21.3.2025

# Cluster overlap as objective function

Pasi Fränti[1], Claude Cariou[2], and Qinpei Zhao[3]

[1]*School of Computing, University of Eastern Finland, Joensuu, Finland*
franti@cs.uef.fi

[2] *Univ Rennes - IETR, Lannion, France*
claude.cariou@univ-rennes1.fr

[3] *Tongji University, Shanghai, China*
13082@tongji.edu.cn

**Abstract**

K-means uses sum-of-squared error as the objective function to minimize within-cluster distances. We show that, as a consequence, it also maximizes between-cluster variances. This means that the two measures do not provide complementary information and that using only one is enough. Based on this property, we propose a new objective function called *cluster overlap*. We adopt the new function within k-means and present an algorithm called *overlap k-means*. It is an alternative way to design a k-means algorithm. The idea is also extended to a localized variant, which can potentially find clusters of arbitrary shape.

***Keywords:*** Clustering, k-means, overlap measure, within-cluster distance, between-cluster distance, arbitrary-shape clusters

Radu's tool: https://cs.uef.fi/~radum/demonstrator/ **(For Pasi to remember)**

## 1. Introduction

Clustering aims at grouping data points $X = \{x_1, x_2, \ldots, x_n\}$ into $c$ clusters by minimizing an objective function that estimates the goodness of the clusters. We assume that there is a distance function $dist(x_i, x_j)$ between the data points. Clustering is given as the partition index $p(i)$ of which cluster the object $x_i$ is assigned to, and the clusters are represented by their centroids $c_j$. The most common objective function is to minimize the sum of squared error between the data points and their cluster centroid:

$$SSE = \sum_{i=1}^{n} dist(x_i, c_{p(i)})^2 \qquad (1)$$

K-means [Forgy1965, Macqueen1967, Lloyd1982] and many other algorithms [Fränti1997, Fränti2018, Malinen2014, Fritzke2020, LeePerkings2021, Baldassi2022] use this objective function. It requires that we can calculate the centroid (mean). It is trivial in Euclidean space but becomes more complex for distance functions designed for strings, sets, time series, and GPS trajectories [Fränti&Mariescu2021]. SSE can also be converted to an analytical function that can be optimized by gradient descent [Malinen2012], coordinate descent [Nie2021], or its split-and-merge variant [Qu2024].

Another popular class of algorithms is agglomerative clustering. The idea is to build the clustering hierarchically, starting with $n$ clusters and then merging clusters until their number reduces to $c$. Agglomerative clustering applies a local optimization strategy by considering all pairs of clusters and merging the pair, which increases the objective function the least. Its main advantages are better clustering accuracy than k-means and the fact that it is not necessary to calculate the centroids.

Regardless of the objective function, we can simply calculate the merge cost as the difference in objective function value before and after the merge:

$$\Delta f_{a,b} = f_{a+b} - f_a - f_b \tag{2}$$

where fa and $f_b$ are the objective function values of the clusters $a$ and $b$ before the merge, and $f_{a+b}$ after their merge.

Many heuristics have been used to measure the *merge cost*. The most relevant to this paper is the classical *Ward's method* [Ward1963], which minimizes the same objective function as k-means (*SSE*). In the context of vector quantization, it is known as *Pairwise nearest neighbor* (PNN) [Equitz1989]. Notably, the merge cost can be calculated only using the cluster sizes ($n_a$, $n_b$) and their centroids ($c_a$, $c_b$) in O(1) time [Equits1989]:

$$\text{MergeCost}(a,b) = \frac{n_a + n_b}{(n_a \cdot n_b)} \cdot dist(c_a, c_b)^2 \tag{3}$$

A trivial implementation of the algorithm requires O($n^3$) time even if the distance matrix is stored. This is due to the time-consuming search for the best pair [Shanbehzadeh1997].

A more efficient variant maintains only the pointer (and merge cost) of the best pair for every cluster. It reduces the time complexity close to O($n^2$) with linear memory requirement [Fränti2000]. A faster variant uses k-nearest neighbor graph, which reduces the time complexity to O($n$ log $n$) at a minor increase in the SSE-values: 0.0%, 0.2%, 2.6% (Birch datasets), and 1.0%, 2.6%, 4.3% (image datasets) [Fränti2006].

Other objective functions have also been considered. Cut-based methods include *Normalized cut* [Shi&Malik2000] and *Cut ratio* [Hagen1992] based on between-cluster distances. Graph-based methods first create a neighborhood graph and then use the links between the data points to define a cut. *Conductance* [Leskovec2010] calculates the ratio between cluster links and all links. *Modularity* [Newman2006] calculates the difference between-cluster distances and between-cluster distances. These objective functions have also been adopted into k-means when data is represented by a graph [Sieranoja&Fränti2022]. Between-cluster distances of each merge were measured during the agglomerative clustering process and then normalized by a generic function based on the cluster sizes [Cohen-Addad2019].

Objective functions also exist in cases where the number of clusters is unknown. These are called *internal cluster validity indexes* and are used to compare clustering results with different values of $c$. Examples of indexes that work reasonably well according to our experiments include *Calinski–Harabasz* [Calinski1974], *Silhouette coefficient* [Rousseeuw1987], *WB-index* [Zhao2014] and *kCE-index* [Hämäläinen2017]. They all measure the ratio of within-cluster and between-cluster distances normalized by the number of clusters in a slightly different way.

The most striking observation of the afunctions is that almost all indices are based on the same two variables: *within-cluster* and *between-cluster* distances [Zhao2014]. In this paper, we show the properties and relationship of these two variables. In particular, we show that the between-cluster

distances can be calculated directly from the within-cluster distances and vice versa. Thus, the two variables are not mutually exclusive. Instead, they measure essentially the same property.

To demonstrate their relationship, we introduce a new overlap objective function based on the between-cluster distances. To keep the computation reasonable, we simplify the function using an overlap measure. We then apply the new objective function within k-means and compare the experimental results of the new Overlap k-means against the standard k-means based on within-cluster distances. We design a k-means based algorithm for the new objective function. A localized variant is also considered for clusters with arbitrary shapes.

Our results are as follows. Between-cluster distances provide an alternative way to construct k-means with similar results but higher computational complexity. The proposed overlap measure addresses this problem by emphasizing the central points in the cluster. It has the advantage of providing more accurate centroid locations.

However, there is a drawback. The border points are important for two reasons. First, they allow centroids to travel from one cluster to another. Second, the far-away points play a significant role in pulling the centroids, enhancing their movement. The overlap k-means Without this pulling force, the algorithms become too rigid.

The rest of the paper is organized as follows. We first discuss the properties of the SSE objective function in Section 2 and show the connection between the within (inner) and between (outer) cluster distances. We then form a k-means variant using the between-cluster distances in Section 3. It has mainly theoretical interest, but we also show how it can be used to create a new localized objective function that can potentially detect arbitrary shapes. The experimental setup is defined in Section 4, and results are given in Section 5. Conclusions are drawn in Section 6.

