computers

MDPI

Article

Clustering Performance of a Recombinator Hartigan–Wong Algorithm

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

[1] Department of Computer Engineering, Modeling, Electronic and System Engineering (DIMES), University of Calabria, 87036 Rende, Italy
[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

The work described in this paper continues basic research aimed at improving clustering algorithms such as K-Means and Random Swap through careful seeding and genetic concepts. This paper, in particular, develops a variation in the Hartigan–Wong (HW) algorithm, which, although computationally more expensive, is recognized as a better solution than K-Means. The new algorithm is named Recombinator Hartigan–Wong (Rec-HW). Rec-HW first builds a population of candidate solutions, each tailored to the minimization of the Sum-of-Squared-Errors (SSE) objective function cost. Candidate solutions are then systematically recombined by exploiting the standard behaviour of HW, which performs crossover and mutation operations. Recombinations, as experimentally confirmed, reduce the number of iterations required by basic HW and tend to favour the emergence of a solution close to the optimal one. The paper describes the design of Rec-HW, whose current implementation depends on parallel Java. Good clustering performance is demonstrated by using both benchmark and real-world datasets.

Keywords: unsupervised clustering; K-means; Hartigan–Wong; careful seeding; clustering indices; genetic concepts; benchmark datasets; realistic datasets; parallel Java

1. Introduction

This paper is concerned with unsupervised clustering through partitional algorithms [1]. The goal is to extract important information, patterns, and similarity relations from the sample data points of a dataset. The problem is of great interest in application domains including machine learning [2], software engineering, artificial intelligence, biomedicine, and so forth. A partitional algorithm [1,3] aims to split the data points into a certain number of clusters (groups), by guaranteeing that points in the same cluster are similar to one another, and points in different clusters are dissimilar. Each cluster is represented by its central point (centroid). Centroids play the fundamental role of guiding partitioning according to a Voronoi organization (e.g., [4,5]), which assigns points to clusters according to the nearest centroid. This same provision also ensures good clustering by pursuing the minimization of the *Sum-of-Squared-Errors* (SSE) objective function (see later in this paper).

A reference point for partitional algorithms is the K-Means heuristic [6–8], which is very often chosen for its simplicity and efficiency. K-Means has been the subject of many studies (e.g., [9–11]) that have pointed out its strong dependence on the initialization of centroids, and

Academic Editor: Paolo Bellavista

Received: 26 May 2026
Revised: 12 June 2026
Accepted: 17 June 2026
Published: 19 June 2026

its local refinement strategy for centroids, which can lead to getting stuck in a local suboptimal solution. Basically, K-Means repeatedly partitions the data points into K clusters (K is an input parameter), based on the currently defined centroids, followed by centroid updates within the resulting clusters. Iterations are continued until centroids stabilize or a maximum number of iterations has been executed. It has been shown in [11] that this technique of centroid refinement is unable to move surplus centroids out of selected clusters of the data space, to well-separated and far cluster areas where centroids are missing.

A significant example of an algorithm that replaces the local refinement strategy of K-Means with a global search, capable of examining the whole dataset for centroid improvement, is Random Swap [12,13]. Similar to K-Means, Random Swap starts with an initialization of centroids. After that, a series of swap iterations is executed. At each swap, a point is randomly selected in the dataset, and it goes to replace a randomly selected centroid. The new centroid configuration is refined by a few K-Means iterations (e.g., 2). If the new solution improves the SSE cost function with respect to the previous solution, it is accepted for further improvements/swaps. Otherwise, the previous solution and related partitioning are restored. Random Swap has proven to be capable of finding solutions close to the optimal one, provided it is iterated an adequate number of times [13].

Another algorithm that improves basic K-Means behaviour through a variation in the centroid refinement strategy based on iterated trials is Hartigan–Wong [14–17]. At each trial, every point of the dataset is extracted from its source cluster and tentatively moved to a destination cluster, with the two clusters that get updated accordingly. The point switches can follow the Voronoi method (see [18]) and be guided by the minimal distance to existing centroids, or, better, they can be Voronoi independent and be motivated by a net reduction in the SSE cost [16,17]. Whereas K-Means point partitioning and centroid updates are operations that can be carried out in parallel [19], Hartigan–Wong trials have to be executed sequentially, with point moves studied one-at-a-time. All of this introduces a computational cost that can be heavy in large and multidimensional datasets. However, the Hartigan–Wong algorithm proves to be more robust and less prone to ending in a local minimum solution than standard K-Means. Consequently, Hartigan–Wong clustering has more chances, in many practical cases, to detect a good solution, close to the optimal one.

Recently [20–23], it has been shown that a promising direction to enhance K-Means consists of using careful seeding methods for centroid initialization, together with genetic techniques [24,25]. Building a population of elitist solutions [25] by careful seeding, each solution independently devoted to the minimization of the SSE, can be a powerful basis for looking for a good clustering solution. Elitist solutions, in fact, are made up of candidate centroids which naturally tend, as experimentally confirmed, to thicken around optimal or ground-truth centroids. Although none of the population solutions can be near the optimal solution, a recombination of the candidate centroids is likely to give rise to a high-quality solution. K-Means (but also Random Swap) can be initialized by a solution selected from the population, which is then refined (crossed) by its partitioning and centroid update phases. In [25], density peaks [26–28] of candidate centroids are detected in the population, using a k-nearest neighbour (KNN) [27] approach, which are used to compose the initial solution that K-Means will refine. Good clustering results are reported in [22,23].

The original contribution of this paper is the development of an evolutionary version of Hartigan–Wong, named Recombinator Hartigan–Wong (Rec-HW). Rec-HW rests on a population of J elitist candidate solutions, established by using a careful seeding method. Subsequent generations of the population are created by initializing centroids from the population, and refining the centroids (crossover) through the trials of the basic Hartigan–Wong. In the case the resultant solution improves the initial solution from which it derives, the initial solution gets replaced (mutation) by the achieved solution, thus modifying the

population. As experimentally confirmed, the effects of trials are to change the population towards a final generation that favours the emergence of an accurate final solution. It is worth noting that the adoption of careful seeding methods both during the population setup and to feed recombinations tends to reduce the number of required trials and the convergence time to an accurate solution.

The paper is a significant extension of the preliminary Voronoi-based conference paper reported in [29]. The present paper differs from the conference version in the following aspects:

- Rec-HW design and implementation in parallel Java [30,31] are clarified.
- Rec-HW design consistently depends on a non-Voronoi organization: each point move, during a trial, actually occurs when it optimizes the objective function cost.
- Rec-HW no longer generates empty clusters, as can instead occur in the preliminary Voronoi version in [29].
- Rec-HW relies on Principal Component Analysis (PCA) [2] to handle multidimensional datasets. Details of PCA implementation are borrowed from [28].
- Rec-HW performs a batch of R independent runs (e.g., R = 50), from which confidence intervals are estimated for the various clustering quality indices.
- Rec-HW is thoroughly tested by applying it to many challenging datasets, both synthetic and realistic. Experimental results are also compared with those generated by competitor algorithms.

This paper is structured as follows. Section 2 briefly reviews some basic clustering concepts, the operation of K-Means, the role of the seeding methods, a Voronoi version of the Hartigan–Wong algorithm, and a synthesis of some clustering accuracy indices. Section 3 describes the proposed Recombinator Hartigan–Wong (Rec-HW) algorithm and its non-Voronoi organization. The section covers also some Java implementation aspects of Rec-HW. Section 4 reports a series of clustering experiments, using both synthetic and real-world challenging datasets. In particular, 16 realistic datasets are analyzed in detail for a punctual comparison with competitor algorithms. Finally, the conclusions are presented together with an indication of ongoing and future work.

2. Basic Concepts

A dataset X with N points is assumed: $X = \left\{ x_i|_{i=1}^{N} \right\}$, where $x_i \in R^D$ and all the D features (coordinates) are supposed to be numerical. Point similarity is expressed by the Euclidean distance, where $d(x_i, x_j)$ denotes the distance between points x_i and x_j:

$$d(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{iD} - x_{jD})^2} \quad (1)$$

The goal of a clustering algorithm is to partition X in K clusters $\left\{ C_j|_{j=1}^{K} \right\}$, $K \ll N$. Each cluster is a subset of points of X. Its centroid is denoted by μ_j. A Voronoi partitioning assigns each point x_i to the cluster that has the nearest centroid ($nc()$) to x_i: $C_j = C_j \cup \{x_i\}$, $\mu_j = nc(x_i)$. A populated cluster has a centroid which can be computed as the mean point of the cluster points:

$$\mu'_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i \quad (2)$$

The *Sum-of-Squared-Errors* (SSE) is the objective function a clustering algorithm has to minimize :

$$SSE = \sum_{j=1}^{K} \sum_{x_i \in Cj} d(x_i, \mu_j)^2 \quad (3)$$

Often the notion of *distortion*, here indicated as $\Phi = \frac{SSE}{N}$, that is, a normalized expression of the SSE, can be used as well.

A clustering solution is a pair consisting of a vector C of K centroids and a vector P of K disjoint partitions: $< C, P >$: $P_i \cap P_j = \varnothing, \cup P_j|_{j=1}^{K} = X$.

