HINGCN: Heterogeneous information network revisited with GCN

Danhao Ding*, Xiang Li*, Nikos Mamoulis†, Ben Kao*
*The University of Hong Kong, Pokfulam Road, Hong Kong
†University of Ioannina, Ioannina, Greece
*{dhding2, xli2, kao}@cs.hku.hk †nikos@uoi.gr

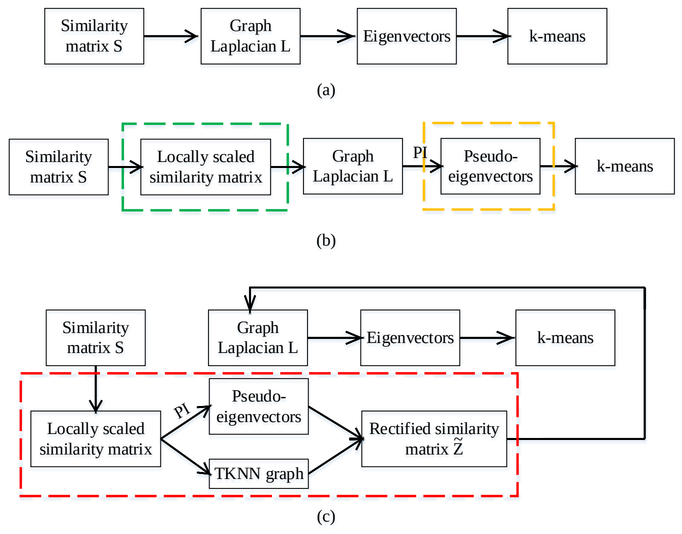


Figure 1: The key steps of (a) basic spectral clustering; (b) with local scaling and PI; (c) ROSC

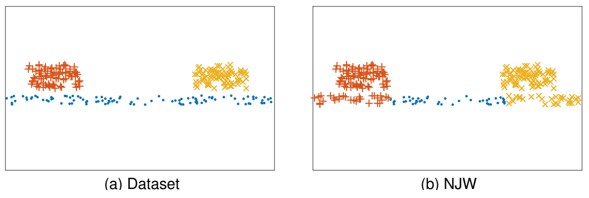


Figure 2: (a) A multi-scale dataset, (b) clustering by NJW

ABSTRACT

GCN on HIN.

KEYWORDS

Semi-supervised classification; graph convolution; heterogeneous information network

ACM Reference Format:
Danhao Ding*, Xiang Li*, Nikos Mamoulis†, Ben Kao*. 2018. HINGCN: Heterogeneous information network revisited with GCN. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3178876.3185993

1 INTRODUCTION

Dan:TODO

Despite the successes of spectral clustering, previous works have pointed out that spectral methods can be adversely affected by the presence of *multi-scale data* [? ?], which is defined as data whose object clusters are of various sizes and densities. As an illustrative example, Figure 2(a) shows a dataset of three clusters: two dense rectangular clusters on top of a narrow sparse stripe cluster. Figure 2(b) shows the result of applying the standard spectral clustering method NJW to the dataset. We see that the stripe cluster is segmented into three parts, two of which are incorrectly merged with the rectangular clusters. The objective of this paper is to address the multi-scale data issue in spectral clustering. In particular, we review existing methods for handling multi-scale data, provide insight into how the issue can be addressed, and put forward our algorithm called ROSC which outperforms existing methods in clustering multi-scale data.

There are two general approaches to address the multi-scale data problem: one on scaling the similarity matrix S and another on applying the power iteration technique to obtain pseudo-eigenvectors that contain rich cluster separation information.

Recall that spectral clustering methods model data objects as a graph and perform clustering by graph partitioning. The similarity matrix S should therefore capture objects' local neighborhood information. A common choice of such a similarity function is the Gaussian kernel: $S_{ij} = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{2\sigma^2}\right)$, where $\boldsymbol{x}$ (boldface) denotes a feature vector of an object x, and σ is a global scaling parameter. A major difficulty in using the Gaussian function to cluster multi-scale data lies in the choice of σ. If σ is set small, then S_{ij} will become small. Objects in a sparse cluster (which are relatively distant among themselves compared with objects in a dense cluster) will likely be judged as dissimilar leading to partitioning of the cluster. On the other hand, if σ is set large, S_{ij} will be large. Hence, nearby dense clusters could be judged similar to each other and are inadvertently merged.