## 2. Properties of the objective function

The intuitive goal of clustering is to minimize the distances within the clusters and maximize those between the clusters. These can be measured by the following three quantities:

- Sum of squared *within*-cluster distances (*SSW*)
- The sum of squared *between-cluster* distances (*SSB*)
- All pairwise (squared) distances (*APD*)

They can be calculated as follows:

$$SSW = \sum_{i,j} dist(x_i, x_j)^2 \;\; \forall\, i,j \mid p(i) = p(j) \tag{4}$$

$$SSB = \sum_{i,j} dist\big(x_i, x_j\big)^2 \;\; \forall\, i,j \mid p(i) \neq p(j) \tag{5}$$

APD is the sum of the other two:

$$APD = SSW + SSB = \sum_{i,j} dist(x_i, x_j)^2 \tag{6}$$

*SSW* and *SSB* depend on the clustering result, whereas *APD* is constant and depends only on the data. *SSW* can also be calculated for each cluster separately:

$$SSW = \sum_{a=1}^{c} SSW_a \tag{7}$$

where $SSW_a$ denotes the sum of squared distances within the cluster $a$:

$$SSW_a = \sum_{i,j} dist(x_i, x_j)^2 \;\; \forall\, i,j \mid p(i) = p(j) = a \tag{8}$$

### 2.1. Use of SSW and SSB

SSW is widely used in agglomerative clustering by calculating the change (delta) in SSW if any pair of clusters $a$ and $b$ are merged:

$$\Delta SSW_{a,b} = SSW_{a+b} - SSW_a - SSW_b = SSB_{a+b} \tag{9}$$

Here, it is important to notice that the change equals the sum of squared distances between the clusters, $SSB_{a+b}$, defined as follows:

$$SSB_{a+b} = \sum_{i,j} dist\left(x_i, x_j\right)^2 \forall\, i,j \mid p(i) = a \;\wedge\; p(j) = b \tag{10}$$

In other words, while the algorithm aims to minimize SSW, it proceeds merely by evaluating the between-cluster values (SSB). SSW can also be normalized by calculating the sum of distances divided by the cluster size:

$$\widehat{SSW} = \sum_{a=1}^{c} \frac{SSW_a}{n_a} \tag{11}$$

This version is used in Ward's method and is shown to equal SSE [Equitz1989]. The unnormalized version in (10) has been shown to provide more balanced cluster sizes compared to SSE [Malinen&Fränti2023]. Average linkage clustering eliminates the cluster sizes completely from the equation by dividing by $n_a \cdot n_b$. Its behavior is opposite to (10) by creating more biased cluster sizes and favoring outlier clusters.

Normalized cut [Shi&Malik2000], cut ratio [Hagen1992], and conductance [Leskovec2010] are all based on SSB but normalized in different ways. The ratio of SSW and SSB is used in [Bubeck2009], and their difference in [Newman2006]. Calinski–Harabasz [Calinski1974] and WB-index [Zhao2014] also use the ratio of SSW and SSB, but they also include $c$ in the equation to allow comparison with different numbers of clusters. This is a useful variation when the number of clusters also needs to be solved.

### 2.2. Relationship between SSE and SSW

In the case of Euclidean distance, SSE is also directly related to SSW according to Huygens's theorem [Aloise2009, Malinen&Fränti2023]:

$$SSW = \sum_{z=1}^{c} \sum_{x_i, x_j \in z} dist(x_i, x_j)^2 = \sum_{a=1}^{c} SSW_a = \sum_{a=1}^{c} n_a SSE_a = n \cdot SSE \qquad (12)$$

where $SSW_a$ and $SSE_a$ refer to the *SSW* and *SSE* values in cluster $a$, and $n_a$ its size. The main consequence of this relationship is that minimizing *SSE* and *SSW* differs only by a scaling factor $n$. In other words, *SSE* sums up $n$ squared distances, whereas SSW sums up $n^2$ distances. It is possible to normalize them into the same scale simply as $SSE = SSW/n$.

We can also reformulate equations (6) and (12) as follows:

$$SSB = APD - SSW = APD - n \cdot SSE$$

$$\Leftrightarrow$$

$$SSE = (APD - SSB)/n \qquad (13)$$

Since *APD* is constant, regardless of the clustering, minimizing *SSE* (and *SSW*) is effectively the same as maximizing *SSB*. This implies that the objective function does not need to contain *SSE* or *SSW*, but *SSB* alone is sufficient. Figure 1 shows an example of two clusters with their corresponding SSW and SSB values.

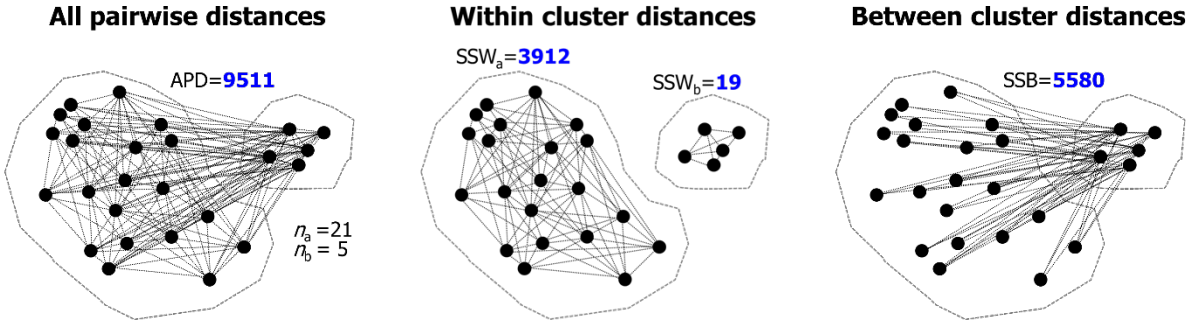


**Figure 1**: Example of *APD* (left), *SSW* (middle), and *SSB* (right). There are O($n^2$) links, but not all of them are drawn. The links have the following relationship: {*APD*} = {*SSW*} ∪ {*SSB*}.

### 2.3. Number of SSB calculations

Motivated by the observations, we reformulate the k-means algorithm using between-cluster distances (*SSB*). However, this can be time-consuming because there are O($n^2$) pairwise distances to be calculated at every iteration, which is too slow. An alternative solution is to store the O($n^2$) size distance matrix, but this would reduce the practicality of the method due to quadratic memory consumption. Neither option looks very promising for scalability.

Fortunately, most SSB distances are between points located far away from each other, so they will almost never be assigned to the same cluster. In other words, they have no contribution to clustering.

The distances that matter are points near each other so that they can sometimes belong to the same cluster depending on the operation of the algorithm. They contribute to the objective function most.

One possibility to make the method faster is to limit the distance calculations only between points in neighboring clusters. However, if the number of clusters is small ($c<<n$), almost all clusters are neighbors. This can also happen when the dimension increases high because the distances tend to become equidistant when using Euclidean ($L_2$) distance [Beyer1999]. Minkowski distances ($L_p$) with other values of $p$ have also been considered and L-norms with fractional values ($0<p<1$) have been shown to work better for high-dimensional data [Aggarwal2001].