Merging two disjoint partitions P_a and P_b whose centroids are respectively μ_a and μ_b, and whose cardinalities are $n_a = |P_a|$ and $n_b = |P_b|$, would create a new partition with an increment in the distortion [21,24,25] which can be predicted to be:

$$\Delta\Phi_{ab} = \frac{n_a * n_b}{n_a + n_b} d(\mu_a, \mu_b)^2 \quad (4)$$

The centroid of the merged partition can be anticipated to be:

$$\mu_{ab} = \frac{n_a * \mu_a + n_b * \mu_b}{n_a + n_b} \quad (5)$$

In a similar way, extracting a sub-partition from a partition would cause a decrement in the distortion quantifiable with the same Formula 4. Centroids of the two resultant partitions could be computed as their mean points (Formula 2).

2.1. K-Means and Seeding Methods

Algorithm 1 reproduces the basic Voronoi behaviour of the K-Means algorithm [6–8]. Classical Lloyd's K-Means rests on random initialization of centroids: K points are chosen in the dataset according to a uniform random probability distribution.

Algorithm 1. Pseudo-code of the K-Means algorithm.

Input: Dataset X, number of clusters K, maximum number of iterations T
1. **initialization**: use a seeding method to define initialize centroids
2. **refinements**:
   iterations = 0
   do{
      partition points $x \in X$ to clusters according to the nearest centroid rule nc(.)
      recompute centroids of resultant clusters as mean points: $\mu'_j = \left(1/|C_j|\right)\sum_{x \in C_j} x$
      ++iterations
   }**while**(centroids differ from the previous ones and iterations < T)

Output: compute SSE and other accuracy clustering indices on the resultant solution.

Random seeding can assign multiple centroids to the same cluster area, with the consequence of splitting a real cluster into multiple smaller but unrealistic clusters. In addition, noise or outlier points could also be selected as centroids. Several improved initialization methods have been proposed [9,18]. In many cases, centroids are incrementally established in K rounds, with the first centroid, which is selected uniformly at random. If $D(x_i)$ represents the minimal distance of a point x_i from the currently defined centroids, *Maximin* [18] chooses the next centroid as a point x^* having maximum $D(x^*)$. This way, centroids are naturally selected far apart from one another A better method is *k-means++* [32], which defines the next centroid stochastically. First, each point x* of the dataset is associated with a probability π of being selected as follows: $\pi(x^*) = \frac{D(x^*)^2}{\sum_{i=1}^{N}(D(x_i))^2}$. Then, the next centroid is chosen through a random switch process. Although k-means++ prohibits parallelization of the K passes, it can enhance K-Means both in speed and accuracy by ensuring a "sparse" initialization of centroids. Despite an increment in the computational cost, a better initialization is achieved by embedding *k-means++* in the *greedy-k-means++* (*g-k-means++*)

method [20]. This new method depends on a constant S (e.g., $S = \lfloor 2 + logK \rfloor$ in [20]). At each new centroid definition, *k-means++* is executed S times, thus achieving S candidate centroids. Among the S proposals, the candidate that minimizes the SSE is chosen as the next centroid. Another example of a careful seeding method, here referred to as *refine*, is a modification of the refinement algorithm of Bradley & Fayyad [33], suggested by Baldassi in [21]. In the original proposal, the dataset is first split into S randomly filled segments. Then the S segments are individually clustered, e.g., using Lloyd's K-Means with a uniform random seed. After that, the S configurations of K centroids are separately used to seed K-Means. The best emerging solution, that is, the one that minimizes the SSE cost evaluated with respect to the S*K dataset of centroid points, is definitely adopted to seed K-Means. Refine proved to be effective in practical cases. However, high accuracy and reliability are ensured by the modification developed in [21]. After the initial split into S segments ($S = \sqrt{N/2K}$ in [21]), the S*K centroids/partitions are systematically reduced to K centroids/partitions by using the Pairwise Nearest Neighbours (PNN) method [24]. At each step, the two partitions are chosen whose merging would cause the minimal increment in the SSE/distortion. The distortion increment and the centroid of a potential merge of two (by construction disjoint) partitions can be anticipated by Equations (4) and (5).

All the above-mentioned seeding methods are implemented in Java. In particular, the PNN refinement technique is carried out in parallel and conveniently exploits the optimization discussed in [34], and experimented in [25]. Both *g-k-means++* and *refine* are at the core basis of the clustering algorithm proposed in this paper.

2.2. Voronoi Version of Hartigan–Wong

This classical variant of K-Means (see, e.g., [18]) is sketched in Algorithm 2. The refinement part of K-Means is replaced by trials. At each trial, every point, sequentially, is tentatively removed from its source cluster and possibly assigned to a different destination cluster. The movement is regulated by the nc(.) rule and causes the centroids of the source and destination clusters to be updated accordingly. Trials are repeated until no further movements occur or a maximum number of iterations has been executed.

Algorithm 2. Hartigan–Wong operation.

Input: Dataset X, number of clusters K, maximum number of iterations T
1. **initialization**:
   define initial centroids with a seeding method
   partition dataset points according to nc(.) rule and define initial clusters
2. **trials**:
   s = true
   iterations = 0
   do {
      s = true
      for (each point $x \in X$){
         remove x from its source cluster sc and update the centroid of sc
         assign x to other cluster dc according to nc(.) rule, and update centroid of dc
         if (dc ! = sc) s = false
      }
      ++iterations
   }**while** (!s and iterations < T)

Output: compute SSE and other accuracy clustering indices on the resultant solution

As experimentally confirmed [29], the algorithm favours careful clustering, and, unlike K-Means, trials avoid getting stuck in a local suboptimal solution in many practical cases. The latter property is better ensured [16,17] when the algorithm is reformulated with a non-Voronoi organization (see later in this paper), which also forbids the creation of empty clusters (see proof of property (2) of Theorem 2.1 in Ref. [16]).

2.3. Clustering Accuracy Indices

The quality of a clustering solution can be evaluated by several internal/external measures. Internal measures include the SSE/distortion and the Silhouette index (SI). The SSE, which costs O(N*K), furnishes an index of the internal compactness of clusters. More particularly, it is a sort of variance of the cluster point distributions. Lower values of the SSE indicate better clustering. The SI captures both the internal cohesion and the external separation of clusters. SI is the average of the Silhouette coefficients of the various points: $SI = \frac{1}{N}\sum_{x \in X} \frac{b_x - a_x}{\max(b_x, a_x)}$. The quantity a_x is the average distance of x to all the remaining points of the same cluster. The value b_x is the minimal average distance of x to the points belonging to other clusters. SI ranges within the interval $[-1, 1]$. SI values near 1 denote high separation of clusters (low or lacking overlapping). Values toward 0 indicate higher cluster overlapping. Finally, SI values close to -1 mirror erroneous clustering. SI costs $O(N^2)$, because it is necessary to compute the distances between every pair of points. Consequently, SI can be difficult to evaluate in large and multidimensional datasets.

Some external measures [35] considered in this paper include the Centroid Index (CI) [36] (also covering its generalization proposed in [37]), the accuracy index ACC [38], the Normalized Mutual Information (NMI) index, the F-Score, and the Adjusted Rand Index (ARI). The following briefly recalls some information about these indices. More details can be found in [23,28].

The Centroid Index (CI) is a useful measure for quantifying the similarity degree between a detected (prototype) clustering solution and a ground-truth solution, e.g., provided by the designer of a synthetic dataset or by the experts of a domain-specific real-world dataset. Often, a benchmark or realistic dataset is accompanied by ground-truth centroids (GTC) or ground-truth partitions (labels) (GTP). CI computes its value in the two-way mapping between the prototype centroids (C) detected by an algorithm and the available GTC: C $\longrightarrow$ GTC, GTC $\longrightarrow$ C. CI measures the maximum number of "orphans" in the two mapping directions. Any centroid in C maps to the nearest centroid in GTC. Resultant "orphans" in GTC are those ground-truth centroids upon which no centroid in C is mapped. This number of orphans denotes the number of real clusters upon which the proposed algorithm was unable to assign a centroid. Similarly, the number of orphans in C, following the GTC $\longrightarrow$ C mapping, represents the number of clusters where multiple centroids were assigned. The CI generalizes to GCI when GTP are involved. In this case, sets of labels (partitions) can be mapped to one another by using, e.g., the Jaccard distance [13]. For simplicity, in the rest of the paper, the term CI will uniformly be adopted, which implicitly refers to GCI when ground-truth partitions are involved. CI ranges in [0, K $-$ 1]. CI = 0 denotes a "structurally" correct clustering, in which, e.g., prototype centroids are very close (although not coincident) to GTC. Values CI > 0 reflect the number of incorrect prototype centroids.

