	0.8549± 0.0054	252	213.93
Binalpha	22	0.0602± 0.0008	3.6000± 0.2993	0.5623± 0.0054	0.6317± 0.0029	0.3766± 0.0048	0.3584± 0.0049	315	59.49
Ecoli	51	0.0288± 0.0019	1.9333± 0.2474	0.8620± 0.0055	0.7183± 0.0087	0.8383± 0.0059	0.7780± 0.0080	339	4.53
Corel_5k	43	0.0852± 0.0021	16.6000 ±1.0653	0.1943± 0.0028	0.2724± 0.0017	0.0848± 0.0006	0.0637± 0.0008	183	252.64
MnistData_05	2	0.0718± 0.0007	1.8000± 0.1456	0.6093± 0.0079	0.5206± 0.0072	0.4663± 0.0083	0.4036± 0.0094	434	98.17
MnistData_10	17	0.0730± 0.0007	1.7000± 0.1668	0.5947± 0.0048	0.5085± 0.0036	0.4534± 0.0045	0.3885± 0.0052	434	19.12
Coil20Data_25	2	0.2133± 0.0020	2.4667± 0.1816	0.7794± 0.0042	0.8226± 0.0032	0.6970± 0.0052	0.6804± 0.0056	893	89.49
Mpeg7	11	0.1066± 0.0011	8.5000± 0.4926	0.6524± 0.0038	0.7698± 0.0020	0.4425± 0.0069	0.4336± 0.0071	5248	2056.03
TDT2_10	7	0.1573± 0.0336	1.6333± 0.2737	0.4890± 0.0153	0.4936± 0.0153	0.2725± 0.0091	0.1367± 0.0123	11028	929.56

5. Conclusions

In this paper, we describe a new seeding method called Hood for the initialization of centroids to be refined by K-Means [9–11] or one of its variants. Hood first identifies, with the KNN technique [22,27], the k-neighborhoods of the points of a dataset. Then centroids are defined by an adaptation of the Maximin method [18,17]. The first centroid coincides with the point with maximum density. Next, centroids are selected, one at a time, by the Maximin strategy using, as the distance of points from the existing centroids, the Jaccard distance of the relevant k-neighborhoods. Hood is implemented in parallel Java [21,29–31] and has the potential for careful clustering.

Three algorithms were developed based on Hood. The first one (Hood-K-Means) uses Hood to drive the local search of classical K-Means. This algorithm is adequate for the management of medium-sized datasets, with regular point distributions. The second and third algorithms, Evo-Hood-K-Means and Evo-Hood-Random-Swap, depend on an evolutionary organization, which rests on a preliminarily built population of candidate elitist solutions achieved by K-Means++ [19]. The population acts as a special reduced dataset where the centroids of the solutions tend to thicken, as confirmed experimentally, around optimal or ground truth centroids. K-Means and Random Swap were adapted to seed initial centroids from the population according to Hood, and to determine, through refinements, a best solution, according to either the *Sum-of-Squared-Error* (*SSE*) objective cost or, when ground truth information is available, the clustering accuracy index (*ACC*) [36], by varying the input k parameter, which regulates the KNN algorithm, in a given range. Evo-Hood-K-Means and, particularly, Evo-Hood-Random-Swap are suited to handle large and multidimensional datasets, with general and irregular point distributions. In reality, a synergy is established between these two algorithms, where Evo-Hood-K-Means is first used to detect the k value that optimizes the assumed objective function cost. After that, some runs of Evo-Hood-Random-Swap are executed with the optimal k value to improve the solution and meet a good value of standard clustering measures. Due to the benefit of Hood initialization, a limited number of swap iterations are normally required in the execution of Evo-Hood-Random-Swap.

In this paper, we describe the design of the Hood-based clustering algorithms. Then, several clustering experiments, based on synthetic and realistic non-trivial datasets, are reported, which confirm the effectiveness of the developed algorithms.

The prosecution of the research work is aimed at addressing the following points: First, to improve the Java implementation of the Hood-based algorithms and port them to Python. Second, to exploit Hood in the development of a further evolutionary algorithm based on the Hartigan-Wong [39] variant of K-Means. Third, to accomplish an in-depth performance comparison with competitor algorithms using complex datasets.

Use of AI tools declaration

The authors declare he has not used Artificial Intelligence (AI) tools in the creation of this article.

Conflict of interest

The author declares no conflict of interest.