To demonstrate the problem, assume that the points are equally divided so that all clusters have size $n/c$ points. In this case, there are $(n/c)^2$ distances between every pair of clusters. If we compute distances only between the nearest cluster, there would still be $c\cdot(n/c)^2 = n^2/c$ distance calculations. If $c = \sqrt{n}$, the number of distance calculations becomes $n^{1.5}$, but when $c$ is a small constant, $c$=O(1), the time complexity remains O($n^2$). See Fig. 2 for an example. In other words, we cannot afford to calculate all between-cluster distances if we want to keep the algorithm fast.

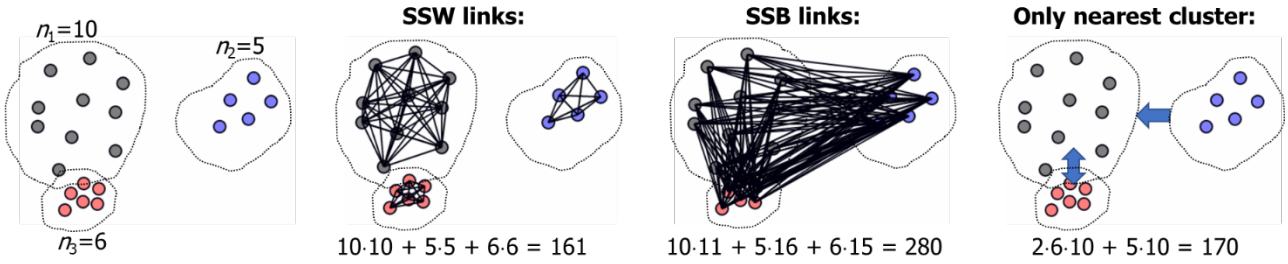


**Figure 2**: Total number of *SSB* calculations (280) and the number of calculations if only considering the nearest cluster (170).

### 2.4. Euclidean distances

When minimizing sum-of-squared *Euclidean* distances, the calculations can be done more efficiently. From (12), we know that the sum of squared distances (*SSE*), multiplied by $n$, equals all pairwise distances within the clusters (*SSW*):

$$SSW_a = n_a \cdot SSE_a \tag{14}$$

This is already utilized in k-means by using the centroids to calculate the within-cluster distances (*SSW*). However, the result also generalizes between-cluster distances as shown in [Kärkkäinen2002, Zhao2014].

$$SSB = APD - n \cdot SSE = n \cdot \frac{APD}{n} - \sum_{a=1}^{c} n_a \cdot SSE_a = \sum_{a=1}^{c} n_a \cdot \left(\frac{APD}{n} - SSE_a\right)$$
$$= \sum_{a=1}^{c} \left(\frac{n_a}{n} \cdot APD - SSW_a\right) = APD - \sum_{a=1}^{c} SSW_a \quad (15)$$

In other words, both *SSW* and *SSB* can be done in O($n$) time. In the following, we do not utilize this property as we are interested in overlap. This requires point-level information that would be lost if we used centroids to represent the clusters. However, this property can be beneficial for others, and thus, it is presented here.

## 3. Overlap k-means (OK)

Among all the point pairs, the most meaningful ones are those that are near to each other but in different clusters. In other words, border pixels. In the proposed method, we, therefore, calculate only one distance per point to its nearest neighbor in *another* cluster. This distance is put into the context of its relative location in its own cluster.

### 3.1 Overlap value

We formulate a so-called *overlap value* for each data point based on the overlap measure proposed in [Fränti&Sieranoja 2018]. The overlap score for a point $x_i$ is calculated as follows:

$$O_1(x_i) = \frac{dist\left(x_i, c_{p(i)}\right)}{dist\left(x_i, \overline{NN}(x_i)\right)} \quad (16)$$

where $\overline{NN}(x_i)$ is the nearest neighbor of $x_i$ in another cluster:

$$\overline{NN}(x_i) = \text{argmin}\, dist\left(x_i, x_j\right), \qquad \forall\, x_j \mid p(i) \neq p(j) \quad (17)$$

The lower the overlap value, the better the cluster assignment of the point. The new objective function then minimizes the sum of all overlap values in the data:

$$Overlap = \sum_{i=1}^{n} O_1(x_i) \quad (18)$$

The overlap value includes two terms. The first term (numerator) represents a sample of the *SSW* function, and the second term (divider) is a sample of the *SSB* function. The overlap is, therefore, a strongly sampled version of the Calinski–Harabasz measure [Calinski1974] and closely related to other sum-of-squared based indexes [Zhao2014]. The difference is that the overlap measure is not normalized by the cluster size.

Example calculations are shown in Fig. 3, where the large cluster (gray points) has points near the small dense cluster (red points). While all border points have high distances to the centroids, only those near the red cluster have a high overlap value.

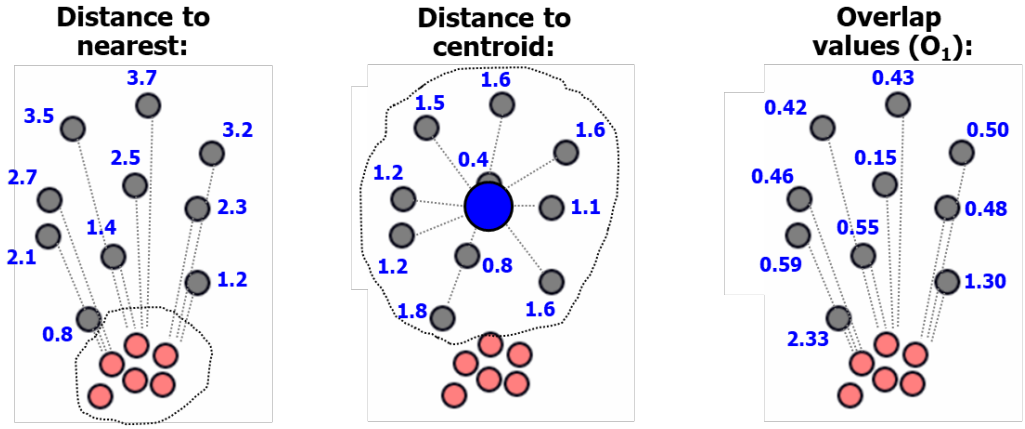


**Figure 3**: The proposed objective function calculates the distance from a point only to its nearest neighbor in another cluster and to its own centroid.

### 3.2 K-means using overlap

K-means has two steps that should be updated according to the new objective function. Theoretically, the nearest cluster should be calculated by minimizing (16). However, we face computational problems. Finding the nearest neighbor point takes O($n$) time for one point, and we have $n$ points to process. The time complexity of the partition would, therefore, grow to O($n^2$). To avoid this, we keep the original k-means assignment and find the nearest centroid.

The second step is to calculate the centroid for the clusters. We calculate the weighted average so that every point is inversely weighted by its overlap value. In this way, the border points with high overlap values will have less impact on the centroid location, which will move closer to the non-border points. The exact update rule is as follows:

$$c_j = \frac{\sum_{p(i)=j} w_i \cdot x_i}{\sum_{p(i)=j} w_i} \tag{19}$$

$$w_i = \exp[-(O_1(x_i)/\gamma)^2] \tag{20}$$

where $\gamma$ is a (constant) bandwidth parameter. In other words, the calculation of the cluster centroid is an overlap-weighted average of the points belonging to the same cluster. Otherwise, the algorithm is the same as k-means.