The ARI pair-matching measure can be computed from the contingency matrix (CM) [35], when partitions (sets of labels) are involved. CM rows are associated with prototype partitions P^i, columns to GTP^j. Each element of CM is valued to: $n_{ij} = \left|P^i \cap GTP^j\right|$. The sums of row elements, and of column elements, are re-

spectively: $n_i|_{i=1}^{K} = \sum_{j=1}^{K} n_{ij} = |P^i|$, $m_j|_{j=1}^{K} = \sum_{i=1}^{K} n_{ij} = |GTP^j|$. The ARI can be computed as:

$$ARI = \frac{\sum_{i,j} \binom{n_{ij}}{2} - \left[\sum_i \binom{n_i}{2} \sum_j \binom{m_j}{2}\right] / \binom{N}{2}}{\frac{1}{2}\left[\sum_i \binom{n_i}{2} + \sum_j \binom{m_j}{2}\right] - \left[\sum_i \binom{n_i}{2} \sum_j \binom{m_j}{2}\right] / \binom{N}{2}} \quad (6)$$

ARI values close to 1 characterize accurate clustering. In particular, ARI = 1 indicates that the obtained prototype partitions P fully agree with the ground-truth partitions. An ARI close to 0 indicates that the agreement would be no different from that of random clustering. Also, the NMI [28], F-score [23], and ACC [28,38] measures can be inferred from the contingency matrix. Values close to 1 indicate good clustering.

2.4. Genetic Concepts for Clustering

Genetic and evolutionary concepts have been successfully experimented with in machine learning and unsupervised clustering. A significant example is the genetic algorithm (GA) developed in [24], where the initial population consists of individuals that are candidate solutions, preliminarily generated by applying a seeding method to the available dataset. Subsequent generations of the population are then produced through the genetic operations of selection, crossover, and (possibly) mutation, guided by the fitness function defined by the clustering objective cost (e.g., minimizing SSE). More particularly, GA rests on elitist solutions, which individually address a reduction in the SSE. GA goes on by detecting (selection), at each step, the best two available elitist solutions, which are then merged by the Pairwise Nearest Neighbours (PNN) technique, acting as a crossover operation. The new solution is refined by K-Means, possibly modified by Random Swap [12] operations, and added to the population by replacing the two originating solutions. After a certain number of generations, the best resultant solution in the population is returned, which proves to be highly accurate. The high computational cost of GA in [24] is smoothed in [25], where a parallel implementation of the algorithm in Java is presented. A notable aspect of [25] is an efficient realization of the PNN operations, guided by the observations reported in [34]. Genetic concepts are also exploited in [23] and in this paper, where the population's initial elitist candidate solutions/centroids are established by using a careful seeding method. Particularly, next generations are created, in [23], by estimating the density of candidate centroids by a k-nearest neighbours approach. Each generation corresponds to a specific k value, which defines the number of distinct nearest centroids to a given candidate centroid, and permits the identification of its neighbourhood. The first K high-density centroids are used to compose the target solution, which is then refined by K-Means. Details about the genetic aspects adopted in this work are clarified in the next section.

3. Recombinator Hartigan–Wong Algorithm

This new proposed algorithm, referred to as Rec-HW, is characterized by its non-Voronoi setting and by its adoption of a genetic approach [24]. As motivated in [16,17], movements of points during trials are no longer constrained by the nearest centroid rule but by a decrement of the distortion function cost. In addition, as in [22–24], a population of candidate solutions, each targeted to a reduction in the SSE/distortion, is preliminarily established. Although no individual solution [20] of the population has, in general, a high chance to be already close to the optimal one, the candidate centroids tend to cluster around the ground-truth centroids [23]. As a consequence, a centroid in a dense area has a high likelihood of being chosen by an initialization method. Moreover, since a careful seeding

method avoids selecting near centroids, the other components of a dense area of a chosen centroid have a lower probability to be selected too.

Rec-HW is driven (selection operation) by a solution extracted from the population, which gets improved by subsequent trials (crossover operation). A refined solution can replace the initial one in the population (mutation operation), thus giving rise to a new generation. Rec-HW can be repeated a certain number of times, with the population that consolidates toward a configuration that favours the definition of an accurate solution.

3.1. Non-Voronoi Behaviour

Let x be a dataset point belonging to a source cluster sc, and consider its possible relocation to a destination cluster dc. As in Algorithm 2, x is first removed from sc. The centroid of sc is recomputed accordingly. Now, all the K clusters in the current partitioning, together with the singleton cluster {x}, are mutually disjoint. Therefore, merging {x} with the cluster dc would cause an increment of the distortion (see Equation (4)) of:

$$\Delta\Phi_{x,dc} = \frac{n_{dc}}{1+n_{dc}} d(x,\mu_{dc})^2 \tag{7}$$

with the updated centroid of dc being:

$$\mu'_{dc} = \frac{x + n_{dc} * \mu_{dc}}{1+n_{dc}} \tag{8}$$

Dually, the extraction of x from sc determines a decrement of the distortion as follows:

$$\Delta\Phi_{sc,x} = \frac{(n_{sc}-1)}{(n_{sc}-1)+1} d\left(x,\mu'_{sc}\right)^2 = \frac{(n_{sc}-1)}{n_{sc}} d\left(x,\mu'_{sc}\right)^2 \tag{9}$$

where n_{sc} is the cardinality of sc before the removal of x, and μ'_{sc} is the centroid of the updated source cluster sc. Overall, switching x from sc to dc would result in a distortion variation:

$$\Phi - \Delta\Phi_{sc,x} + \Delta\Phi_{x,dc} \tag{10}$$

It is clear that moving x is convenient if $\Delta\Phi_{sc,x} > \Delta\Phi_{x,dc}$, because it would determine a reduction in the whole distortion cost. Actually, all the existing clusters have to be checked, with dc being identified as the cluster that would imply the maximal reduction in the distortion.

3.2. Operation of Rec-HW

An abstract description of Rec-HW operation is reported in Algorithm 3. A solution is represented by a pair <C,P>, where C is a vector of K centroids, and P is a vector of K partitions (set of labels, that is, indices of dataset points). Both the creation of the population and the recombination phase exploit careful seeding, e.g., *refine* for building candidate solutions, *g-k-means++* for extracting a solution from the population to refine by trials. Rec-HW, though, acts as a meta-method, because other choices are possible as well. As one can see from Algorithm 3, Rec-HW trials are used both to refine solutions (make them elitist [23]) to add to the population, and to improve working solutions during recombinations. It is worth noting that due to the use of careful seeding, a small number of trials are required for refinements. Consequently, the maximum number of trials T used in Algorithm 2 now becomes useless.

Algorithm 3. Abstract operation of Recombinator Hartigan–Wong algorithm.

Input: Dataset X, number of clusters K, number of solutions J of population, number of repetitions/recombinations R

1. **Initialization**: create population $\wp$ with J solutions (J*K centroids)
   $\wp \leftarrow \varnothing$
   repeat J **times** {
   C ← seeding(X, K, *refine*)
   P ← partition(X, C, K)
   <C′,P′> ← refine-by-trials(<C,P>)
   $\wp \leftarrow \wp \cup$ {C′}
   }
2. **Recombination**: create R generations of $\wp$
   best-cost ← ∞, best ← ?
   repeat R **times**{
      C ← seeding($\wp$, K, *g-k-means++*)
      P ← partition(X, C, K)
      <C′,P′> ← refine-by-trials(<C,P>)
      cost ← SSE(<C′,P′>)
      if (cost < best-cost){
         best ← <C′,P′>
         best-cost ← cost
      }
   }

Output: SSE and other clustering quality indices, including statistical data, of the emerged best solution.

3.3. Implementation Issues

Rec-HW is implemented in functional Java [30,31], using parallel streams and lambda expressions [13,19,22,23,39]. A critical point in Rec-HW is the non-parallelizable actions during a trial. Dataset points are to be examined one-at-a-time, sequentially, with the effects of a move influencing subsequent moves and so forth. Several implementation choices were adopted to accelerate recurrent operations. In particular, to efficiently support the cluster updates performed during point relocations, a global array centre[K] was introduced. At any time, centre[k] stores the sum of the data points currently assigned to cluster k, together with its cardinality, which is automatically updated during point additions and removals. Whenever a point p is added to or removed from a cluster c, its coordinates are correspondingly added to or subtracted from centre[c]. From the information in centre[c], it is immediately possible to update the centroid of c. At each new trial, the array centre[.] is reset, and its content is renewed (in parallel) according to the most recent partitioning operation (also carried out in parallel). Algorithm 4 illustrates the partitioning corresponding to a given centroid vector, and the initial setting of centre[.], just before starting a new trial. Each DataPoint p maintains, besides the coordinates, the identity of the belonging cluster (CID-Cluster ID). After the initial seeding, the global vector centroids[K] contain the initialized centroid points.

Operations 1 and 2 in Algorithm 4 can exploit Java parallel streams, which in turn depend on the underlying fork/join mechanism capable of spawning multiple threads to work in parallel on separated segments of the data. During partitioning, dataset points are processed in parallel. To avoid data inconsistencies, each point only modifies itself. Point modifications are carried out as part of the map's functional operation. The various maps are actually triggered by the forEach terminal operation. Of course, the actual gain in

computing speed depends on the size of the data. For small datasets, it can be convenient to disable the parallelism (see the global PARALLEL field). Similar considerations can be repeated for the parallel stream that initializes the array centre[].