To tackle this problem, the ZP method [?] applies *local scaling* and modifies the Gaussian similarity to $S_{ij} = \exp\left(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||^2}{\sigma_i \sigma_j}\right)$. Here, σ_i (and likewise for σ_j) is a local scaling parameter for object x_i. It is defined as the distance between x_i and its l-th nearest neighbor for some empirically determined value l. If x_i is located in a sparse cluster, then σ_i will be large. This boosts the similarity of x_i and its neighboring objects and thus avoids the splitting of sparse clusters. Also, if x_i is located in a dense cluster, σ_i will be small. Objects will have to be very close to be considered neighbors. This avoids the merging of nearby dense clusters.

Previous studies [? ?] have suggested that employing more eigenvectors beyond the k smallest ones can help capture more cluster separation information and thus improve spectral clustering in handling multi-scale data. A *power iteration* (PI) method [?] was put forward as an efficient method for computing the dominant eigenvector of a matrix. It is observed in [?] that one can *truncate* the iteration process to obtain an intermediate pseudo-eigenvector $\boldsymbol{v}_t$. It is shown that $\boldsymbol{v}_t$ represents a weighted linear combination of all the eigenvectors and is thus a very effective replacement of the k smallest eigenvectors typically used in a standard spectral clustering process. Figure 1(b) shows how the local-scaling method (green box) and the power iteration method (yellow box) are integrated into the basic spectral clustering method.

In this paper we take a different approach to tackle the multi-scale data problem. The core idea is to construct an $n \times n$ coefficient matrix Z such that the entry Z_{ij}[1] reflects how well an object x_i characterizes another object x_j. Our objective is to derive such a Z with "grouping effect": if two objects x_i and x_j are highly correlated (and thus should be put into the same cluster), then their corresponding coefficient vectors $\boldsymbol{z}_i$ and $\boldsymbol{z}_j$ given in Z are similar. The interesting question we address is how to find such a Z.

The main feature of our algorithm ROSC is illustrated by the red box shown in Figure 1(c). Instead of using PI to obtain low dimensional embeddings of the objects as input to k-means (yellow box in Figure 1(b)), ROSC uses the embeddings to construct the matrix Z (upper path in the red box). We note that two objects that belong to the same cluster could be located at distant far ends of a cluster, their high correlation is therefore not expressed properly by a distance-based similarity matrix S. To capture the high correlation between distant objects in the same cluster, we propose to use a transitive K nearest neighbor (TKNN) graph (lower path in the red box). Two objects x_i and x_j are connected by an edge in the TKNN graph if there is a sequence of objects $< x_i, \ldots, x_j >$ such that adjacent objects in the sequence are mutual K nearest neighbors of each other. We use the TKNN graph to regularize the matrix Z so that it possesses the desired grouping effect. The matrix Z is then fed to the pipeline of spectral clustering, taking the role of S.

Our main contributions are:

• We proposed a heterogeneous graph convolution algorithm that is capable of capturing edge information.

• Dan:don't know

• We conduct extensive experiments to evaluate the performance of HINGCN against 9 other classification methods. Our results show that HINGCN performs very well against the competitors. In particular, it is very robust in that it consistently performs well over all the datasets tested. Also, it outperforms others by wide margins for datasets that are highly multi-scale.

The rest of the paper is organized as follows. Section 2 mentions related works on heterogeneous graph neural networks, graph embedding and described several semi-supervised classification algorithms. Section 4 presents the HINGCN algorithm. Section 5 describes the experiments and presents experimental results. Finally, Section 6 concludes the paper.

[1]Given a matrix M, we use a pair of subscripts to specify an entry of M.

2 RELATED WORK

2.1 Heterogeneous graph neural networks

Dan:TODO

2.2 Heterogeneous graph embedding

Dan:TODO Although the two bottlenecks have attracted great attention, few methods aim to solve both. In fact, power iteration based methods can not only reduce the high time complexity, but also utilize all the relevant eigenvectors rather than just the top-k eigenvectors, which further leads to a good performance in data containing multi-scale clusters or much noise. Recently, without any assumptions, FUSE [?] was proposed which has been proved both effective and efficient even on data containing multi-scale clusters. Its success arises from the fusion of all the cluster-separation information, but noise is also encoded. Further, based on ICA, it might be trapped into local optimum and run slowly. These shortcomings motivate us to further explore spectral clustering.