**Algorithm:** Overlap K-means (OK)

OK($X$, $c$) $\rightarrow \{p_i\}$
Input: $X$, $c$, $\gamma$
Output: $p_i$, $1 \leq i \leq n$, $p_i \in \{1, \ldots, c\}$

- Select $c$ points randomly from $X$ as centroids $c_j$, $1 \leq j \leq c$
- While $\exists i$: $p_i \neq p_i^{\text{old}}$
  - Step 1: For each $x_i$, assign label $p_i$ as the index of its nearest centroid;
  - Step 2: For each $x_i$, calculate the overlap value $O_1(x_i)$ according to (16);
  - Step 3: For each cluster, calculate new centroid according to (19);

### 3.3 Localized variant

Despite the alternative formulation, Overlap k-means is merely another variant of k-means that optimizes essentially the same variables: minimizing SSW and maximizing SSB. The overlap score itself is localized regarding the SSB calculations by considering only the nearest neighbor in another cluster. The motivation was merely to reduce the calculations.

Next, we extend the localization to the SSW part by removing the dependency on the centroid. We want the overlap to be based only on local properties. To achieve this, we calculate a local neighborhood of a point defined by its *k-nearest neighbors* (KNN). *Localized* overlap value is then defined as:

$$O_2(x_i) = \frac{Meanshift(x_i)}{dist\left(x_i, \overline{NN}(x_i)\right)} \tag{21}$$

where $Meanshift(x_i)$ denotes the distance of $x_i$ to the mean of its *KNN* neighbors in the *same cluster*. It is calculated as follows:

$$Meanshift(x_i) = \frac{\sum_{y \in \underline{KNN}(x_i)} dist(x_i, y)}{k} \tag{22}$$

where $\underline{KNN}(x_i)$ is the set of $k$ nearest neighbors of $x_i$ in the same cluster.

The divider is the same in both $O_1$ and $O_2$, i.e., the distance to the nearest neighbor in another cluster. However, the numerator is different. In $O_2$, it depends on the local neighborhood alone no longer requires centroid. The overlap score $O_1$ minimizes SSW, which implies spherical clusters, whereas $O_2$ minimizes *Meanshift*, which allows clusters with arbitrary non-convex shapes.

It is both an advantage and a disadvantage. The idea is that the higher the mean-shift distance, the more points differ from their neighborhoods. In specific, border points have high mean-shift values. This idea was originally developed for outlier detection in [Yang2021].

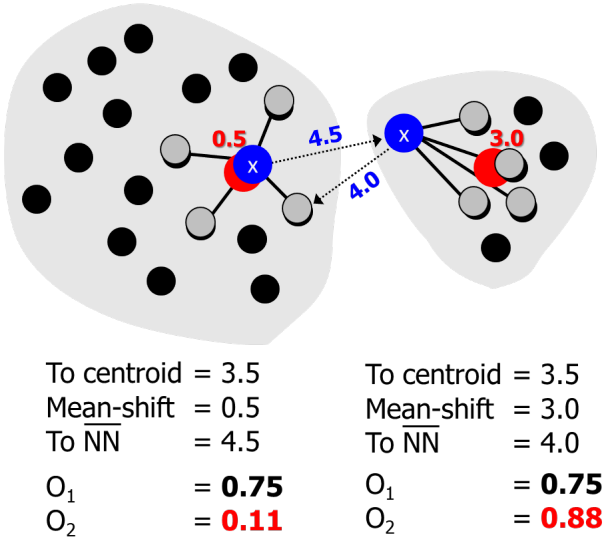

**Figure 4**: Two points (blue) having the same distance to their cluster centroid (3.5), almost the same distance to the nearest point in another cluster (4.5 and 4.0), but with very different mean shift values (0.5 and 3.0).

**Algorithm:** Localized variant (OK-local)

OK-local($X$, $k$) $\rightarrow \{p_i\}$
Input: $X$, $c$, $\gamma$
Output: $p_i$, $1 \leq i \leq n$, $p_i \in \{1, \ldots, c\}$

- Select $c$ points randomly from $X$ as centroids $c_j, 1 \leq j \leq c$
- While $\exists i$: $p_i \neq p_i^{old}$
  Step 1: For each $x_i$, assign label $p_i$ as the index of its nearest centroid;
  Step 2: For each $x_i$, calculate the overlap value $O_2(x_i)$ according to (16);
  Step 3: For each cluster, calculate new centroid according to (19);

## 4. Experimental setup

The datasets used in the experiments are listed in Table 1 and demonstrated in Figure 5. We selected datasets from the clustering basic benchmark [Fränti&Sieranoja 2018] and from [Sieranoja2019]. The datasets have varying cluster overlaps, shapes, and densities.

**Table 1:** Datasets used in the experiments.

| Dataset: | Reference: | n | d | C |
|---|---|---|---|---|
| Flame | [Fu2007] | 240 | 2 | 2 |
| Aggregation | [Gionis2007] | 788 | 2 | 7 |
| S1 | [Virmajoki2016] | 5,000 | 2 | 15 |
| S2 | [Virmajoki2016] | 5,000 | 2 | 15 |
| S3 | [Virmajoki2016] | 5,000 | 2 | 15 |
| S4 | [Virmajoki2016] | 5,000 | 2 | 15 |
| A1 | [Kärk2002] | 3,000 | 2 | 20 |
| A2 | [Kärk2002] | 5,250 | 2 | 35 |
| A3 | [Kärk2002] | 7,500 | 2 | 50 |
| Dim32 | [Fränti2006] | 1,024 | 32 | 16 |
| Unbalance | [Rezaei2016] | 6,500 | 2 | 8 |
| Birch1 | [Zhang1997] | 100,000 | 2 | 100 |
| Birch2 | [Zhang1997] | 100,000 | 2 | 100 |
| Birch3 | [Zhang1997] | 100,000 | 2 | 100 |
| Worms-2d | [Sieranoja2019] | 105,600 | 2 | 35 |
| Worms-64d | [Sieranoja2019] | 105,600 | 64 | 25 |

