Self-Organizing-Queue Based Clustering

Baohua Sun, Dapeng Wu

***Abstract*—In this paper, we consider the problem of clustering, given the similarity matrix of a set of data points or nodes; this problem is a.k.a. graph clustering. Spectral clustering techniques are typically used to solve this problem. The performance of the existing spectral clustering techniques is not satisfactory for many applications. To improve the performance, we take a bio-inspired approach to the graph clustering problem and enable fictitious queues with self-organizing capability to group similar nodes into the same cluster; we call the resulting scheme, Self-Organizing-Queue (SOQ) clustering scheme. Experimental results have demonstrated the superiority of our SOQ scheme over the existing spectral clustering techniques and K-means algorithm.**

***Index Terms*—Graph clustering, spectral clustering, K-means**

I. Introduction

In this paper, we consider the problem of graph clustering, i.e., given the similarity matrix of a set of nodes (or data points), partition the nodes into clusters. Spectral clustering techniques [1] are typically used to solve the graph clustering problem [2]. The performance of the existing spectral clustering techniques is not satisfactory for many applications. To improve the performance, we take a bio-inspired approach to the graph clustering problem. Our idea is to place all nodes into multiple fictitious queues, each of which corresponds to one cluster; then we enable these fictitious queues with self-organizing capability to group similar nodes into the same cluster; we call the resulting scheme, Self-Organizing-Queue (SOQ) clustering scheme. To show the working of our SOQ scheme, let us first look at an example of how humans do grouping. In Fig. 1, at the beginning of time slot $t = 1$, suppose students from two different classes are mixed up to form a line/row; assume that each student is affiliated with only one class; in Fig. 1, a circle represents a student from one class while '$\times$' represents a student from another class. The students are asked to form two groups, each of which only consists of students from the same class. Suppose in each time slot, only one person is allowed to move, and we call this person the Current Person. Suppose only the person in the middle of the line is allowed to move[1]; so the middle person is the Current Person. A strategy of grouping is that in each slot, the Current Person looks around and moves to the head of the line if he/she sees more of his/her classmates are on his/her left-hand side than that on his/her right-hand side; otherwise, he/she moves to the tail of the line. This procedure repeats until no change of grouping. Fig. 1 shows the grouping result at the beginning of each slot t and the move to be made during each slot t; at $t = 20$, the procedure stops and the two classes are separated. Our SOQ clustering scheme is based on this grouping strategy. In the remainder of the paper, we call such a line/row (of people/nodes) as a queue; a queue has a head and a tail.

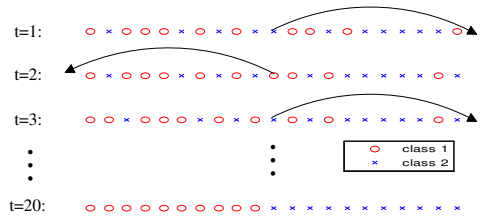


Fig. 1: An example of the working of Self Organizing Queue

II. Description of SOQ Clustering Scheme

Algorithm 1 shows the key steps in SOQ. In Algorithm 1, Current_Queue is a variable, pointing to a queue called 'Current Queue'; Current_Person is a variable, pointing to a member in a queue called 'Current Person'; Next_Queue is a variable, pointing to a queue called 'Next Queue'. Similarity matrix $\mathbf{W}$ has a dimension of $N \times N$ and each entry $W_{i,j}$ in $\mathbf{W}$ denotes the similarity measure between Node i and Node j. If the input is a dissimilarity matrix, one possible option is to convert the dissimilarity matrix into a similarity matrix by adding a negative sign to the dissimilarity matrix. Algorithm 1 outputs K queues/clusters, each with an index set of queue members.

Algorithm 1: SOQ.

Input: a set of N nodes and similarity matrix $\mathbf{W}$.

1) Initialization: divide the set of N nodes into K queues; assign a queue to Current_Queue; Flag=1.
2) While (Flag)
3) **WHO:** Choose *who* in Current_Queue as Current_Person.
4) **HOW:** (*How* to) select a queue as Next_Queue for Current_Person to join.
5) **WHERE:** (*Where* to) place Current_Person in Next_Queue.
6) Assign Next_Queue to Current_Queue.
7) **WHEN:** (*When* to) let Flag=0, i.e., stop the loop.
8) Endwhile

Output: the resulting K queues/clusters.