Algorithm 4. Java parallel version of data partitioning and centre[] initialization.

```
...
//0. initialize array centre[]
for (int k = 0; k < K; ++k) {centre[k].reset(); centre[k].setCID(k);}
//1. partition dataset points according to current defined centroids[]
Stream<DataPoint> p_stream = Stream.of(dataset);
if (PARALLEL) p_stream = p_stream.parallel();
p_stream
    .map( p -> {
        double md = Double.MAX_VALUE;
        for( int k = 0; k < K; ++k ){
            double d = p.distance(centroids[k]);
            if (d < md) {md = d; p.setCID(k);}
        }
        return p;
    })
    .forEach (p->{});
//2. put in centre[c] the sum of points and cardinality of cluster c
Stream<DataPoint> c_stream = Stream.of(centre);
if (PARALLEL) c_stream = c_stream.parallel();
c_stream
    .map( c -> {
        for( int i = 0; i < N; ++i ){
            if (dataset[i].getCID()==c.getCID()) c.add(dataset[i]);
        }
        return c;
    })
    .forEach( c->{} );
...
```

The sequential operations carried out in a trial can be inspected in Algorithm 5, where one can easily retrieve the non-Voronoi design underlying Rec-HW.

As in [23,28], Rec-HW can apply a preliminary scaling to the dataset points, e.g., by dividing each coordinate by the overall maximum, applying min-max normalization, and so forth. In addition, the Principal Component Analysis (PCA) [2,28] technique is used to reduce the number of coordinates in multidimensional datasets to the most relevant ones. Technical details about the calculation of the eigenvalues of the covariance matrix are reported in [28].

The Rec-HW algorithm is repeated R times to detect the best emerging solution. Moreover, the R values (e.g., R = 50) of clustering indices such as ARI, NMI, ACC, and so forth are accumulated across the various runs, and the confidence interval (confidence degree 95%) can finally be estimated for each measure.

Algorithm 5. Trials operations of Rec-HW.

```
. . .
//3. make trials
it = 0;
boolean s = true;
do{
    s = true; ++it;
    //for each data point xi, that is, dataset[i]
    for( int i = 0; i < N; ++i ){
        //remove xi from its source cluster sc
        sc = dataset[i].getCID(); centre[sc].sub(dataset[i]);
        double nsc = centre[sc].getN(); //new cardinality of sc
        DataPoint musc = new DataPoint(centre[sc]); musc.mean(); //new centroid of sc
        //compute distortion decrement Ddec
        double d = dataset[i].distance(musc);
        double Ddec = (nsc/(nsc + 1)) * d * d;
        //detect cluster dc which would imply the minimum distortion increment Dinc
        dc = 0;
        double Dinc = Double.MAX_VALUE;
        for( int c = 0; c < K; ++c ){
            centre[c].add(dataset[i]);
            double ndc = centre[c].getN();
            DataPoint mudc = new DataPoint(centre[c]); mudc.mean();
            //compute minimal Dinc
            d = dataset[i].distance(mudc);
            double Ddc = (ndc/(ndc + 1)) * d * d;
            centre[c].sub(dataset[i]);
            if (Ddc < Dinc) {Dinc = Ddc; dc = c;}
        }
        //check if switch has to be carried out
        if( Ddec > Dinc && dc != sc ){
            //move xi to dc
            centre[dc].add(dataset[i]); dataset[i].setCID(dc);
            centroids[sc] = new DataPoint(centre[sc]); centroids[sc].mean();
            centroids[dc] = new DataPoint(centre[dc]); centroids[dc].mean();
            s = false;
        }
        else{
            //no move
            centre[sc].add(dataset[i]); dataset[i].setCID(sc);
            centroids[sc] = new DataPoint(centre[sc]); centroids[sc].mean();
        }
    }//for
}while( !s );
. . .
```

4. Series of Clustering Experiments

The effectiveness of Rec-HW was thoroughly checked by clustering both synthetic and real-world datasets. All the experiments were carried out 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. Clustering the A3 Dataset [40]

A first indication of the accuracy and reliability of Rec-HW clustering was achieved by applying it to the A3 2-dimensional synthetic dataset [40], which has circular clusters and ground-truth centroids. Dimensions of A3 are: N = 7500, D = 2, K = 50. For simplicity, A3 was preliminarily scaled by dividing all the point coordinates by the overall maximum. PCA confirmed that the two coordinates of points are principal components. The dataset was separately clustered by Repeated Lloyd's K-Means (R-LKM) (uniform *random* seeding) [7], the basic Hartigan–Wong algorithm [15] with, respectively, a Voronoi (R-V-HW) and a non-Voronoi behaviour based on minimum distortion increment (R-nV-HW), with *g-k-means++* seeding, and, finally, by Rec-HW by preparing a population of J = 10 solutions with the *refine* initialization method, and by making recombinations with the *g-k-means++* method. A batch of R = 30 runs was executed for each algorithm. The results are collected in Table 1. The SSE column refers to the solution with the minimum SSE. CI and SI are the values corresponding to the minimum SSE. The success rate (SR %) denotes the number of observed runs that ended with CI = 0. For Rec-HW, the Elapsed Time includes both the time for the population setup, and the time for the recombinations.

Table 1. Clustering results for the A3 dataset [40].

Algorithm	SSE	CI	SI	SR (%)	avIT	ET (s)
R-LKM	9.37	4	0.54	0	27.1	0.67
R-V-HW	6.74	0	0.60	7	8.93	3.04
R-nV-HW	6.74	0	0.60	10	7.67	5.25
Rec-HW	6.74	0	0.60	100	2.5	4.18

Table 1 confirms the positive effects of careful seeding on the clustering results. No run of R-LKM (with uniform random seeding) finished with CI = 0 (SR = 0%), and the minimum observed CI was 4 (four clusters/centroids were incorrectly estimated). All the Hartigan–Wong-based algorithms were capable, although with some interesting differences, of generating a good solution with non-zero SR.

As one can see from Table 1, the use of a non-Voronoi organization reduces the number of trials required for the refinement of the starting solution, and increases the success rate. In some executions of the R-V-HW, the creation of an empty cluster was observed, which forced the repetition of the same run. The best performance was achieved by Rec-HW, which correctly solved the dataset with a success rate of 100%, meaning that each run, although started with a different centroid configuration, with a few trials (2.5 on average), always generates a solution with the same minimal SSE and CI = 0. The SR = 100% also indicates that even one single run of the Rec-HW is sufficient, for this dataset, to generate an accurate clustering.

The very good clustering can also be seen on Figure 1, which shows the partitions and the centroids proposed by Rec-HW. The obtained centroids are depicted by black coloured points. Ground-truth centroids are instead red coloured. In all the partitions, prototyped centroids and ground-truth centroids almost coincide.

4.2. Dataset Clustering Sensitive to SSE Minimization

The usual hypothesis for many clustering algorithms, including Rec_HW, is that good clustering follows from SSE minimization. Unfortunately, this is not always true (see also

the next sub-section). Nevertheless, six synthetic datasets that obey the basic hypothesis are considered below. The dimensional attributes of the datasets, available from [40] and equipped with ground-truth centroids, are reported in Table 2. The datasets have clusters with different point distributions (e.g., Gaussian, as in S4), overlapping degrees, sizes, and positions in the data space. The Birch1 and Birch2 datasets are composed of 100 spherical clusters regularly distributed, respectively, on a 10 × 10 grid and over a sinusoid. They can be more difficult to handle due to their size.

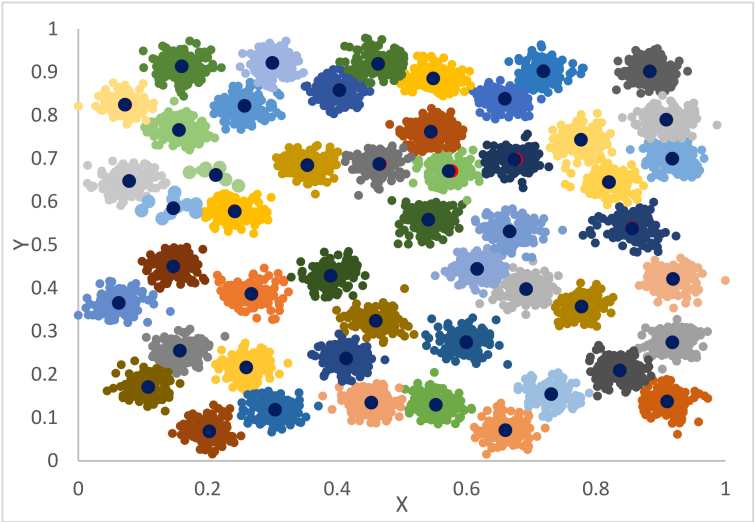

Figure 1. Rec-HW generated partitions and centroids for the A3 dataset [40].

Table 2. Second group of datasets [40] studied by Rec-HW.