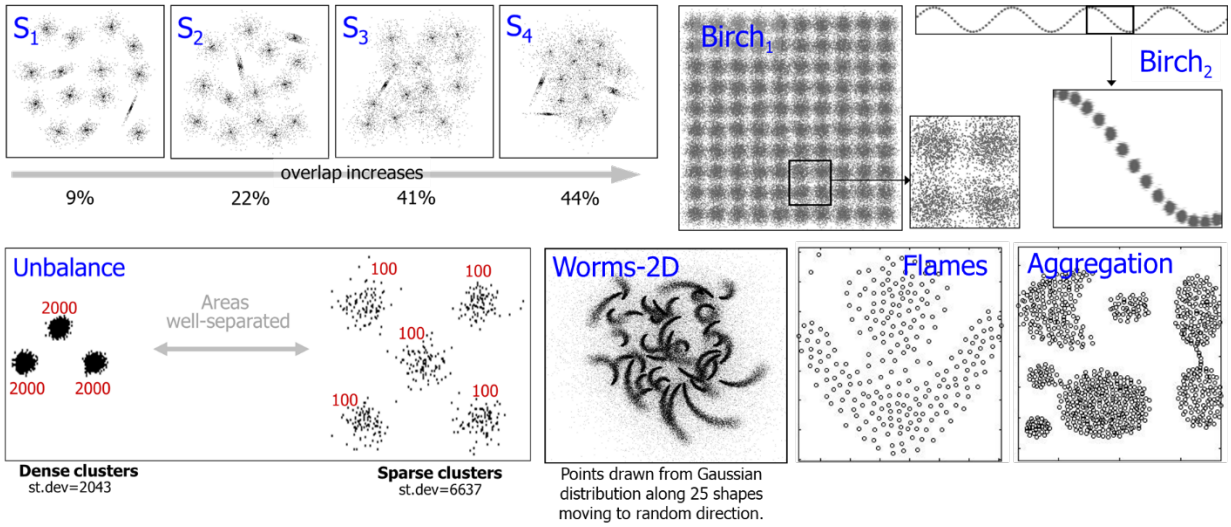


**Figure 5**: Visualization of the selected datasets. Some datasets are seemingly simple with regular structure (Birch1-3), whereas others have challenges like varying cluster overlap (S1-S4), density (Unbalance), or shapes (Worms-2d, Flames, Aggregation). REPLACE FLAMES BY WORMS64

We cluster the datasets with the following algorithms:

- K-means (KM)
- K-means++ (KM++)
- Random swap (RS)
- Density peaks (DensP)
- Overlap k-means (OK)
- Localized variant (OK-local)

K-means is the main point of comparison to see whether Overlap k-means provides similar clustering results to the standard k-means with the same seeding method. K-means itself can be improved by simple tricks like repeats and better seeding [Fränti&Sieranoja2019]. We, therefore, also tested the most popular seeding called K-means++ [Arthur&Vasilievski2007], which is based on a randomized further point heuristic.

We also include random swap [Fränti2018], which is almost as simple as k-means and provides a state-of-the-art reference to how good clustering accuracy one can expect by minimizing SSE. It is one of the few clustering algorithms that find the correct cluster allocation for all the benchmark data. Another algorithm included is Density peaks [Rodriquez2014], which can also potentially cluster non-spherical datasets.

We run all algorithms 100 times and report the average results, with the exceptions of Birch and Worms datasets, which are repeated only 10 times due to their higher processing times. Clustering results are measured by clustering accuracy (ACC) [Fränti2024] and centroid index (CI) [Fränti2016].

We have two main hypotheses to test: (1) Does OK provide similar results to k-means, and (2) can OK-local find clusters having non-uniform shapes and density?

## 5. Results

The main results are summarized in Tables 2 and 3. Good algorithms like Random swap and Density peaks can find correct clustering (CI=0) for most datasets. They only fail with the non-spherical datasets Birch3, Worms, and Aggregation, which cannot be clustered correctly by the SSE objective function. K-means++ manages to solve Dim32 and Unabalance, whereas K-means does not solve any of the datasets (average CI=5.8).

Beyond the clustering accuracy, the interest here is more on the comparison between k-means (KM) and its overlap variant (OK). The results of these two are indeed similar, but there is a noticeable difference favoring k-means. The localized variant provides results closer to k-means. The corresponding average CI values are 5.8 (K-means), 7.6 (OK), and 5.8 (Local-OK).

**Table 2:** Clustering results measured by centroid index (CI)$^{\downarrow}$.

| | S1 | S2 | S3 | S4 | A1 | A2 | A3 | Dim32 | Unb | B1 | B2 | B3 | W2 | W64 | Agg | **Av** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **KM** | 2.0 | 1.1 | 1.2 | 1.3 | 2.0 | 4.2 | 6.5 | 3.5 | 3.5 | 8.8 | 16.1 | 24 | 9.0 | 10.2 | 0.7 | 5.8 |
| **KM++** | 0.2 | 0.5 | 0.6 | 0.4 | 0.6 | 0.7 | 1.7 | **0.0** | **0.0** | 3.3 | 1.5 | 21 | 8.2 | 9.5 | 0.7 | 3.1 |
| **RS** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | 16 | 7.0 | 3.0 | 1.0 | 1.7 |
| **DensP** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | **0.0** | 21 | 8.0 | 3.0 | 1.0 | 2.1 |
| **OK** | 2.3 | 2.3 | 2.5 | 2.2 | 2.9 | 6.0 | 7.5 | 4.0 | 4.0 | 12 | 20 | 31 | 11 | 11.3 | 1.8 | 7.6 |
| **OK-local** | 2.1 | 1.2 | 1.3 | 0.9 | 2.5 | 4.7 | 5.6 | 3.7 | 4.0 | 8.3 | 16 | 24 | 8.0 | 9.9 | 1.0 | 5.8 |

**Table 2:** Clustering results measured by clustering accuracy (ACC)$^{\uparrow}$ in %.

| | S1 | S2 | S3 | S4 | A1 | A2 | A3 | Dim32 | Unb | B1 | B2 | B3* | W2 | W64 | Agg | **Av** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **KM** | 87 | 90 | 80 | 74 | 90 | 87 | 86 | 82 | 60 | 87 | 84 | - | 51 | 52 | 82 | **74** |
| **KM++** | 98 | 94 | 83 | 78 | 97 | 98 | 96 | **100** | **100** | 95 | 98 | - | 52 | 54 | 83 | **82** |
| **RS** | 99 | 97 | 86 | 80 | **100** | **100** | **100** | **100** | **100** | 97 | 100 | - | 52 | 76 | 83 | **85** |
| **DensP** | 99 | 97 | 85 | 79 | 98 | 99 | 99 | 95 | **100** | **100** | **100** | - | 55 | 74 | 86 | **85** |
| **OK** | 83 | 82 | 74 | 71 | 84 | 82 | 84 | 75 | 58 | 83 | 78 | - | 50 | 48 | 73 | **68** |
| **OK-local** | 86 | 89 | 79 | 76 | 87 | 86 | 88 | 77 | 60 | 88 | 84 | - | 52 | 53 | 83 | **73** |

*Ground truth partitions are not available.

Our first hypothesis was that Overlap k-means would provide similar results to k-means, but the results do not support it. We analyze the reasons in Figure 6 with the S1 dataset, which is easy for good algorithms but troublesome for k-means due to small cluster overlap [Fränti&Sieranoja2018]. The results show that k-means can move all centroids to their correct location except two. Overlap k-means, however, performs even worse with four clusters unsolved.