From Algorithm 1, it can be seen that the key features of SOQ are: 1) self organizing, i.e., each person/node has the ability to decide where it wants to join; 2) sequential process, i.e., each time only one queue is selected as Current_Queue[2];

Baohua Sun and Dapeng Wu are with Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611. Email of Baohua Sun: bsun@ufl.edu. Corresponding author: Prof. Dapeng Wu; email: wu@ece.ufl.edu.

[1]If there are an even number of students in the line, we let the odd-indexed middle student be the middle of the line.

[2]Allowing multiple queues to be selected as Current Queues causes non-convergence in all our experiments.

3) similarity matrix $\mathbf{W}$ can be asymmetric, and the entries in $\mathbf{W}$ can take any real value, including negative values. Note that none of the existing spectral clustering algorithms allows asymmetric similarity matrix and similarity matrix with negative entries.

There are many variations of SOQ, depending on how to implement Step 1, 3, 4, 5, and 7. Next, we show some examples including the default setting used in the basic version of SOQ.

In Step 1 (Initialization), the default setting is to divide the set of N nodes into K queues by *random selection*, and assign the queue with the most members to Current_Queue. Another way of obtaining initial K queues is to use a spectral clustering technique [1]; any spectral clustering technique is applicable here.

In Step 3 (WHO), we can choose any member of Current_Queue as Current_Person; the default setting is to choose the head of Current_Queue as Current_Person.

In Step 4 (HOW), Current Person i can use the following default criterion (called *Most Friends*) to choose Queue $\hat{k}$ as the Next Queue to join:

$$\hat{k} = \arg\max_{k} \frac{\sum_{j \in C_k} (W_{i,j} + W_{j,i})}{|C_k|} \tag{1}$$

where C_k is the set of indices of members in Queue k. Note that Current Person i itself is not counted in its Current_Queue in Eq. (1) since Current Person i is looking for a queue with most friends excluding itself.

In Step 5 (WHERE), we can place Current_Person at any position in Next_Queue; the default setting is to place Current_Person at the tail of Next_Queue.

In Step 7 (WHEN), the default criterion to stop the algorithm is that none of the queues has membership change.

Algorithm 1 has a limitation: when the Current Person joins the Next Queue, the Current Queue may become empty; then the number of clusters will be reduced to $K-1$ rather than the target value K. To address this, we propose CESOQ (one Cluster being Empty SOQ) as shown in Algorithm 2; specifically, we insert Step 5.1 between Step 5 and Step 6 in Algorithm 1. In Algorithm 2, the Crosstalk Density $\rho(C_s, C_t)$ between Queue s and Queue t is defined as:

$$\rho(C_s, C_t) = \frac{\sum_{j \in C_s} \sum_{i \in C_t} (W_{i,j} + W_{j,i})}{2 \times |C_s| \times |C_t|} \tag{2}$$

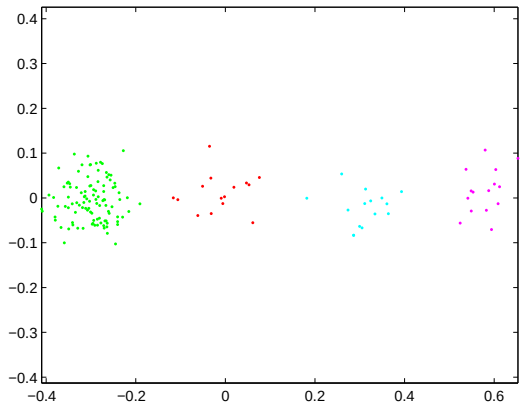
Algorithm 2: CESOQ.
Input and Step 1 through 5 are the same as Algorithm 1.
5.1) If (Current_Queue is empty)
  Use Algorithm 1 to partition each of the non-empty queues into two clusters (a pair of clusters).
  Find the pair of clusters with minimum crosstalk density.
  One of the two clusters with minimum crosstalk density replaces its original queue, and the other cluster replaces the empty queue.
 Endif
Step 6 through 8 and Output are the same as Algorithm 1.

Algorithm 2 has a limitation: within-cluster-talk of a cluster may be smaller than cross-talk between clusters. To address this, we propose MSSOQ (Merge Split SOQ) as shown in Algorithm 3; specifically, we insert Step 1.1 and 1.2 between Step 1 and Step 2, and append new Steps 9–15 to Step 8 in Algorithm 2.