Dataset	N	D	K
Asymmetric	1000	2	5
Overlap	1000	2	6
S4	5000	2	15
Unbalance2	6500	2	8
Birch1/2	100,000	2	100

All the datasets in Table 2 were studied by Rec-HW after a preliminary scaling by the overall maximum of the point coordinates. A population of J = 10 solutions was achieved using the *refine* seeding, and a batch of 30 repetitions fed by *g-k-means++* was used. Results are collected in Table 3. As one can see, all the datasets were correctly clustered (CI = 0) with a success rate of 100% in all the cases.

Table 3. Clustering results of the datasets in Table 2.

Dataset	SSE	CI	SI	SR (%)	avIT	ET (s)
Asymmetric	0.98	0	0.64	100	1	0.23
Overlap	1.30	0	0.45	100	1	0.34
S4	16.44	0	0.48	100	5.72	2.41
Unbalance2	1.09	0	0.85	100	1.97	0.81
Birch1	92.77	0	0.46	100	6.5	225.58
Birch2	0.23	0	0.78	100	2.1	110.13

The quality of the achieved clustering can also be checked in Figures 2–7, which depict the results proposed by Rec-HW. As in Figure 1, ground-truth centroids are shown in red, and prototype centroids are shown in black. Detected centroids coincide with or are very close to the ground-truth centroids. The Unbalance2 dataset (Figure 5) is characterized by three high-density clusters (in yellow, green, and blue) immersed within five low-density clusters. Results for Birch1 and Birch2 coincide with or are better than those reported in [22,23].

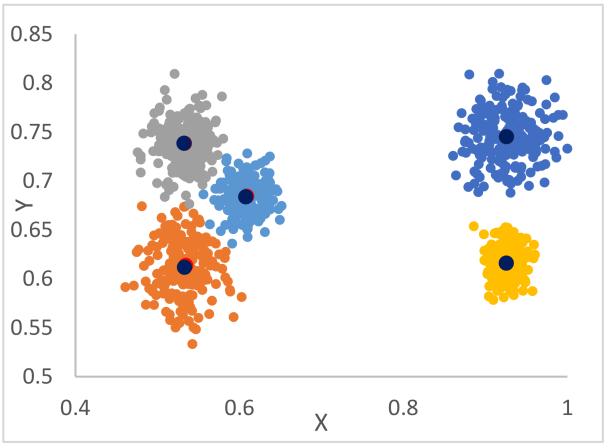

Figure 2. Detected partitions and centroids for the Asymmetric dataset.

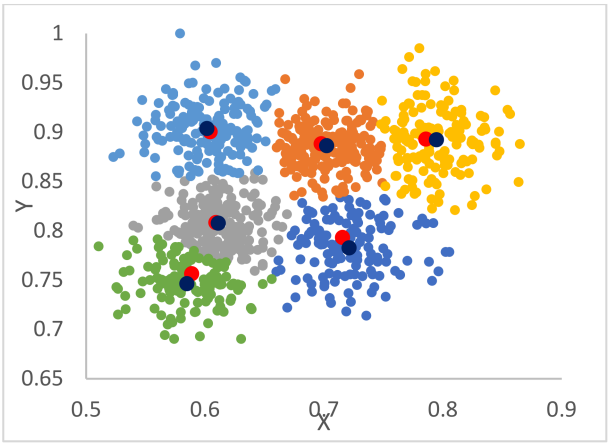

Figure 3. Partitions and centroids for the Overlap dataset.

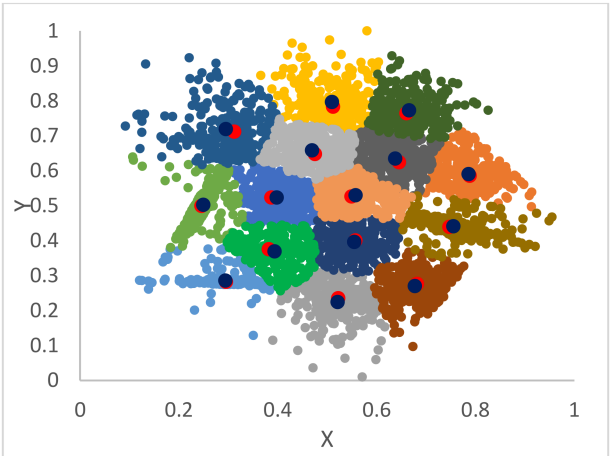

Figure 4. Partitions and centroids for the S4 dataset.

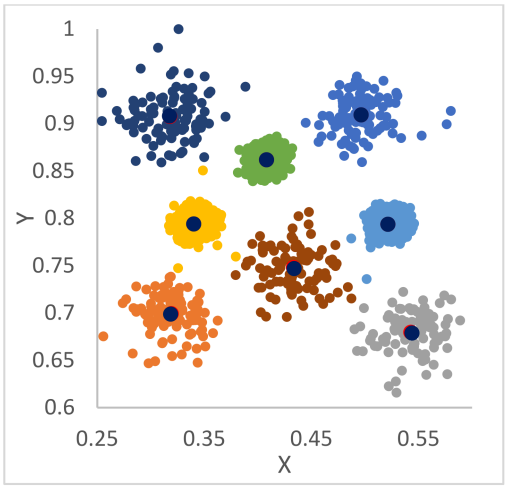

Figure 5. Partitions and centroids for the Unbalance2 dataset.

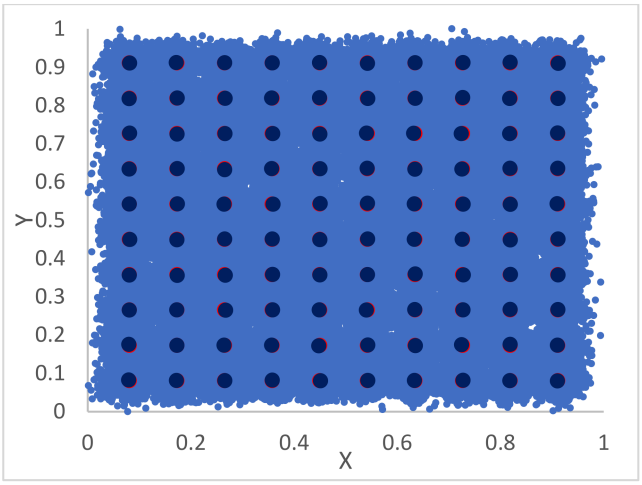

Figure 6. Prototype centroids for the Birch1 dataset.

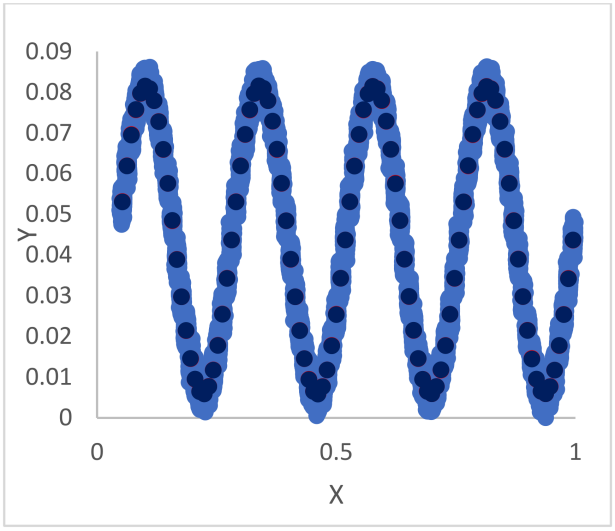

Figure 7. Prototype centroids for the Birch2 dataset.

It is worth noting that the PCA analysis confirmed, in almost all the datasets, a couple of point coordinates, except for Birch2, where only one feature (the first one) was determined to be the principal component.

4.3. Some Challenging Datasets

Table 4 describes some datasets that are challenging to cluster. Birch3 is a synthetic dataset with the same dimensions as Birch1 and Birch2 (see Table 2). However, the 100 clusters of Birch3 (see Figure 8) are irregularly defined, have different sizes, and are randomly placed in the data space. To the best of our knowledge, no clustering solution with a value of the Centroid Index CI [36] lower than 11 seems to have been reported in the literature. The source of difficulty, as shown, e.g., in [13], is that Birch3 clustering is not sensitive to SSE minimization. It was observed that, sometimes, a reduction in the CI value corresponds to an increase in the SSE. Another hard-to-cluster dataset is Worms_2D [40] (see Figure 9). It contains 35 clusters whose worm-like artificial shape is established by starting from a random position and moving, step-by-step, according to a random direction. At each step, points follow a Gaussian distribution that generates a cloud around the current position. Clustering algorithms not based on the concepts of SSE minimization but on the detection and the exploitation of density peaks in the dataset [26–28] have reported a CI (here Generalized CI because Worms_2D is provided with ground-truth partitions) of about 7.5–8 and, in one case [28], a value of 3 was observed.

Table 4. Datasets which are difficult to cluster.