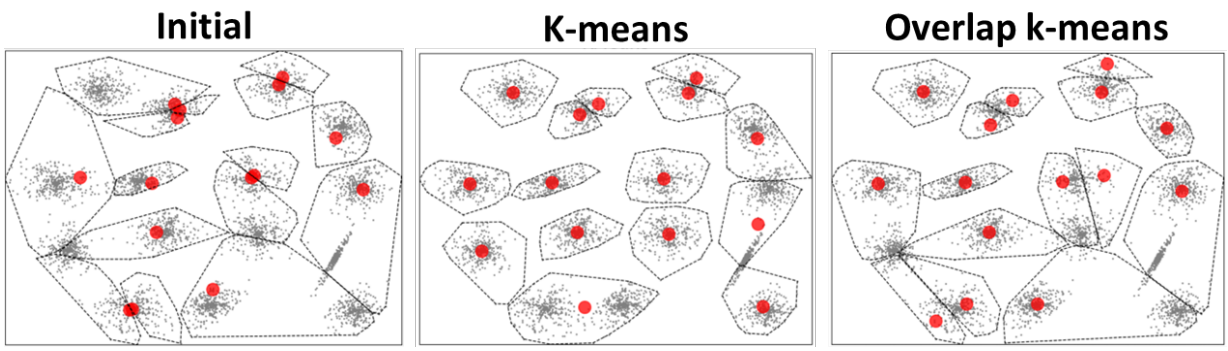


**Figure 6**: K-means and Overlap k-means result with S1 for the same initial solution (left). Overlap k-means positions the centroids better within the clusters but has problems locating them correctly globally. The CI-values of the solutions are CI=2 (k-means) and CI=4 (overlap k-means) accordingly.

The reason is studied in Figure 7, where the pulling effect of the faraway points attracts a centroid in just three iterations with k-means. However, the same does not happen with Overlap k-means. The reason is due to the low weight of the faraway in the centroid calculation. This leads to nicely balanced locations of the centroids *within* the same cluster but leaving the one above empty.

We conclude that it is possible to design k-means via the between-cluster distances, but the chosen design is not good. It makes the algorithm practical by avoiding the overwhelming computations of the between-cluster distances, but it significantly reduces the pulling effect of the remote points, which is vital for k-means. In other words, the algorithm becomes too rigid and tends to get stuck in a local minimum. An effective algorithm should do the opposite: enhance the vital faraway boundary points instead of restricting their use.

The second hypothesis was that the localized variant could also work for non-spherical datasets. This does not happen either. The results of OK-local are closer to k-means than those of OK, but they do not perform well with the non-spherical datasets (Birch3, Worms, and Aggregation) significantly better. We cannot conclude whether the idea is not good enough or the potential is not fulfilled by the inferior local optimization.

We did not carry out further tests other than this proof-of-concept. The main contribution is theoretical, showing the connection between the within-cluster and between cluster distances. If one wants to achieve a better algorithm based on the overlap concept, we outline two potential ways to do it here.

1. Design a stochastic variant starting with the opposite: give high weight for the high overlap points. This would further enhance the dynamics of k-means, hopefully pushing the centroids even further than the standard SSE function would do. Then, decrease the weights and gradually converge to the Overlap k-means variant, which would focus on the central points.

2. Replace k-means with a better local optimizer: random swap. This would completely avoid the problem of local minima. If there are any benefits of the OK or OK-local, their random swap counterparts have a higher potential to exploit them.

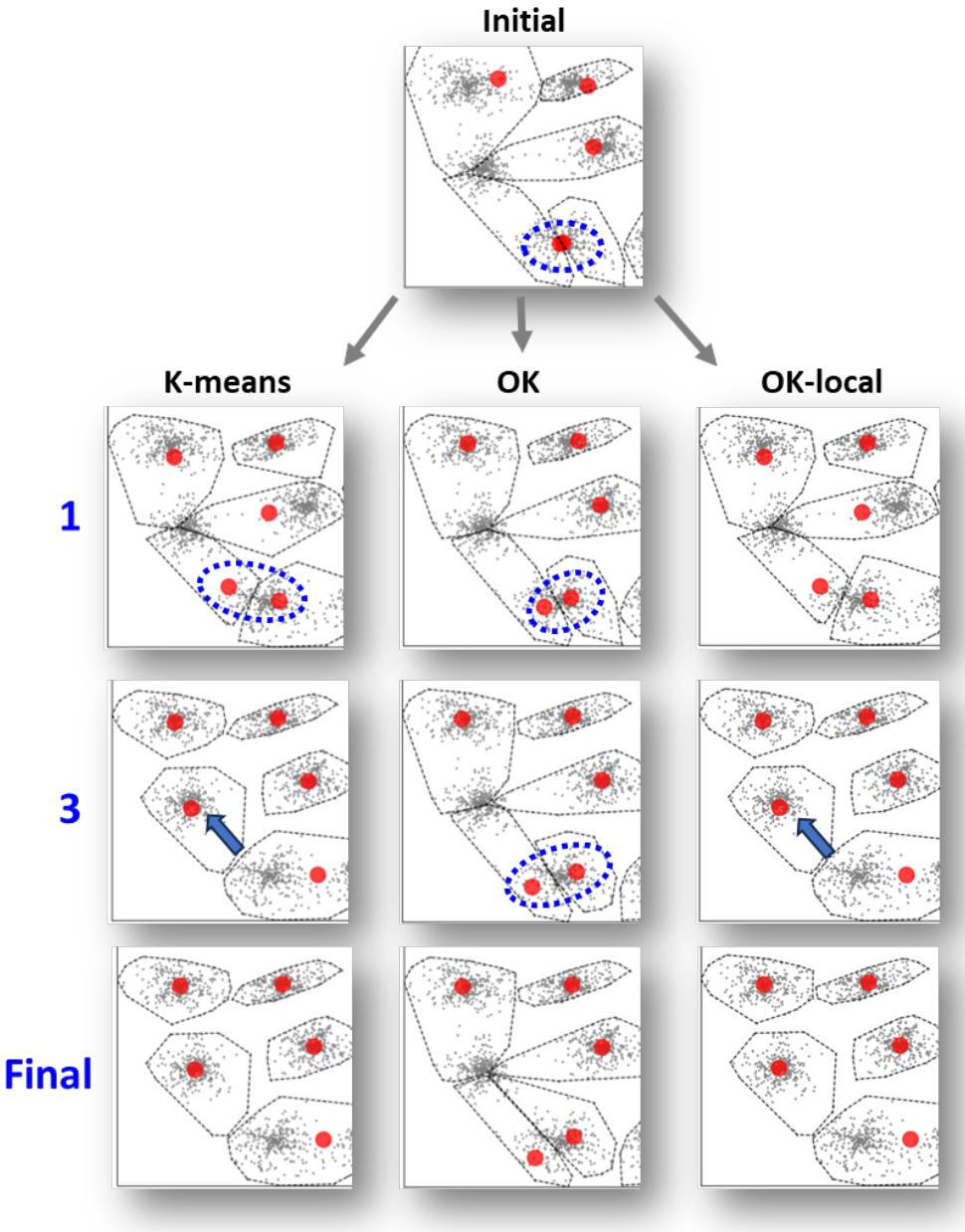


**Figure 7**: Snapshots of the bottom left corner of S1 after the 1st iteration, 3rd iteration, and the final solution. The pulling effect of the faraway points in k-means manages to move one of the centroids to the cluster missing a centroid. Overlap k-means gives less weight to the border points, which stabilizes the centroids close to their original location and makes the algorithm less dynamic.

## 6. Conclusions

We have the following conclusions:

First, within-cluster (SSW) and between-cluster distances (SSB) measure the same thing effectively and do not offer any complementary information. K-means uses the sum-of-squared

(SSE) objective function, which equals SSW. We have shown that the same objective can be constructed using between-cluster distances.