3 PRELIMINARY

Definition 1. **Heterogeneous graphs**. Dan:TODO □

Example 1. *Dan:TODO*

Definition 2. **Metapath**. Dan:TODO

Example 2. *In Fig.* **??**, *Dan:TODO*

4 ALGORITHM

In this section we describe our HINGCN algorithm. Figure 1(c) shows a flow diagram of HINGCN. Dan:TODO:signposting

Pretrained Node Representations as Edge Features

Dan:TODO

Aggregating node embedding

Dan:TODO

$$X = XZ. \tag{1}$$

Aggregating edge embedding

Dan:TODO

$$X = XZ. \tag{2}$$

Jumping connections for adaptive depth

Dan:TODO

$$X = XZ. \tag{3}$$

Aggregation across different meta-path semantics

Dan:TODO

$$X = XZ. \tag{4}$$

Dan:need some intuition/justification shit Given a node i, and its set of embeddings under different semantics, a metapath aggregator is a function γ in the form of $y_i = \gamma_\Theta(\{x_{i,1}, x_{i,2}, ..x_{i,P}\})$, where $x_{i,p}$ is the embedding of node i under metapath p, and y_i is the aggregated embedding of node i in the HIN. Θ is the learnable paremeters of the aggregator. Here we propose several metapath aggregators.

• **Attention**

• **Element-wise Gated Sum** Inspired by forget gate mechanisms in recurrent networks (RNN) Dan:TODO:cite, we compute a soft gate between 0 (low importance) and 1 (high importance) to represent different importance to each embedding entry. We make the element-wise gated sum layer as follows:

$$o_{\Phi_p} = \sigma(W_o^T Z_{\Phi_p}), \qquad \tilde{Z}_{\Phi_p} = \tanh(W_z^T Z_{\Phi_p})$$

and

$$Z'_{\Phi_p} = o_{\Phi_p} \odot \tilde{Z}_{\Phi_p} \tag{5}$$

The gated unit o_{Φ_p} with sigmoid output (ranged (0,1)) is used to control which elements are to be aggregated in the output. • **Other aggregators** In preliminary experiments, CONCAT, MAX-POOLING and MEAN aggregators did not generate a better result, and we omit results of these aggregators in experiment section.

HINGCN: Semi-supervised classification on Heterogeneous Information Network via Graph Convolution Networks

After meta-path aggregation layer, the embeddings are fed to a 2-layer MLP for final output. In semi-supervised classification tasks, we minimize the Cross-Entropy loss and the model generates final prediction based on the largest output value. Finally, HINGCN is summarized in Algorithm 1.

To this end, we summarize ROSC as follows. ROSC first computes the TKNN graph $\mathcal{G}$ and the associated weight matrix $\mathcal{W}_K$. Then it applies power iteration to generate p pseudo-eigenvectors that form a $p \times n$ matrix X. After whitening and normalization on X, the coefficient matrix Z^* can be calculated by solving Eq. **??**, which further leads to the construction of a rectified similarity matrix $\tilde{Z}$. Since $\tilde{Z}$ has the grouping effect, the standard spectral clustering methods (e.g., NCuts) will be finally applied to derive more robust clustering results. The time complexity of ROSC will be no more than the standard spectral clustering methods, which is $O(n^3)$ in general. In the future, we will attempt to improve it.

5 EXPERIMENT

We conducted extensive experiments to evaluate the performance of HINGCN. This section summarizes our results. We compare the various methods using three popular measures, namely, *accuracy*, *micro-F1*, and *macro-F1*. These measures evaluate clustering quality and their values range from 0 to 1, with a larger value indicating a better clustering quality.

5.1 Datasets

A summary of statistics of the datasets rae shown in table **??**.