Dataset	N	D	K
Birch3	100,000	2	100
Worms_2D	105,600	2	35
Olivetti	400	4096	40

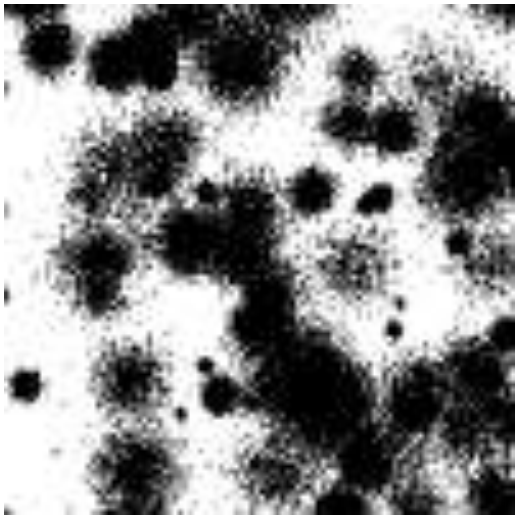

Figure 8. Birch3 dataset from [40].

The realistic Olivetti dataset (see, e.g., [20]) represents a facial recognition problem, where 40 human subjects, each photographed in 10 different poses, have to be identified from the various images. Each facial photo is represented by $64 \times 64 = 4096$ pixels. In [22], a CI = 7 was achieved.

The three datasets were clustered by first creating, for each dataset, a population of J = 20 solutions, and by making R = 100 repetitions. Birch3, which comes with ground-truth centroids, and Worms_2D, which is provided with ground-truth partitions, were first scaled by the overall maximum of the coordinates. Olivetti (provided by ground-truth centroids and ground-truth partitions) was processed without scaling.

Figure 9. Worms_2D dataset from [40].

As one can see from Table 5, clustering results generated by Rec-HW are in line with or slightly better than those reported in the literature. Figure 10 depicts the observed SSE vs. the number of runs. For simplicity, only the decreases in the SSE are registered. In the runs where the SSE increases, the previous minimal value is replicated.

Table 5. Clustering results of the datasets in Table 4.

Dataset	SSE	CI	SI	SR (%)	avIT	ET (s)
Birch3	37.74	11	0.52	0	28.07	1134.7
Worms_2D	79.32	7	0.36	0	25.45	475.6
Olivetti	11,480.75	6	0.16	0	4.56	422.2

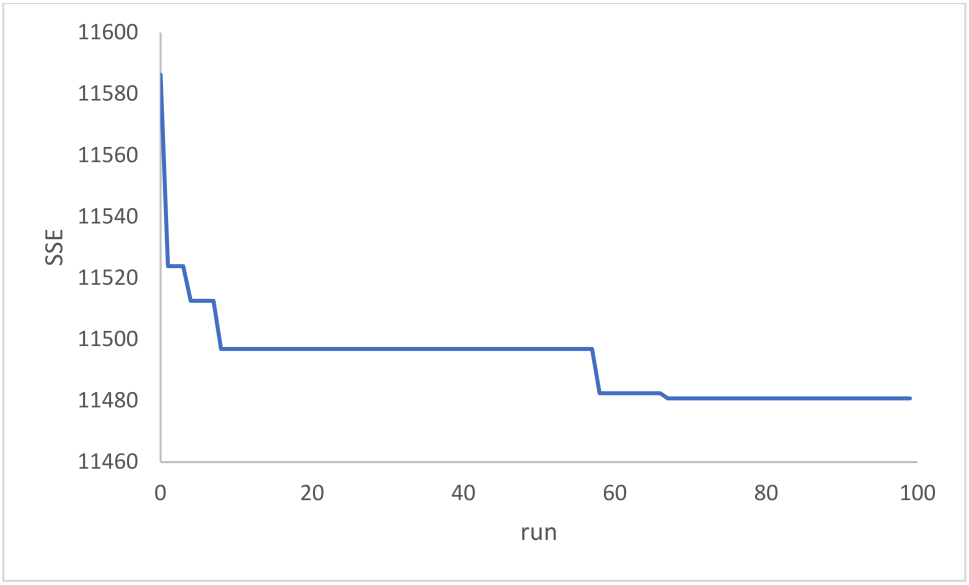

Figure 10. SSE vs. run for the Olivetti dataset.

4.4. Realistic Datasets

In [41], a new formulation of the K-Means objective function as a trace maximization problem was proposed. The new variant of K-Means, which here will be referred to as Nie-K-Means, does not need to recalculate centroids at each iteration and requires fewer

additional intermediate variables during the optimization process. Nie-K-Means emerged as a more efficient and accurate version of the K-Means algorithm.

Below, the 16 real-world multidimensional datasets that were used in [41] to test the features of the Nie-K-Means algorithm are also selected as a further test for Rec-HW. Some of these datasets are available from the UCI public repository [42]. The remaining datasets were obtained directly from the Nie-K-Means research work. The datasets were also experimented with in a recent paper [23] to evaluate the K-Means clustering driven by density peaks of candidate centroids.

Table 6 specifies the datasets of [41], listed by increasing number of dimensions (coordinates). All the datasets were clustered by Rec-HW with no scaling, J = 20 solutions (built with the *refine* seeding) in the population, and by performing R = 50 repetitions. For comparison purposes with [41], Rec-HW recombinations were repeated using, separately, the uniform *random* seeding and the *g-k-means++* seeding.

Table 6. Real-world datasets of Nie-K-Means work [41].

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	36,771	10

Table 7 reports the basic clustering results, that is the minimal observed SSE, and its corresponding CI and SI values, the success rate (SR%), the average of iterations during trials, the overall Elapsed Time ET (s) required by Rec-HW, and the number (PCA) of the principal components detected by Principal Component Analysis.

Table 7. Rec-HW basic results, with *random* seeding, for the datasets of Table 6.

Dataset	SSE	CI	SI	SR (%)	avIT	ET (s)	PCA
Iris	78.95	0	0.55	80	4.20	0.55	4
Balance	3472.32	1	0.17	0	7.48	0.98	4
Dermatology	5580.60	1	0.19	12	4.96	1.18	34
Uspst	63,342.50	1	0.15	0	19.12	22.09	226
USPSdata_20	66,241.37	1	0.16	0	17.30	18.97	226

Table 7. *Cont.*

Dataset	SSE	CI	SI	SR (%)	avIT	ET (s)	PCA
USPSdata	33,3917.29	1	0.17	0	29.24	151.92	226
MSRA25	151,542,870.88	2	0.19	0	11.68	18.54	240
PalmData25	502,924,881.30	15	0.29	0	8.68	138.83	252
Binalpha	67,101.17	7	0.06	0	14.98	54.5	315
Ecoli	338.90	2	0.01	0	7.74	3.99	339
Corel_5k	4,367,794.42	18	0.09	0	34.78	416.31	183
MnistData_05	8,750,188,511.82	1	0.07	0	24.56	107	434
MnistData_10	17,594,185,757.59	1	0.06	0	32.90	273	434
Coil20Data_25	2,390,598,975.34	4	0.23	0	13.54	89.9	893
Mpeg7	5508.91	12	0.11	0	11.58	2617.48	5248
TDT2_10	195,506.52	2	0.19	0	7.96	815.4	11,028

As one can see from Table 8, the use of careful seeding guarantees, in general, better values for the SSE and SI. In addition, the average number of iterations per trial diminishes when *g-k-means++* replaces *random* seeding. Except for the Iris dataset, a CI > 0 was always registered, meaning that the achieved clustering is approximate. Moreover, the case of Dermatology with an SR = 90%, and a CI = 1 confirms that the best clustering (CI = 0) does not always coincide with the attainment of minimal SSE.

Table 8. Rec-HW basic results, with *g-k-means++* seeding, for the datasets of Table 6.

Dataset	SSE	CI	SI	SR (%)	avIT	ET (s)
Iris	78.95	0	0.55	100	1.38	0.59
Balance	3472.32	1	0.17	0	5.34	1.57
Dermatology	5585.02	1	0.22	90	2.12	1.21
Uspst	63,342.41	1	0.16	0	4.06	13.6
USPSdata_20	66,241.37	1	0.16	0	3.86	12.17
USPSdata	333,917.36	1	0.16	0	7.74	79.42
MSRA25	151,542,870.88	2	0.19	0	5.3	15.24
PalmData25	478,967,435.73	8	0.32	0	4.4	239.68
Binalpha	66,886.37	6	0.07	0	9.46	53.62
Ecoli	338.90	2	0.01	0	6.14	4.04
Corel_5k	4,355,292.35	18	0.09	0	14.14	257.49
MnistData_05	875,002,3791.49	1	0.06	0	11.5	73.37
MnistData_10	17,594,181,633.10	1	0.06	0	13.22	172.19
Coil20Data_25	2,360,483,642.93	3	0.21	0	6.24	72.34
Mpeg7	5474.46	12	0.11	0	7.74	4124.4
TDT2_10	195,329.82	2	0.19	0	4.92	767.8

In order to better understand the quality of the resulting clustering results, Tables 9 and 10 report the estimated values for the ACC, NMI, F-Score, and ARI indices,

respectively for the case where uniform *random* and *g-k-means++* seeding is adopted. These values will be directly comparable with similar results documented in [41].

Table 9. Rec-HW results about ACC, NMI, F-Score, and ARI, with *random* seeding, for the datasets of Table 6.

