BIOINFORMATION open access

Supplementary material:

Methodology:

The Improved Harmony Search Algorithm

In this method the harmony memory is initialized with *n* number of candidate solutions in a systematic way, where *n* is the number of columns in the GE matrix. Each row in the harmony memory corresponds to a specific clustering solution represented by *k* columns. Each column represents a cluster and the value corresponding to the column indicates the index of the data-point that represents the centroid of the corresponding cluster.

First of all, the first column of the GE matrix (*Sample-IDs* or *Gene-IDs,* as the case may be) is sorted based on the value of the data items present in the second column (expression levels *of the gene in the samples* or *in the sample of the genes*). The sorted data items are then divided into *k* equal sets. The index of the data point which comes at the middle of each set is selected as the centroid of that cluster. A set of *k* such centroids will constitute a candidate solution. The above procedure is repeated for all the *n* columns of the Gene Expression matrix and the harmony memory is initialized with *n* candidate solutions thus obtained. Then the fitness values of all the solutions are evaluated as the Average Distance of Data points to Cluster centroids (ADDC). This value is measured as

$$ADDC = \frac{1}{k}\{\sum_{i=1}^{k}\left(\frac{\sum_{j=1}^{ni} D(ci, dij)}{ni}\right)\}$$

where *k* is the number of clusters, *ni* is the number of data items in each cluster and *D(ci, dij)* is the distance of each data point to the corresponding cluster centroid. A new solution is then improvised from the harmony memory using the approach described in [10]. The fitness of the new solution is determined by computing the ADDC value. If this solution is better than the worst solution in the harmony memory, then the harmony memory is updated by replacing the worst solution with the newly generated solution. This improvisation step is repeated for MI (Maximum Iterations) number of iterations. Finally the best solution in the harmony memory is returned as the initial centroids of the clusters.

Phase 1: Improved Harmony Search Algorithm for finding the Initial Centroids

Input:

Gene Expression data matrix, D (Samples * Genes) / / *m+1* rows * *n+1* columns

k / / Number of desired clusters.

HMCR, PAR, MI / / Harmony search optimization parameters

Output:

A set of *k* initial centroids

Steps:

1. For each column of the data matrix, *i = 2* to *n+1*
   - 1.1 Sort the first column of the data matrix based on column *i*;
   - 1.2 Divide the sorted column into *k* equal parts;
   - 1.3 For each part *j =1* to *k*, determine the index of the data item at the middle.
     - 1.3.1 Initialize the harmony memory *HM[i-1][j]* with the middle index.

   Endfor

   Endfor
2. Calculate the fitness value of all the candidate solutions in the harmony memory.
3. For *p = 1* to *MI*

   3.1 Improvise a new solution from the harmony memory.

   3.2 If the new solution is better than the worst solution in the harmony memory, replace the worst solution with the new solution.

   Endfor
4. Return the best solution in the harmony memory as the set of initial centroids.

Assigning Data Points to Clusters

After determining the initial centroids using the Improved Harmony Search algorithm, the data items are assigned to the clusters with the nearest centroids using a variant of the method followed in [6].

Phase 2: Algorithm for assigning data-points to the clusters

Input:

D = {d_1, d_2,.....,d_n} / / set of *n* data-points.

C = {c_1, c_2,.....,c_k} / / set of *k* centroids

Output:

A set of *k* clusters

Steps:

1. Compute the Euclidean distance of each data-point *di* (1<=i<=n) to all the centroids *cj* (1<=j<=k) as *d(di, cj)*;
2. For each data-point *di*, find the closest centroid *cj* and assign *di* to cluster *j*.
3. Set ClusterId[i]=j; / / j:Id of the closest cluster
4. Set Nearest_Dist[i]= *d(di, cj)*;
5. For each cluster *j* (1<=j<=k), recalculate the centroids;
6. **Repeat**
7. For each data-point *di*,

7.1 Compute its distance from the centroid of the present nearest cluster;

7.2 If this distance is less than or equal to the present nearest distance, the data-point stays in the cluster; Else

7.2.1 For every centroid *cj* (1<=j<=k), Compute the distance *d(di, cj)*;

Endfor;

7.2.2 Assign the data-point *di* to the cluster with the nearest centroid *cj;*

7.2.3 Set ClusterId[i]=j;

7.2.4 Set Nearest_Dist[i]= *d(di, cj)*;

Endfor;

8. For each cluster *j* (1<=j<=k), recalculate the centroids;

Until convergence (i.e., no more data-points cross the cluster boundaries).