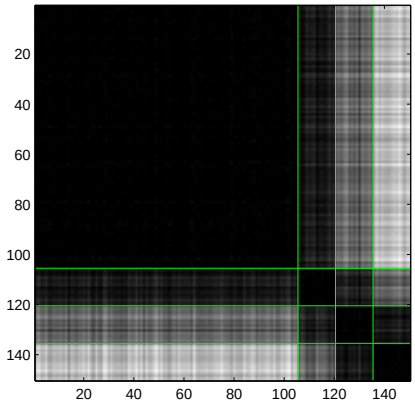
Algorithm 3: MSSOQ.
Input and Step 1 are the same as Algorithm 2.
1.1) Flag2=1.
1.2) While (Flag2)
Step 2 through Step 8 are the same as Algorithm 2.
9) Use (2) to calculate $\rho(C_s, C_t)$ for $s \neq t$, $s \in \{1, \cdots, K\}$, and $t \in \{1, \cdots, K\}$.
10) Compute $\rho_{between} = \max_{\{s,t:s \neq t\}} \rho(C_s, C_t)$, and $\{s_1^*, t_1^*\} = \arg\max_{\{s,t:s \neq t\}} \rho(C_s, C_t)$.
11) Use Algorithm 1 to split each queue C_s into two new queues, denoted by $C_{s,1}$ and $C_{s,2}$.
12) Use (2) to calculate $\rho(C_{s,1}, C_{s,2})$ for $s \in \{1, \cdots, K\}$.
13) Compute $\rho_{within} = \min_{s \in \{1, \cdots, K\}} \rho(C_{s,1}, C_{s,2})$, and $s_2^* = \arg\min_{s \in \{1, \cdots, K\}} \rho(C_{s,1}, C_{s,2})$.
14) If ($\rho_{within} < \rho_{between}$)
  If ($s_2^* == s_1^*$)
    If ($\rho(C_{s_2^*,1}, C_{t_1^*}) > \rho(C_{s_2^*,2}, C_{t_1^*})$)
      Merge $C_{s_2^*,1}$ and $C_{t_1^*}$ into a new Queue t_1^*, and call $C_{s_2^*,2}$ as a new Queue s_2^*.
    Else
      Merge $C_{s_2^*,2}$ and $C_{t_1^*}$ into a new Queue t_1^*, and call $C_{s_2^*,1}$ as a new Queue s_2^*.
    Endif
  Else
    If ($s_2^* == t_1^*$)
     If ($\rho(C_{s_2^*,1}, C_{s_1^*}) > \rho(C_{s_2^*,2}, C_{s_1^*})$)
      Merge $C_{s_2^*,1}$ and $C_{s_1^*}$ into a new Queue s_1^*, and call $C_{s_2^*,2}$ as a new Queue s_2^*.
     Else
      Merge $C_{s_2^*,2}$ and $C_{s_1^*}$ into a new Queue s_1^*, and call $C_{s_2^*,1}$ as a new Queue s_2^*.
     Endif
    Else
      Merge $C_{s_1^*}$ and $C_{t_1^*}$ into a new Queue s_1^*, call $C_{s_2^*,1}$ as a new Queue s_2^*, and call $C_{s_2^*,2}$ as a new Queue t_1^*.
    Endif
  Endif
 Else Flag2=0.
 Endif
15) Endwhile
Output is the same as Algorithm 2.

III. Experimental Results

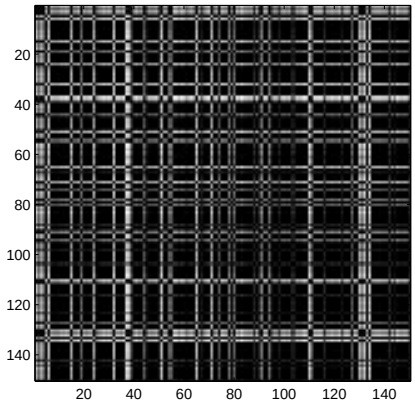
In this section, we test the performance of MSSOQ with synthetic data and real-world data, and compare it with the existing main-stream clustering algorithms, i.e., K-means, unnormalized spectral clustering (SC for short) [1], the normalized spectral clustering algorithm by Ng, Jordan, Weiss (NJW for short) [3], and ncut [4].