Algorithm 1 HINGCN

```
Input: $S$, $k$.
Output: $C = \{C_1, \dots, C_k\}$
1: Compute the TKNN graph and the reachability matrix $\mathcal{W}$
2: Calculate $W = D^{-1}S$, where $D_{ii} = \sum_j S_{ij}$
3: Apply PI on $W$ and generate $p$ pseudo-eigenvectors $\{\boldsymbol{v}_r\}_{r=1}^{p}$
4: $X = \{\boldsymbol{v}_1^T; \boldsymbol{v}_2^T; \dots; \boldsymbol{v}_p^T\}$; $X$ = whiten($X$)
5: Normalize each column vector $\boldsymbol{x}$ of $X$ such that $\boldsymbol{x}^T\boldsymbol{x} = 1$
6: Calculate the coefficient matrix $Z^*$ by Eq. ??
7: Construct $\tilde{Z} = (|Z^*| + |(Z^*)^T|)/2$
8: Run standard spectral clustering methods, e.g., NCuts, with $\tilde{Z}$ as the
   similarity matrix to obtain clusters $C = \{C_r\}_{r=1}^{k}$
9: return $C = \{C_1, \dots, C_k\}$
```

• **DBLP**: We extract a subset of DBLP with contains 4057 authors (A), 14328 papers (P) and 20 conferences (C). 8789 terms of papers are aggregated to each author as feature, after tf-idf transformation. Here we select meta-paths $\{APA, APAPA, APCPA\}$ for experiments.
• **Yelp**: A standard spectral clustering method with symmetric normalization.
• **Freebase**: A standard spectral clustering method with symmetric normalization.

5.2 Algorithms for comparison

We evaluate HINGCN and 9 other methods. These methods are grouped into the following four categories.
• **NJW**: A standard spectral clustering method with symmetric normalization.
• **NCuts**: A standard spectral clustering method with divisive normalization.
• **ZP**: A self-tuning spectral clustering method for multi-scale clusters. It uses eigenvector rotation to estimate the number of clusters. To make a fair comparison, we directly set k as in other methods.
• **PIC**: A power iteration based method which generates only one pseudo-eigenvector.
• **PIC-k**: A power iteration based method which generates $\lceil log(k) \rceil$ pseudo-eigenvectors.
• **DPIC**: A power iteration based method which employs Schur complement deflation to generate mutually orthogonal pseudo-eigenvectors.
• **DPIE**: A power iteration based method which generates a set of diverse pseudo-eigenvectors.
• **FUSE**: A full spectral clustering method based on power iteration and ICA.

[Experiment settings] The parameters of all the methods are set according to their original papers. For all the datasets, we use Euclidean distance of objects' attributes to derive S, which are locally scaled based on ZP's local scaling procedure. For ROSC, we set $\alpha_1 = 1.0$, $\alpha_2 = 0.01$, and we set $K = 4$ in constructing TKNN graphs. All methods adopt k-means as the last step to return clustering results. For this step, we run k-means 100 times with random starting centroids and the most frequent cluster assignment is used [?]. For ROSC, we generate k pseudo-eigenvectors with random starting vectors as is done in [?]. For each method and dataset, we run the experiment 50 times and report average results.

5.3 Performance results

We first use two synthetic datasets to visually illustrate the performance of the various methods. After that, we use 7 real datasets to perform an in-depth analysis of the algorithms.

[Synthetic datasets] The synthetic datasets are designed to represent very difficult cases of clustering, with a highlight on multi-scale data. Figure 3(a) shows the synthetic dataset Syn1, which consists of three clusters of different sizes and densities. Specifically, there is a medium-sized sparse rectangular cluster (blue) sandwiched by a small dense circular cluster (magenta) and a large dense rectangular one (yellow). These clusters are physically very close to each other. In particular, the distance between two objects of different clusters could be smaller than the distance of two objects that belong to the same cluster.

We apply all 10 methods on Syn1. Due to space limitations, for each category of methods, we only show the best performing one. They are, namely, NJW, PIC-*k*, FUSE, and ROSC. Their clustering results are shown in Figures 3(b)-(e), respectively. From the figures, we see that both NJW and PIC-*k* can identify the small circular magenta cluster. However, the blue and the yellow rectangular clusters are chopped into halves, and these halves are incorrectly merged. FUSE fails for this dataset as well. In particular, about 1/3 of the yellow cluster is merged with the blue cluster, while about half of the blue cluster is merged with the magenta cluster. Among the four methods, ROSC is the only one that can recover the yellow cluster. This indicates the effectiveness of the TKNN graph in correlating objects that are at the far ends of the big elongated cluster. Moreover, although ROSC does not recover the complete blue cluster, a majority fraction of the blue objects are clustered by ROSC as a single group. In contrast, for the other three methods, all blue objects are either merged with those of the magenta cluster or those of the yellow cluster.