References

1. R. Wang, *Introduction to machine learning: from math to code*, Cambridge: Cambridge University Press, 2026. https://doi.org/10.1017/9781009023870
2. F. Nielsen, Partition-based clustering with k-means, In: *Introduction to HPC with MPI for data science*, Cham: Springer International Publishing, 2016, 163–193. https://doi.org/10.1007/978-3-319-21903-5_7
3. B. Everitt, S. Landau, M. Leese, D. Stahl, *Cluster analysis*, Hoboken: John Wiley & Sons, Inc., 2011. https://doi.org/10.1002/9780470977811
4. A. Rodriguez, A. Laio, Clustering by fast search and find of density peaks, *Science*, **344** (2014), 1492–1496. https://doi.org/10.1126/science.1242072
5. P. Fränti, Genetic algorithm with deterministic crossover for vector quantization, *Pattern Recogn. Lett.*, **21** (2000), 61–68. https://doi.org/10.1016/S0167-8655(99)00133-6
6. C. Baldassi, Recombinator-k-means: an evolutionary algorithm that exploits k-means++ for recombination, *IEEE Trans. Evolut. Comput.*, **26** (2022), 991–1003. https://doi.org/10.1109/TEVC.2022.3144134
7. L. Nigro, F. Cicirelli, Improving clustering accuracy of K-means and Random Swap by an evolutionary technique based on careful seeding, *Algorithms*, **16** (2023), 572. https://doi.org/10.3390/a16120572
8. L. Nigro, F. Cicirelli, F. Pupo, Genetic elitist approach and density peaks to improve K-Means clustering, *Algorithms*, **19** (2026), 131. https://doi.org/10.3390/a19020131
9. J. MacQueen, Some methods for classification and analysis of multivariate observations, *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, 281–297.
10. S. Lloyd, Least squares quantization in PCM, *IEEE Trans. Inform. Theory*, **28** (1982), 129–137. https://doi.org/10.1109/TIT.1982.1056489
11. A. Jain, Data clustering: 50 years beyond k-means, *Pattern Recogn. Lett.*, **31** (2010), 651–666. https://doi.org/10.1016/j.patrec.2009.09.011
12. S. Dasgupta, The hardness of K-means clustering, Technical Report CS2008-0916.
13. M. Mahajan, P. Nimbhorkar, K. Varadarajan, The planar k-means problem is NP-hard, *Theor. Comput. Sci.*, **442** (2012), 13–21. https://doi.org/10.1016/j.tcs.2010.05.034
14. M. Emre Celebi, H. Kingravi, P. Vela, A comparative study of efficient initialization methods for the k-means clustering algorithm, *Expert Syst. Appl.*, **40** (2013), 200–210. https://doi.org/10.1016/j.eswa.2012.07.021
15. P. Fränti, S. Sieranoja, K-means properties on six clustering benchmark datasets, *Appl. Intell.*, **48** (2018), 4743–4759. https://doi.org/10.1007/s10489-018-1238-7
16. P. Fränti, S. Sieranoja, How much can k-means be improved by using better initialization and repeats? *Pattern Recogn.*, **93** (2019), 95–112. https://doi.org/10.1016/j.patcog.2019.04.014
17. A. Vouros, S. Langdell, M. Croucher, E. Vasilaki, An empirical comparison between stochastic and deterministic centroid initialisation for K-means variations, *Mach. Learn.*, **110** (2021), 1975–2003. https://doi.org/10.1007/s10994-021-06021-7
18. T. Gonzalez, Clustering to minimize the maximum inter cluster distance, *Theor. Comput. Sci.*, **38** (1985), 293–306. https://doi.org/10.1016/0304-3975(85)90224-5
19. D. Arthur, S. Vassilvitskii, k-means++: the advantages of careful seeding, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007, 1027–1035.