(a) Positions of the sample points



(b) Similarity matrix before random permutation



(c) Similarity matrix after random permutation

Fig. 2: Synthetic data set and its similarity matrices

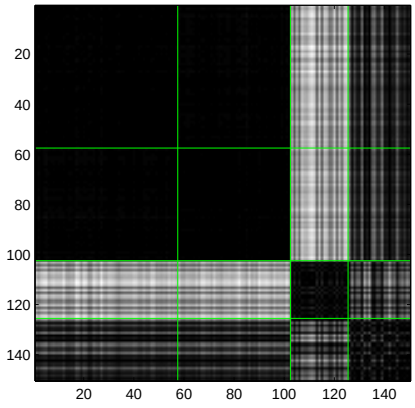
A. Experiments with Synthetic Data

1) Description of Synthetic Data Set: The synthetic data set consists of 2-D Gaussian-distributed sample points. To simulate four clusters, we use four 2-D Gaussian distributions with the same standard deviation of 0.05 and mean (-0.3, 0), (0, 0), (0.3, 0), and (0.6, 0), respectively, and each 2-D Gaussian distribution corresponds to one cluster; the two dimensions of the 2-D Gaussian are independent and identically distributed. The number of samples for the four clusters are 105, 15, 15, and 15, respectively, and the total number of points N is 150. Fig. 2(a) shows the 2-D positions of the 150 sample points. Note that in this synthetic data set, the size of the first cluster is much larger than other clusters; this is challenging for many clustering algorithms since these clustering algorithms only perform well when all the clusters have similar sizes.

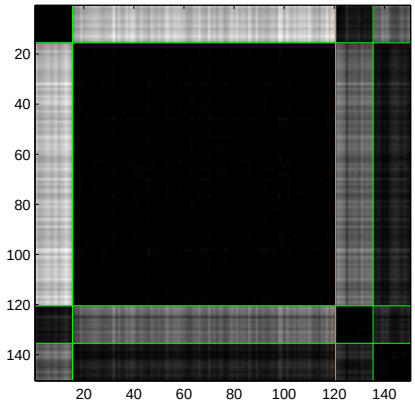
For spectral clustering algorithms, a similarity matrix is needed. Denote the generated 2-D points by $\mathbf{p}_1, \mathbf{p}_2, ..., \mathbf{p}_N$, with $\mathbf{p}_n = (x_n, y_n)$ for $1 \leq n \leq N$. We generate the similarity measure $W_{i,j}$ between any two points $\mathbf{p}_i$ and $\mathbf{p}_j$ by $W_{i,j} = \exp(-||\mathbf{p}_i - \mathbf{p}_j||^2/(2\sigma^2))$, where $\sigma = 1$ in this set of experiments. In this way, we obtain a similarity matrix $\mathbf{W}$. In order to make a fair performance comparison, we randomly permute $\mathbf{W}$. Fig. 2(b) shows the similarity matrix before random permutation; the darker of the entry at Row i and Column j, the larger similarity measure between point $\mathbf{p}_i$ and point $\mathbf{p}_j$. Fig. 2(c) shows the similarity matrix (of the sample points) with randomly permuted entries.

2) Clustering Performance: The input of SC, NJW, ncut, and MSSOQ is the similarity matrix (of the sample points) with randomly permuted entries. Fig. 3 shows the clustering results of ncut and MSSOQ. The results obtained by SC, NJW are similar to ncut. Each similarity matrix shown in Fig. 3 is permuted according to the clustering results. E.g., Fig. 3(b) shows that $\mathbf{p}_1$ to $\mathbf{p}_{15}$ belong to Cluster 1, $\mathbf{p}_{16}$ to $\mathbf{p}_{120}$ belong to Cluster 2, $\mathbf{p}_{121}$ to $\mathbf{p}_{135}$ belong to Cluster 3, and $\mathbf{p}_{136}$ to $\mathbf{p}_{150}$ belong to Cluster 4; after re-mapping to the original order given in Fig. 2(a), we find that all the 150 points are correctly grouped into the corresponding clusters; hence, MSSOQ achieves zero clustering error. We run the algorithms 20 times, each with different randomly permuted input; and we obtain 20 clustering results; by comparing to the ground truth in Fig. 2(a), we obtain the error rate for each experiment. For the 20 experiments, we calculate the mean clustering error rate and 95% confidence interval, which are listed in the second column of Table I, where μ_{error} denotes the mean clustering error rate and $\mu_{error} \pm \kappa$ denotes upper/lower bound of the confidence interval, respectively. Table I demonstrates that MSSOQ significantly outperforms K-means, SC, NJW, and ncut for this synthetic data set.