Second, if we calculate the centroids for the data, both approaches can be calculated in O($n$) time [Kärkkäinen2002, Zhao2014]. Otherwise, we need to calculate all pairwise distances. Within-cluster distances (SSW) take $k{\cdot}(n/k)^2 = O(n^2/k)$ time in case of balanced cluster sizes. Between-cluster distances, on the other hand, require the calculation of all pairwise distances in the entire data, which leads to $O(n^2)$ time complexity.

Third, we presented a new algorithm called overlap k-means (OK), which utilizes between-cluster distances. Instead of minimizing all distances, it focuses on the distances near the cluster borders. However, this construction lost the dynamics of the k-means algorithm, leading to a more rigid algorithm behavior. The results of the localized variant were more consistent with that of k-means.

Fourth, the overlap idea has further potential. We outlined two possibilities. The first one would be a stochastic variant, which would emphasize the border points more in the beginning and gradually reduce their effect, leading to the Overlap k-means. The second possibility would be simply to replace the inferior k-means with a more robust random swap algorithm. These two are points for future studies.

## References


1. E. Forgy, "Cluster analysis of multivariate data: efficiency vs. interpretability of classification", *Biometrics*, 21 768–780, 1965.
2. J. MacQueen, "Some methods for classification and analysis of multivariate observations", *Berkeley symposium on Mathematical Statistics and Probability*. 281-297, 1967.
3. S.P. Lloyd, "Least squares quantization in PCM", *IEEE Trans. Information Theory*, 28 (2), 129–137, 1982.
4. P. Fränti, J. Kivijärvi, T. Kaukoranta and O. Nevalainen, "Genetic algorithms for large scale clustering problems", *The Computer Journal*, 40 (9), 547-554, 1997.
5. P. Fränti, "Efficiency of random swap clustering", *Journal of Big Data*, 5 (1), 2018.
6. M.I. Malinen, R. Mariescu-Istodor, P. Fränti, "K-means*: clustering by gradual data transformation", *Pattern Recognition*, 47 (10), 3376-3386, 2014.
7. B. Fritzke, Breathing k-means, arXiv preprint arXiv:2006.15666, 2020.
8. J. Lee, and D. Perkins, A simulated annealing algorithm with a dual perturbation method for clustering. *Pattern Recognition*, 112, 107713, 2021.
9. C. Baldassi, "Recombinator K-means: an evolutionary algorithm that exploits k-means++ for recombination". *IEEE Trans. on Evolutionary Computation*, 26 (5), 991-1003, 2022.
10. P. Fränti and R. Mariescu-Istodor, "Averaging GPS segments: competition 2019", *Pattern Recognition*, 112, 107730, April 2021.
11. M.I. Malinen and P. Fränti, Clustering by analytic functions, *Information Sciences*, 217, 31-38, December 2012.
12. F. Nie, J. Xue, D. Wu, R. Wang, H. Li, and X. Li, Coordinate descent method for k-means. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 44(5): 2371-2385, 2021.
13. F. Qu, Y. Shi, Y. Yang, Y.Hu, Y. Liu, Coordinate descent K-means algorithm based on split-merge, *CMC-Computers, Materials & Continua*, 81 (3), 2024.
14. J.H. Ward, "Hierarchical Grouping to Optimize an Objective Function", *Journal of the American Statistical Association*, 58, 236–244, 1963.
15. W.H. Equitz, "A New Vector Quantization Clustering Algorithm," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, no. 10, pp. 1568-1575, Oct. 1989.

16. J. Shanbehzadeh and P.O. Ogunbona, “On the Computational Complexity of the LBG and PNN Algorithms”, *IEEE Trans. Image Processing*, 6 (4), 614-616, Apr. 1997.
17. P. Fränti, T. Kaukoranta, D.-F. Shen and K.-S. Chang, "Fast and memory efficient implementation of the exact PNN", *IEEE Trans. on Image Processing*, 9 (5), 773-777, May 2000.
18. P. Fränti, O. Virmajoki and V. Hautamäki, "Fast agglomerative clustering using a k-nearest neighbor graph", *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 28 (11), 1875-1881, November 2006.
19. J. Shi, J. Malik, Normalized cuts and image segmentation, *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 22, 888–905, 2000.
20. L. Hagen, A. B. Kahng, New spectral methods for ratio cut partitioning and clustering, *IEEE Trans. on Computer-Aided Design*, 11, 1074–1085, 1992.
21. J. Leskovec, K. J. Lang, and M. Mahoney, ”Empirical comparison of algorithms for network community detection”. *ACM Int. Conf. on World wide web*, 631–640. 2010.
22. M.E. Newman and M. Girvan, “Finding and evaluating community structure in networks”, *Physical Review E*, 69(2): 026113, 2004.
23. S. Sieranoja and P. Fränti, “Adapting k-means for graph clustering”, *Knowledge and Information Systems* (KAIS), 64. 115-142, 2022.
24. V. Cohen-addad, V. Kanade, F. Mallmann-trenn, and C. Mathieu. “Hierarchical clustering: objective functions and algorithms. *Journal of ACM*, 66 (4), Article 26, 2019.
25. T. Calinski, and J. Harabasz. “A dendrite method for cluster analysis”. *Communication in statistics*, 3, 1-27, 1974.
26. P. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, *J. Comput. Appl. Math.,* 20 (1987), 53–65.
27. Q. Zhao and P. Fränti, "WB-index: a sum-of-squares based index for cluster validity", *Data & Knowledge Engineering*, 92, 77-89, July 2014.
28. J. Hämäläinen, S. Jauhiainen, T. Kärkkäinen, “Comparison of internal clustering validation indices for prototype-based clustering”, *Algorithms*, 10 (3), 105, 2017.
29. M.I. Malinen and P. Fränti, “All-pairwise squared distances lead to more balanced clustering”, *Applied Intelligence and Computing*, 3 (3), 93-115, 2023.
30. S. Bubeck and U. von Luxburg. Nearest neighbor clustering: A baseline method for consistent clustering with arbitrary objective functions. *Journal of Machine Learning Research*, 10:657–698, 2009.
31. D. Aloise, A. Deshpande, P. Hansen, P. Popat, "NP-hardness of Euclidean sum-of-squares clustering", *Machine Learning*, 75, 245–248, 2009.
32. K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? *International Conference on Database Theory*. Springer, 217–235, 1999.
33. C.C. Aggarwal, A. Hinneburg, and D.A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. *International Conference on Database Theory (ICDT’01)*. Springer-Verlag, London, UK, 422–434, 2001.
34. I. Kärkkäinen and P. Fränti, "Dynamic local search algorithm for the clustering problem", *Research Reports*, A-2002-6, University of Joensuu, 2002.
35. P. Fränti and S. Sieranoja, "K-means properties on six clustering benchmark datasets", *Applied Intelligence*, 48 (12), 4743-4759, December 2018.
36. J.W. Yang, S. Rahardja and P. Fränti, "Mean-shift outlier detection", *Pattern Recognition*, 115, 107874, July 2021.
37. S. Sieranoja and P. Fränti, “Fast and general density peaks clustering”, *Pattern Recognition Letters*, 128, 551-558, 2019.
38. L. Fu and E. Medico, FLAME, a novel fuzzy clustering method for the analysis of DNA microarray data. *BMC - bioinformatics*, 8(1): p. 3, 2007.
39. A. Gionis, H. Mannila, and P. Tsaparas, Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1): p. 1-30, 2007.
40. P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems", *Pattern Recognition*, 39 (5), 761-765, May 2006.
41. M. Rezaei and P. Fränti, "Set-matching measures for external cluster validity", *IEEE Trans. on Knowledge and Data Engineering*, 28 (8), 2173-2186, August 2016.