Dataset	ACC	NMI	F-Score	ARI
Iris	0.8427 ± 0.0246	0.7106 ± 0.0175	0.7786 ± 0.0182	0.6582 ± 0.0326
Balance	**0.5266 ± 0.0078**	**0.1218 ± 0.0112**	**0.4647 ± 0.0066**	**0.1410 ± 0.0105**
Dermatology	0.8324 ± 0.0197	0.8404 ± 0.0179	0.7659 ± 0.0313	0.7019 ± 0.0412
Uspst	**0.7002 ± 0.0043**	**0.6123 ± 0.0036**	**0.5751 ± 0.0053**	**0.5240 ± 0.0060**
USPSdata_20	0.6934 ± 0.0065	0.6193 ± 0.0048	0.5700 ± 0.0073	0.5183 ± 0.0083
USPSdata	**0.7106 ± 0.0020**	**0.6131 ± 0.0012**	**0.5824 ± 0.0023**	**0.5320 ± 0.0028**
MSRA25	0.5334 ± 0.0112	0.5855 ± 0.0106	0.3961 ± 0.0139	0.3304 ± 0.0170
PalmData25	**0.7864 ± 0.0036**	**0.9235 ± 0.0014**	**0.7129 ± 0.0050**	**0.7099 ± 0.0050**
Binalpha	**0.4636 ± 0.0045**	**0.5883 ± 0.0026**	**0.3085 ± 0.0035**	**0.2886 ± 0.0037**
Ecoli	0.5165 ± 0.0065	0.5626 ± 0.0048	0.4668 ± 0.0066	0.3545 ± 0.0076
Corel_5k	**0.1896 ± 0.0011**	**0.2712 ± 0.0008**	**0.0836 ± 0.0004**	**0.0626 ± 0.0005**
MnistData_05	0.5566 ± 0.0080	0.4826 ± 0.0056	0.4163 ± 0.0073	0.3491 ± 0.0080
MnistData_10	0.5736 ± 0.0062	0.4932 ± 0.0042	0.4289 ± 0.0050	0.3625 ± 0.0055
Coil20Data_25	**0.6619 ± 0.0107**	**0.7727 ± 0.0045**	**0.5903 ± 0.0091**	**0.5664 ± 0.0097**
Mpeg7	**0.5883 ± 0.0036**	**0.7547 ± 0.0017**	**0.4398 ± 0.0037**	**0.4316 ± 0.0038**
TDT2_10	**0.4736 ± 0.0062**	**0.4763 ± 0.0066**	**0.2716 ± 0.0030**	**0.1374 ± 0.0038**

Table 10. Rec-HW results about ACC, NMI, F-Score, and ARI, with *g-k-means++* seeding, for the datasets of Table 6.

Dataset	ACC	NMI	F-Score	ARI
Iris	**0.8867 ± 0.0000**	**0.7419 ± 0.0000**	**0.8111 ± 0.0000**	**0.7163 ± 0.0000**
Balance	**0.5290 ± 0.0061**	**0.1244 ± 0.0076**	**0.4673 ± 0.0051**	**0.1448 ± 0.0081**
Dermatology	**0.9340 ± 0.0072**	**0.8938 ± 0.0026**	**0.8979 ± 0.0126**	**0.8728 ± 0.0157**
Uspst	**0.7034 ± 0.0005**	**0.6184 ± 0.0002**	**0.5851 ± 0.0004**	**0.5361 ± 0.0005**
USPSdata_20	**0.7042 ± 0.0003**	**0.6318 ± 0.0006**	**0.5928 ± 0.0010**	**0.5446 ± 0.0012**
USPSdata	**0.7116 ± 0.0001**	**0.6159 ± 0.0001**	**0.5890 ± 0.0002**	**0.5401 ± 0.0002**
MSRA25	**0.5826 ± 0.0060**	**0.6238 ± 0.0055**	**0.4349 ± 0.0077**	**0.3760 ± 0.0089**
PalmData25	**0.8409 ± 0.0027**	**0.9392 ± 0.0010**	**0.7781 ± 0.0036**	**0.7758 ± 0.0036**
Binalpha	**0.4877 ± 0.0029**	**0.5971 ± 0.0015**	**0.3237 ± 0.0023**	**0.3044 ± 0.0024**
Ecoli	0.5346 ± 0.0062	0.5733 ± 0.0038	0.4796 ± 0.0062	0.3680 ± 0.0069
Corel_5k	**0.1904 ± 0.0007**	**0.2695 ± 0.0005**	**0.0841 ± 0.0004**	**0.0628 ± 0.0004**
MnistData_05	0.5631 ± 0.0036	0.4876 ± 0.0022	0.4188 ± 0.0015	0.3526 ± 0.0016
MnistData_10	0.5822 ± 0.0015	0.4988 ± 0.0014	0.4308 ± 0.0014	0.3651 ± 0.0016
Coil20Data_25	**0.7286 ± 0.0063**	**0.8055 ± 0.0033**	**0.6680 ± 0.0071**	**0.6500 ± 0.0075**
Mpeg7	**0.6049 ± 0.0025**	**0.7594 ± 0.0013**	**0.4488 ± 0.0030**	**0.4407 ± 0.0030**
TDT2_10	**0.4759 ± 0.0041**	**0.4769 ± 0.0044**	**0.2702 ± 0.0027**	**0.1338 ± 0.0035**

An in-depth analysis reveals that Rec-HW, with *random* seeding during recombinations, ensures that, in 9 of 16 cases (shown in bold in Table 9), Rec-HW outperforms Nie-K-Means. The situation in the remaining datasets is as follows:

(a) USPSData_20: Results of Nie-K-Means almost coincide with those of Rec-HW;

(b) Dermatology, MSRA25, MnistData_10: Rec-HW only generates better values for the ACC and the NMI indices;

(c) Ecoli: Rec-HW is better only in the NMI index;

(d) MnistData_05: Rec-HW is better only in the ACC measure;

(e) Iris: Nie-K-Means outperforms Rec-HW in all four indices. However, the CI = 0 ensured by Rec-HW in 80% of the repetitions demonstrates that the Rec-HW clustering is, in any case, correct.

The above-described positive behaviour, even when uniform random feeds the recombinations, confirms the fundamental role played by the population of candidate centroids.

In the case that *g-k-means++* replaces the *random* seeding, the observed results are as in Table 10. Point-by-point comparison with [41] reveals that now:

(a) in 13 of 16 cases (shown in bold in Table 10), Rec-HW outperforms Nie-K-Means in all four measures, ACC, NMI, F-Score, and ARI;

(b) in the Ecoli dataset, Rec-HW only generates a better NMI index;

(c) in the MnistData_05, Rec-HW only generates a better ACC value;

(d) in the MnistData_10, Rec-HW results almost coincide with those of Nie-K-Means.

As a consequence, except for two datasets, the clustering quality with careful seeding appears to be significantly better than that of Nie-K-Means.

The results of Table 10 also improve those reported in [23], where only for 10 datasets was a more accurate clustering achieved than that of the Nie-K-Means.

4.5. Sensitivity Analysis

Rec-HW operation is controlled by two parameters: J, the number of candidate solutions initially put in the population, and R, the number of repetitions/recombinations. The value of R influences the robustness of the clustering results, as captured by statistical estimates (e.g., confidence intervals) of clustering quality measures (e.g., the accuracy index ACC). The value of J depends on the dimensions (number of sample data N, number of point coordinates D) and the point distribution of a dataset. All the datasets in Table 6 were studied by using J = 20 and R = 50, which were felt adequate for the experimental work. However, a J value suitable for clustering a given dataset, can be inferred by preliminarily studying one or more clustering indices vs. J. Table 11 reports the observed ACC measure, together with the cluster index CI, the average number of trials executed per repetition avIT, and the overall Elapsed Time ET (in sec) for the Binalpha dataset (see Table 6), when J is varied from 5 to 30, by step 5, and R = 100 repetitions of Rec-HW are used. Table 11 confirms that J = 20 was an acceptable choice for studying Binalpha. In addition, the remaining very good results in Table 10, emerged by comparison with [41], indicate that J = 20 and R = 50 are also adequate for the other datasets.

Table 11. Clustering sensitivity of the Binalpha dataset of Table 6 to the J parameter. R = 100 repetitions of Rec-HW used.

J	ACC	CI	avIT	ET (s)
5	0.4804 ± 0.0024	7	7.15	62.32
10	0.4786 ± 0.0024	7	8.18	73.95
15	0.4787 ± 0.0022	7	8.4	80.4
20	0.4877 ± 0.0024	6	9.33	94.42

Table 11. *Cont.*

J	ACC	CI	avIT	ET (s)
25	0.4823 ± 0.0022	7	9.23	98.72
30	0.4789 ± 0.0025	6	8.78	102.68

5. Conclusions

The Hartigan–Wong (HW) clustering algorithm [14,15] is a variant of the K-Means algorithm [6–8], known for being less likely to become stuck in a local suboptimal solution [16,17]. At the source of this better behaviour is a replacement of the local strategy of K-Means iterations that redo point partitioning according to the nearest centroid rule (Voronoi organization), and recalculate centroids of update clusters, with trials where each dataset point is tentatively extracted from its source cluster and relocated to a different destination cluster, e.g., by ensuring the movement would cause a reduction in the overall *Sum-of-Squared-Errors* (SSE) or distortion cost (non-Voronoi organization) [16,17].