20. P. Fränti, Efficiency of Random Swap algorithm, *J. Big Data*, **5** (2018), 13. https://doi.org/10.1186/s40537-018-0122-y
21. L. Nigro, F. Cicirelli, P. Fränti, Parallel Random Swap: an efficient and reliable clustering algorithm in Java, *Simul. Model. Pract. Th.*, **124** (2023), 102712. https://doi.org/10.1016/j.simpat.2022.102712
22. S. Sieranoja, P. Fränti, Fast and general density peaks clustering, *Pattern Recogn. Lett.*, **128** (2019), 551–558. https://doi.org/10.1016/j.patrec.2019.10.019
23. L. Nigro, F. Cicirelli, ParDP: a parallel density peaks-based clustering algorithm, *Mathematics*, **13** (2025), 1285. https://doi.org/10.3390/math13081285
24. M. Du, S. Ding, H. Jia, Study on density peaks clustering based on k-nearest neighbors and principal component analysis, *Knowl.-Based Syst.*, **99** (2016), 135–145. https://doi.org/10.1016/j.knosys.2016.02.001
25. Q. Zhu, J. Feng, J. Huang, Natural neighbor: a self-adaptive neighborhood method without parameter K, *Pattern Recogn. Lett.*, **80** (2016), 30–36. https://doi.org/10.1016/j.patrec.2016.05.007
26. Q. Dai, Z. Xiong, J. Xie, X. Wang, Y. Zhang, J, Shang, A novel clustering algorithm based on the natural reverse nearest neighbor structure, *Inform. Syst.*, **84** (2019), 1–16. https://doi.org/10.1016/j.is.2019.04.001
27. T. Cover, P. Hart, Nearest neighbor pattern classification, *IEEE Trans. Inform. Theory*, **13** (1967), 21–27. https://doi.org/10.1109/TIT.1967.1053964
28. L. Nigro, F. Cicirelli, Fast clustering convergence by genetic algorithm, In: *Intelligent sustainable systems*, Singapore: Springer Nature, 2024, 331–342. https://doi.org/10.1007/978-981-97-9324-2_27
29. R. Urma, M. Fusco, A. Mycroft, *Modern Java in action*, New York: Manning Publications Co., 2019.
30. V. Subramanian, *Functional programming in Java*, Dallas: The Pragmatic Bookshelf, 2014.
31. L. Nigro, Performance of parallel K-Means algorithms in Java, *Algorithms*, **15** (2022), 117. https://doi.org/10.3390/a15040117
32. P. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *J. Comput. Appl. Math.*, **20** (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7
33. M. Rezaei, P. Fränti, Set matching measures for external cluster validity, *IEEE Trans. Know. Data Eng.*, **28** (2016), 2173–2186. https://doi.org/10.1109/TKDE.2016.2551240
34. P. Franti, C. Cariou, Q. Zhao, Cluster overlap as objective function, *CMC-Comput. Mater. Con.*, **85** (2025), 4687–4704. https://doi.org/10.32604/cmc.2025.066534
35. F. Wang, L. Li, P. Fränti, Avoiding local minima in overlap-based clustering by Random Swap, *Systems and Computing*, **2** (2026), 1–17. https://doi.org/10.64409/sycom.v2.i1.22
36. P. Fränti, S. Sieranoja, Clustering accuracy, *Appl. Comput. Intell.*, **4** (2024), 24–44. https://doi.org/10.3934/aci.2024003
37. P. Fränti, M. Rezaei, Q. Zhao, Centroid index: cluster level similarity measure, *Pattern Recogn.*, **47** (2014), 3034–3045. https://doi.org/10.1016/j.patcog.2014.03.017
38. P. Fränti, M. Rezaei, Generalized centroid index to different clustering models, In: *Structural, syntactic, and statistical pattern recognition*, Cham: Springer, 2016, 285–296. https://doi.org/10.1007/978-3-319-49055-7_26
39. L. Nigro, F. Cicirelli, Clustering performance of a recombinator Hartigan-Wong algorithm, *Computers*, **15** (2026), 394. https://doi.org/10.3390/computers15060394

40. *School of Computing, Clustering basic benchmark*, University of Eastern Finland, 2026. Available from: https://cs.uef.fi/sipu/datasets/.
41. *AT&T, Olivetti facial recognition dataset*, Scikit-learn developers, 2026. Available from: https://scikit-learn.org/stable/modules/generated/sklearn.datasets.fetch_olivetti_faces.html#sklearn.datasets.fetch_olivetti_faces.
42. F. Nie, Z. Li, R. Wang, X. Li, An effective and efficient algorithm for K-means clustering with new formulation, *IEEE Trans. Know. Data Eng.*, **35** (2023), 3433–3443. https://doi.org/10.1109/TKDE.2022.3155450
43. *UCI, UC Irvine machine learning repository*, UC Irvine, 2026. Available from: https://archive.ics.uci.edu/ml.



AIMS Press

©2026 the Author(s), licensee AIMS Press. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0)