42. T. Zhang, R. Ramakrishnan, M. Livny, "BIRCH: A new data clustering algorithm and its applications", *Data Mining and Knowledge Discovery*, 1 (2), 141-182, 1997.
43. P. Fränti and S. Sieranoja, "How much k-means can be improved by using better initialization and repeats?", *Pattern Recognition*, 93, 95-112, 2019.
44. D. Arthur, S. Vassilvitskii, "K-means++: the advantages of careful seeding", *ACM-SIAM Symp. on Discrete Algorithms* (SODA'07), January 2007.
45. A. Rodriguez, and A. Laio, Clustering by fast search and find of density peaks. *Science*, 344(6191), 1492-1496, 2014.
46. P. Fränti and S. Sieranoja, "Clustering accuracy", *Applied Computing and Intelligence*, 4 (1), 24-44, 2024.
47. P. Fränti, M. Rezaei and Q. Zhao, "Centroid index: cluster level similarity measure", *Pattern Recognition*, 47 (9), 3034-3045, 2014.

## Appendix: Pseudo codes of the algorithms

```
X = dataset
K = Size of KNN
NN = Nearest neighbors
NC = number of clusters
```

### 1) Overlap K-means (OK)

```
PerformOKM(X, NC, K) {
    /* initial solution */
    NN := FindKNN(X,K);
    C := SelectRandomRepresentatives(X,NC);
    P := OptimalPartition(C,X);
    P_old := zeros(size(P));
    O := Overlap-v1(P,X,NN,C);
    gamma = Mean(O);
    WHILE P <> P_old
        {
         P_old = P;
         (P,C,O) := OK-means(P,C,X,O,NN,gamma);
        }
    RETURN (P,C,O);
}

OK-means(P,C,X,O,NN,gamma) {
    /* performs one K-means iteration with KNN-overlap weighting */
        C := OptimalRepresentatives_w_Overlap(P,X,O,gamma);
        P := OptimalPartition(C,X);
        O := Overlap-v1(P,X,NN,C);
        }
    RETURN (P,C,O);
}


OptimalPartition(C,X) {
    FOR i := 1 TO N DO
        {
        P[i] := FindNearestRepresentative(C,X[i]);
        }
    RETURN P;
}

FindNearestRepresentative(C,x) {
    j := 1;

    FOR i := 2 TO k DO
        {
        IF Dist(x,C[i]) < Dist(x,C[j]) THEN
           {
           j := i;
           }
        }
    RETURN j;
}

Overlap-v1(P,X,NN,C) {
    FOR i := 1 TO N DO
        {
         B := NN[i];
         Q := Points in X with label P[i];
```

```
        T := B ∩ (X \ Q) /*The nearest neighbors having a different label
         D2 := SmallestDistance(T,X[i]); /*Smallest distance between X[i] and T*/
         D1 := Dist(X[i],C[P[i]]);       /*Dist. between X[i] and its centroid*/
         O[i] := D1/D2;                  /*Overlap cost*/
        }

    RETURN O;
}


OptimalRepresentatives_w_Overlap(P,X,O,gamma) {
    /* initialize Sum[1..k] and Count[1..k] by zero values! */

    /* sum vector and count for each partition */
    FOR i := 1 TO N DO
        {
        j := P[i];
        Sum[j] := Sum[j] + X[i]*exp(-(O[i]/gamma)**2);
        Count[j] := Count[j] + exp(-(O[i]/gamma)**2);
        }

    /* optimal representatives are average vectors */
    FOR i := 1 TO k DO
        {
        IF Count[i] <> 0 THEN
           {
           C[i] := Sum[i] / Count[i];
           }
        }

    RETURN C;
}

SmallestDistance(Data,Query) {

    Dmin := some large positive value;
    FOR i := 1 TO N DO
        {
        D := Dist(Data[i],Query);
        IF D < Dmin THEN
           {
           Dmin := D;
           }
        }
    RETURN Dmin;
}

AvgDistance(D) {
    Sum = 0;
    FOR i := 1 TO K DO
        {
        Sum = Sum + D[i];
        }
    Davg := Sum/K;
    RETURN Davg;
}
```

## 2) Localized variant (OK-local)

```
PerformOK-local(X, NC, K) {
    /* initial solution */
    NN := FindKNN(X,K);                        /* the array of KNNs  */
    C := SelectRandomRepresentatives(X, NC);
    P := OptimalPartition(C,X);
    P_old := zeros(size(P));
    O := Overlap_v1(P,X,NN,C);
    gamma = Mean(O);
    WHILE P <> P_old
        {
         P_old = P;
         (P,O) := OKNN(P,NN,X,O,gamma);
        }
    RETURN (P,O);
}

OKNN(P,NN,X,O,gamma) {
    /* performs one KNN iteration with KNN-overlap weighting *
        P := NNPartition(O,NN,P,gamma);
        O := Overlap_v2(P,X,NN);
        }
    RETURN (P,O);
}

NNPartition(O,NN,P,gamma) {
    FOR i := 1 TO N DO
        {
        nn := NN[i];
        W := exp(-(O[nn]/gamma)**2);  /* overlap-based weights*/

        Pnew[i] := WeightedMode(P[nn],W);
        }

    RETURN Pnew;
}

WeightedMode(P,W) {
    K := Length(P);  /* the number of neighbors */
    D := zeros(max(P),1);
    FOR n := 1 TO K DO
        D[P[n]] := D[P[n]] + W[n];
    Pnew := argmax(D);

RETURN Pnew;
}


Overlap_v2(P,X,NN) {
    FOR i := 1 TO N DO
        {
        B := NN[i];
        Q := {x ∈ X | P[x] == P[i]}; /*Find subset of points in X with label P[i]*/
        T1 := B ∩ Q (the nearest neighbors having the same label)
        T2 := B ∩ (X \ Q) (the nearest neighbors having a different label)
         D2 := SmallestDistance(T2,X[i]); /*Smallest distance between X[i] and T2*/
         D1 := Dist(X[i],Mean[X[T1]]);       /*Dist. between current pt and its
centroid*/
         O[i] := D1/D2;                    /*Overlap cost*/
        }
```

```
    RETURN O;
}
```

**Leftover:**

48. E. Schubert, J. Sander, M. Ester, H.P. Kriegel, and X. Xu. DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst*. 42, 3, Article 19, 2017.

Init and final values (SSE and/or CI) could be plotted as in Fig.6 here: http://cs.uef.fi/sipu/pub/RLS-PAA2000.pdf

Then instead of standard deviation could be reported max and min values (or 25% and 75% quartals).