1. WRS Algorithm：

WRS (D)

Input: All characteristic indicators of a data set

Output: A feature subset of a data set

BEGIN

// Initialization

$w=\{\varepsilon_1,\varepsilon_2,\cdots,\varepsilon_n\}$ // Initialization of the Weights of Characteristic Indicators

$cluster=\phi$ // Initialization of clustering collections

T // Initialization iterations' number

I //Initialized cluster members' number

S // Initialized the number of features in feature subset

for $t<T$ *or* *OCQ-NMI* tends to be stable *do*

for $i<I$ *do*

features = sample(w, S).

//Select feature subsets randomly from feature indicators based on weight

k = getClusterK(features)

// Calculates the number of clusters, as detailed in Algorithm 2

c = kMedoids(features, k)

// Get clustering members from *K-Medoids* . Detailed algorithm is shown in algorithm 3

Save the cluster members in the cluster set

OCQ-NMI = getOCQNMI(cluster)

//Calculate the OCQ-NMI index of the iterated clustering group

Update the set of weight *w* according to OCQ-NMI

END

2. getClusterK (D)

getClusterK (D)

Input: data set to be clustered

Output: Optimal number of clusters

BEGIN

//Initialization

N //Sample number of data sets

$k_{max}=\sqrt{N}$ //Calculate the maximum number of clusters

k_{Dunn} = *getKFromDunn (D)* //Calculate the number of clusters based on Dunn index

k_{CH} = *getKFromCH (D)* //Calculate the number of clusters based on CH index

k_{DB} = *getKFromDB (D)* //Calculate the number of clusters based on the DB index

Choose the maximum value not greater than k_{max} from k_{Dunn}, k_{CH} and k_{DB} as the optimal

number of clusters.
END

3. kMedoids (D)

kMedoids (D)

Input: Data set to be clustered, optimal number of clusters K
Output: Clustering results
BEGIN
//Initialization

```
α = {ζ_1, ζ_2, ···, ζ_k}        // Select k initial centers at random
π = {π_1, π_2, ···, π_k}        // Select k clusters at random
E                               // Random cost set
    for i < N do
  The cluster π_j is assigned according to the distance between data point x_i and initial cluster center ζ_j.
for α change do
    for i < k do
        for x_j ∈ π_i do
                Compute E(x_j, ζ_i) and add collection E
Select the lowest cost point in set E that is greater than the threshold value to exchange with the origin center point in set α
END
```

4. getSWT (C_m^i, C_m^j, CT) to Compute the Similarity of Clusters

getSWT (C_m^i, C_m^j, CT)

Input: Cluster C_m^i and C_m^j ， Triple Set CT of Cluster C_m^i and C_m^j
Output: The similarity of Cluster C_m^i and C_m^j
BEGIN
// Initialization

```
CT                  // Triple Set
    for C_l^t ∈ CT do
            Calculated W_{C_m^i}^{C_l^p} 和 W_{C_m^i}^{C_l^p} according to formula 9 in the article
        Get the weights of two clusters: W = min(W_{C_m^i}^{C_l^p}, W_{C_m^j}^{C_l^p})
        WSum += W
        if W > W_MAX
            W_MAX = W
    return WSum / W_max
```

END

5. getS (π_m, x_i, x_j, Π)to Compute the Similarity of Data Points

getS (π_m, x_i, x_j, Π ***)***

Input: Cluster Members π_m, Data Points x_i, x_j , Cluster Collection Π

Output: Similarity of data points x_i, x_j within cluster member π_m

BEGIN

// Initialization

CT // Triple Set

Find clusters C_m^i and C_m^j in cluster member π_m where data points x_i and x_j are located respectively.

for $\pi_k \in \Pi$ *and* $\pi_k \neq \pi_m$ *do*

    for $C_k^i \in \pi_k$ *do*

        if cluster C_m^i 、 C_m^j and C_k^i make up a triple

            Put cluster C_k^i into set CT

if $C_m^i = C_m^j$

    return 1;

else

    return $getSWT(C_m^i, C_m^j, CT)$

END

6. getS (π_m, x_i, x_j, Π)to Compute the Similarity of Data Sets

WCT (Π, X ***)***

Input: Cluster Collection Π, Data Set X

Output: Clustering Collective CTS Similarity Matrix S

BEGIN

// Initialization

N // Number of data points in a data set

M // Number of Cluster Members Included in Cluster Collection

S // CTS Similarity Matrix

    for i *in* $1:N$ *do*

    for j *in* $i:N$ *do*

        sum = 0;

        for m *in* $1:M$ *do*

            $s = getS(\pi_m, x_i, x_j, \Pi);$

            sum += s;

```
        S[i, j] = S[j,i] = sum / M ;
END
```