Figure 4(a) shows another synthetic dataset, Syn2, which consists of a sparse ring cluster (green) and two dense circular clusters (red and blue). The best performing methods of the 4 categories are NJW, PIC-*k*, ZP, and ROSC. Their clustering results are shown in Figures 4(b)-(e), respectively. From the figures, we see that the dataset is a very difficult case for existing methods. For example, with NJW and ZP, the green ring cluster is partitioned into three segments, two of which are merged incorrectly with the circular clusters. A similar situation is also seen for PIC-*k*. In contrast, ROSC is the only method that can recover almost the entire green cluster. It is also able to correctly identify the two circular clusters except for a small number of objects on the green ring.

Tables 1 and 2 show the purity, AMI and RI scores of all 10 methods for the datasets Syn1 and Syn2, respectively. From the tables, we see that the scores of ROSC are all much larger than those of the other methods. It thus significantly outperforms the others for these two difficult cases. The two tables also show the scores of ROSC-R, which is ROSC without using the TKNN graph or the reachability matrix for regularization. Considering the wide gaps between the scores of ROSC and ROSC-R, we see a strong positive effect of the reachability regularization. The use of TKNN graph and reachability has a significant effect in the two synthetic datasets because both datasets consist of large elongated clusters (e.g., the green ring in syn2). Objects in these clusters are at far distances from each other and their correlations are effectively captured by the reachability matrix that ROSC employs.

We further investigate how the various methods deal with multi-scale data by varying the sizes and densities of some clusters in the synthetic datasets. Here, we show some representative results. Specifically, we consider the middle sparse blue cluster in Syn1 (Figure 3(a)) and make two changes: (1) increase its density while keeping its size unchanged, and (2) increase its size while maintaining its density unchanged. We use Δd to denote the density change (e.g., Δd = 100% means that the density of the cluster is doubled). We change the size of the cluster by changing its length (enlarging the cluster sideway), while the height is kept unchanged. We use Δs to denote the size change (e.g., Δs = 50% means that the cluster is 1.5 times wider than the one shown in Figure 3(a)). We make similar changes to Syn2 by modifying the density and size of the ring cluster. Specifically, we gradually reduce the size of the ring from a whole ring (◯, Δs = 0%) to a lower half ring (◡, Δs = -50%).

We show the performance scores of the 10 methods as we apply the changes in density and size to the clusters in Syn1 (Figures 5 and 6) and Syn2 (Figures 7 and 8). From the figures, we see that ROSC gives the best and the most stable performance among all the methods over the whole spectrum of test cases. The performance gaps between ROSC and other competitors are also sizable. This shows that ROSC is very robust in dealing with multi-scale data of various sizes and densities.

[Real datasets] We further evaluate the methods using 7 real datasets. They are: *MNIST0127* (hand-written digit images), *COIL20* (images), *Yale_5class* (facial images), *isolet_5class* (speech, UCI repository), *seg_7class* (images, UCI repository), *Yeast_4class* (biological data, UCI repository), *glass* (UCI repository). Table 3 summarizes these datasets. For each dataset, we show the number of objects, the number of dimensions, and the number of clusters. Also, to show whether a dataset is multi-scale or not, we measure the *size* and *density* of each gold-standard cluster in each dataset. Specifically, for each cluster, we find the largest distance of any object-pair of the cluster. This distance is taken as the *diameter* of cluster, reflecting how big in size the cluster is. Moreover, for each cluster, we find the average distance, ρ, of all object-pairs of the cluster. Then, we use $1/\rho$ as a measure of density. These cluster sizes and densities are shown in bar graphs in Table 3. The sizes (densities) shown are all normalized by the size (density) of the biggest (densest) cluster of the corresponding dataset to the range [0, 1]. Intuitively, the more variations in the bars of a graph indicate the more multi-scale the corresponding dataset is.

From Table 3, we see the dataset *COIL20* is highly multi-scale. For example, cluster 2 (the largest cluster) is 5.6 times larger than cluster 15 (the smallest cluster). However, cluster 15 is 5.9 times denser than cluster 2. The dataset *glass* is moderately multi-scale, and *Yale_5class* is relatively uniform. The dataset *seg_7class* is interesting because it has one very big and sparse cluster (cluster 3); the other 6 clusters are quite uniform.