This paper develops an extension of the Hartigan–Wong algorithm, named Recombinator Hartigan–Wong (Rec-HW), that further improves the accuracy of the achievable clustering solutions. Rec-HW adopts an evolutionary, genetic approach [20,22–24] that is founded on a population of candidate solutions/centroids, obtained using a careful seeding method. Population centroids are then selected (still by a careful initialization method), crossed, and mutated by standard HW trials. New generations get created, which tend to collapse, as experimentally confirmed, into a configuration that favours the extraction of a final solution close to the optimal one.

Although trials are necessarily executed one-at-a-time sequentially, many internal recurring tasks (partitioning, operations to accelerate point movements, computation of clustering quality measures, and so forth) conveniently exploit the benefits of parallel Java [30,31] on a multi-core machine, through the systematic recourse to functional parallel streams and lambda expressions [13,19,22,23,28,39].

The paper demonstrates the effectiveness of Rec-HW through several clustering experiments directed to challenging synthetic/benchmark and realistic datasets. In a significant case, Rec-HW was successfully applied to the 16 realistic datasets used in [41] to test the efficiency of Nie-K-Means, that is, a special, careful variant of K-Means. Except for two datasets where Nie-K-Means was slightly better than Rec-HW, in all the remaining cases, Rec-HW outperformed Nie-K-Means.

The continuation of this research is geared towards the following points. First, to extend Rec-HW with the option to direct trials either to the minimization of SSE (default) or to the maximization of the Silhouette Index (SI). Due to the high $O(N^2)$ cost of SI, which is impractical in large datasets, the idea is to use the Calinski–Harabasz index (CH) [43], viewed as a similar index to SI but easier to compute, to constrain point movement to an increment of the CH. Second, to exploit density peaks of candidate centroids [23] as a further initialization method for Rec-HW recombinations. Third, to assess the clustering performance of Rec-HW by using non-parametric statistical tests like the Wilcoxon and Friedman tests. Fourth, to port the Rec-HW implementation to Python.

Author Contributions: Conceptualization, L.N.; methodology, L.N. and F.C.; software, L.N.; validation, L.N. and F.C.; investigation, L.N. and F.C.; writing—original draft preparation, L.N.; writing—review and editing, L.N. and F.C.; 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: Source Java code can be made available by request to the authors.

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

References

1. Aggarwal, C.C.; Reddy, C.K. *Data Clustering—Algorithms and Applications*; CRC Press: Boca Raton, FL, USA; Taylor and Francis Group: London, UK, 2014.
2. Wang, R. *Introduction to Machine Learning: From Math to Code*; Cambridge University Press: Cambridge, UK, 2025.
3. Nielsen, F. Partition-based clustering with k-means. In *Introduction to HPC with MPI for Data Science*; Springer International Publishing: Cham, Switzerland, 2016; pp. 163–193.
4. Aurenhammer, F.; Klein, R. *Voronoi Diagrams*; Fernuniv., Fachbereich Informatik: Berlin, Germany, 1996.
5. Koivistoinen, H.; Ruuska, M.; Elomaa, T. A Voronoi diagram approach to autonomous clustering. In *International Conference on Discovery Science*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 149–160.
6. MacQueen, J. Some methods for classification and analysis of multivariate observations. In *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability*; University of California: Berkeley, CA, USA, 1967; pp. 281–297.
7. Lloyd, S.P. Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [CrossRef]
8. Jain, A.K. Data clustering: 50 years beyond k-means. *Pattern Recognit. Lett.* **2010**, *31*, 651–666. [CrossRef]
9. Celebi, M.E.; Kingravi, H.A.; Vela, P.A. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert Syst. Appl.* **2013**, *40*, 200–210. [CrossRef]
10. Fränti, P.; Sieranoja, S. K-means properties on six clustering benchmark datasets. *Appl. Intell.* **2018**, *48*, 4743–4759. [CrossRef]
11. Fränti, P.; Sieranoja, S. How much can k-means be improved by using better initialization and repeats? *Pattern Recognit.* **2019**, *93*, 95–112. [CrossRef]
12. Fränti, P. Efficiency of random swap algorithm. *J. Big Data* **2018**, *5*, 1–29. [CrossRef]
13. 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]
14. Hartigan, J.A. *Clustering Algorithms*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1975.
15. Hartigan, J.A.; Wong, M.A. Algorithm AS 136: A K-Means clustering algorithm. *J. R. Stat. Soc. Ser. C (Appl. Stat.)* **1979**, *28*, 100–108. [CrossRef]
16. Telgarsky, M.; Vattani, A. Hartigan's method: K-Means clustering without voronoi. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, Sardinia, Italy, 13–15 May 2010*; JMLR Workshop and Conference Proceedings; PMLR: Cambridge, MA, USA; pp. 820–827.
17. Slonim, N.; Aharoni, E.; Crammer, K. Hartigan's K-means vs. Lloyd's K means—Is it time for a change? In Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI), Beijing, China, 3–9 August 2013.
18. 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]
19. Nigro, L. Parallel K-Means algorithms in Java. *Algorithms* **2022**, *15*, 117. [CrossRef]
20. Baldassi, C. Recombinator-k-means: An evolutionary algorithm that exploits k-means++ for recombination. *IEEE Trans. Evol. Comput.* **2022**, *26*, 991–1003. [CrossRef]
21. Baldassi, C. Systematically and efficiently improving K-Means initialization by pairwise-nearest-neighbor smoothing. *arXiv* **2022**, arXiv:2202.03949.
22. 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]
23. Nigro, L.; Cicirelli, F.; Pupo, F. Genetic Elitist Approach and Density Peaks to Improve K-Means Clustering. *Algorithms* **2026**, *19*, 131. [CrossRef]
24. Fränti, P. Genetic algorithm with deterministic crossover for vector quantization. *Pattern Recognit. Lett.* **2000**, *21*, 61–68. [CrossRef]
25. Nigro, L.; Cicirelli, F. Fast clustering convergence by genetic algorithm. In *International Conference on WorldS4*; Springer Nature: Singapore, 2024; pp. 331–342.
26. Rodriguez, R.; Laio, A. Clustering by fast search and find of density peaks. *Science* **2014**, *344*, 1492–1496. [CrossRef] [PubMed]
27. Sieranoja, S.; Fränti, P. Fast and general density peaks clustering. *Pattern Recognit. Lett.* **2019**, *128*, 551–558. [CrossRef]
28. Nigro, L.; Cicirelli, F. ParDP: A parallel density peaks-based clustering algorithm. *Mathematics* **2025**, *13*, 1285. [CrossRef]
29. Nigro, L.; Cicirelli, F. Evolutionary Hartigan-Wong clustering algorithm. In *Proceedings of the 10th International Conference WorldS4, London, UK, 28–30 July 2026*; Springer: Berlin/Heidelberg, Germany, 2026.
30. Urma, R.G.; Fusco, M.; Mycroft, A. *Modern Java in Action*; Manning: Shelter Island, NY, USA; Simon Schuster: New York, NY, USA, 2019.
31. Subramaniam, V. *Functional Programming in Java: Harness the Power of Streams and Lambda Expressions*; The Pragmatic Programmers LLC: Dallas, TX, USA, 2023.

32. Arthur, D.; Vassilvitskii, S. k-means++: The advantages of careful seeding. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2007; pp. 1027–1035.
33. Bradley, P.S.; Fayyad, U.M. Refining initial points for k-means clustering. In Proceedings of the Fifteenth International Conference on Machine Learning, Madison, WI, USA, 24–27 July 1998; pp. 91–99.
34. 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)*; IEEE: New York, NY, USA, 1998; Volume 3, pp. 104–108.
35. Rezaei, M.; Franti, P. Set matching measures for external cluster validity. *IEEE Trans. Know. Data Eng.* **2016**, *28*, 2173–2186. [CrossRef]
36. Fränti, P.; Rezaei, M.; Zhao, Q. Centroid index: Cluster level similarity measure. *Pattern Recognit.* **2014**, *47*, 3034–3045. [CrossRef]
37. 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 2016)*; Springer International Publishing: Cham, Switzerland, 2016; Volume 10029, pp. 285–296.
38. Franti, P.; Sieranoja, S. Clustering accuracy. *Appl. Comput. Intell.* **2024**, *4*, 24–44. [CrossRef]
39. Nigro, L.; Fränti, P. Two medoid-based algorithms for clustering Sets. *Algorithms* **2023**, *16*, 349. [CrossRef]
40. Fränti, P. Benchmark Datasets Repository. Available online: http://cs.uef.fi/sipu/datasets/ (accessed on 1 May 2026).
41. 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]
42. UCI Machine Learning Repository. Available online: http://archive.ics.uci.edu/ml (accessed on 1 May 2026).
43. Caliński, T.; Harabasz, J. A dendrite method for cluster analysis. *Commun. Stat.* **1974**, *3*, 1–27. [CrossRef]