3) Time Complexity: We run the algorithms 20 times, each with different randomly permuted input; and obtain the run-time of each experiment. For the 20 experiments, we calculate the mean time complexity and 95% confidence interval, which are listed in the second column of Table II, where μ_t denotes the mean run-time and $\mu_t \pm \kappa$ denotes upper/lower bound of the confidence interval, respectively. Table II shows that MSSOQ has a higher time complexity than K-means, SC, NJW, and



(a) ncut



(b) MSSOQ

Fig. 3: Clustering results of synthetic data

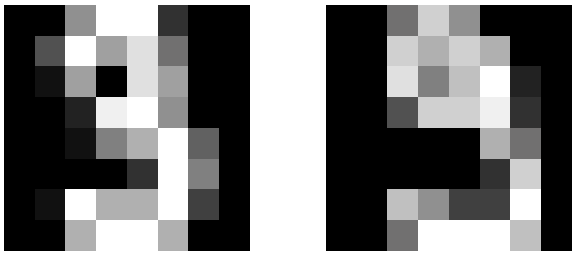
TABLE I: Error rate

$\mu_{error} \pm \kappa$	Synthetic Data	Handwritten Digits
K-means	0.3090 ± 0.0865	0.2661 ± 0.0349
SC	0.3483 ± 0.0620	0.3704 ± 0.0219
NJW	0.3867 ± 0	0.3231 ± 0.0017
ncut	0.5020 ± 0.0474	0.3228 ± 0.0204
MSSOQ	0 ± 0	0.1603 ± 0.0192

TABLE II: Time complexity (in unit of seconds)

$\mu_t \pm \kappa$	Synthetic Data	Handwritten Digits
K-means	0.0139 ± 0.0141	0.4085 ± 0.2298
SC	0.0489 ± 0.0235	1.2401 ± 0.3889
NJW	0.3438 ± 0.0038	13.7281 ± 0.2577
ncut	0.0774 ± 0.0039	8.9902 ± 0.1694
MSSOQ	0.6537 ± 0.0484	32.4512 ± 4.0747

ncut for this synthetic data set. Though MSSOQ consumes more time than the existing ones, the time complexity is still acceptable.



(a) digit 3 (b) digit 9

Fig. 4: Digit 9 in (b) is very similar to digit 3 in (a), and is clustered into the cluster of digit 3.

B. Experiments with Real-World Data

1) Description of Real-World Data Set: The real-world data set used in this set of experiments consists of images of handwritten digits, which is described in [5] and is downloadable at [6]. The 10 digits data set is used in our experiment. There are 10 clusters in the data set, with 100 members in each cluster. The dimension of the similarity matrix is 1000 by 1000.

2) Clustering Performance: We run the algorithms 20 times, each with different randomly permuted input; and we obtain 20 clustering results; by comparing to the ground truth, we obtain the error rate for each experiment. For the 20 experiments, we calculate the mean clustering error rate and 95% confidence interval, which are listed in the third column of Table I. Table I demonstrates that MSSOQ significantly outperforms K-means, SC, NJW, and ncut for this set of handwritten digits.

To identify what causes incorrect clustering in MSSOQ, we find that the handwritten digit 9 in Fig. 4(b) is very similar to the handwritten digit 3 in Fig. 4(a), and hence this sample of digit 9 is clustered into the cluster of digit 3. We also find that all the clustering algorithms under our study mis-classified this sample. We examine other mis-classified instances and find out that most of the instances mis-classified by MSSOQ are also mis-classified by K-means, SC, NJW, and ncut.

3) Time Complexity: We run the algorithms 20 times, each with different randomly permuted input; and obtain the runtime of each experiment. For the 20 experiments, we calculate the mean time complexity and 95% confidence interval, which are listed in the third column of Table II. Though MSSOQ consumes more time than the existing ones, the time complexity is still acceptable.

REFERENCES

[1] U. Von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, no. 4, pp. 395–416, 2007.

[2] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[3] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm," *Advances in neural information processing systems*, vol. 2, pp. 849–856, 2002.

[4] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[5] D. Verma and M. Meila, "A comparison of spectral clustering algorithms," *University of Washington, Tech. Rep. UW-CSE-03-05-01*, 2003.

[6] ——, "digit1000.mat," 2003. [Online]. Available: http://www.stat.washington.edu/spectral/datasets.html