Tables 4, 5 and 6 show the purity, AMI, and RI scores of the 10 methods when they are applied to the 7 real datasets. Each row in the table corresponds to a (measure, dataset) combination — or a contest among the 10 methods. There are thus 21 (3 measures × 7 datasets) contests. For each contest, the winner's score is shown in bold type. For ROSC, its ranking in each contest is given in the

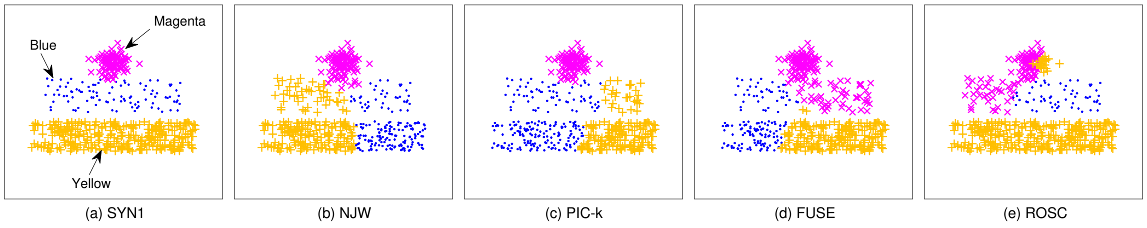


Figure 3: Clustering results on Syn1

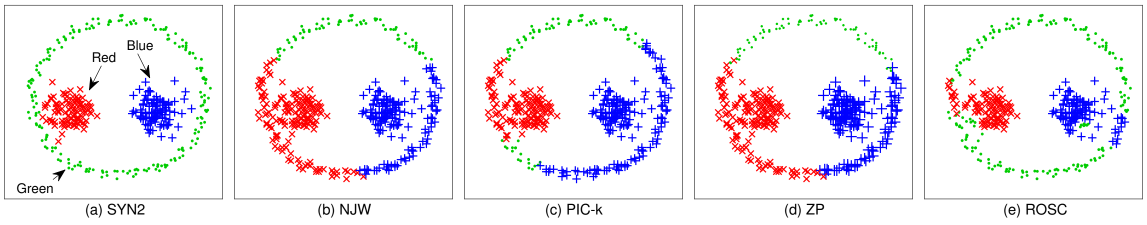


Figure 4: Clustering results on Syn2

Measure	NJW	NCuts	PIC	PIC-k	DPIC	DPIE	ZP	FUSE	ROSC-R	ROSC
Purity	0.8000	0.8000	0.7262	0.7772	0.6663	0.6348	0.8000	0.7688	0.7594	**0.8538**
AMI	0.4338	0.4256	0.3941	0.4686	0.2502	0.1381	0.4232	0.4873	0.4331	**0.6255**
RI	0.6811	0.6817	0.6513	0.6901	0.5786	0.4707	0.6812	0.6837	0.6762	**0.8354**

Table 1: Purity, AMI, and RI scores of methods for dataset Syn1

Measure	NJW	NCuts	PIC	PIC-k	DPIC	DPIE	ZP	FUSE	ROSC-R	ROSC
Purity	0.6775	0.6775	0.6541	0.6849	0.5196	0.5171	0.6875	0.6773	0.7002	**0.8359**
AMI	0.4681	0.4679	0.4157	0.4740	0.2136	0.1320	0.4746	0.4554	0.4602	**0.6178**
RI	0.6725	0.6724	0.6477	0.6789	0.4866	0.4392	0.6780	0.6683	0.6909	**0.8056**

Table 2: Purity, AMI, and RI scores of methods for dataset Syn2

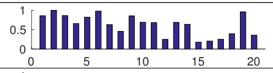
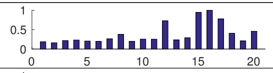
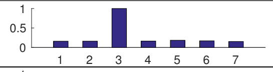
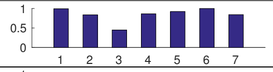
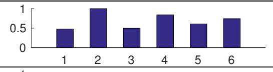
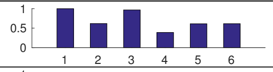
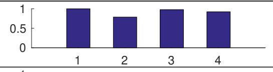
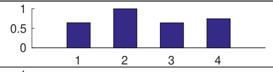
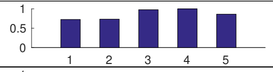
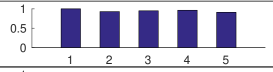
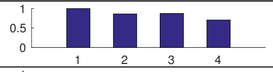
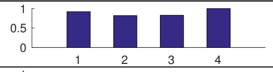
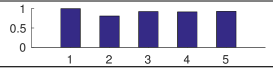
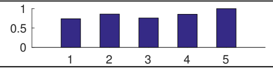
Dataset	#objects	#dimensions	#clusters	size	density
COIL20	1440	1024	20		
seg_7class	210	19	7		
glass	214	9	6		
MNIST0127	1666	784	4		
isolet_5class	300	617	5		
Yeast_4class	1299	8	4		
Yale_5class	55	1024	5		

Table 3: Statistics of 7 real datasets

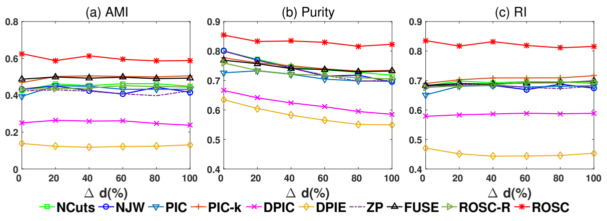


Figure 5: Results vs. varying blue cluster's density in SYN1

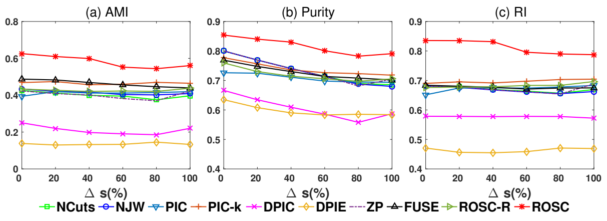


Figure 6: Results vs. varying blue cluster's size in SYN1

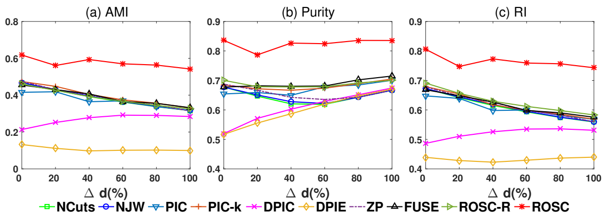


Figure 7: Results vs. varying green cluster's density in SYN2

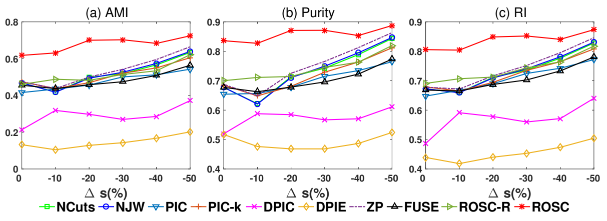


Figure 8: Results vs. varying green cluster's size in SYN2

bracket next to its score. The performance of a method can be judged by the column under the method spanning the three tables. From the tables, we make several observations:

(O1): Among the 21 contests, standard spectral clustering methods (NJW, NCuts) win only 2 times (AMI-*glass*-NJW and AMI-*isolet_5CLASS*-NJW). PI-based methods (PIC, PIC-*k*, DPIC, DPIE) also win only 2 times (purity-*isolet_5CLASS*-DPIE and RI-*isolet_5CLASS*-DPIE). These two categories of methods, which are not designed to specifically handle multi-scale data, are generally not outstanding.

(O2): The multi-scale-data-oriented methods (ZP, FUSE) win 4 times. In fact, if we take away ROSC, then (ZP, FUSE) win 14 times of the 21 contests. They are thus fairly effective in clustering multi-scale data. However, with a head-to-head comparison between ZP and FUSE, we see that among the 21 contests, ZP wins 9 times while FUSE wins 12 times. In some cases, there are big gaps in their performance scores. For example, for the contest (AMI-*COIL20*), ZP beats FUSE 0.5702 to 0.4448; for (AMI-*Yale_5CLASS*), FUSE outscores ZP 0.3495 to 0.2788. There is also a case (AMI-*isolet_5CLASS*) in which FUSE (0.6516) loses significantly to the basic NJW method (0.7595). We thus see that although ZP and FUSE generally perform well for multi-scale data, they are not very robust in providing consistently good performances across the datasets.

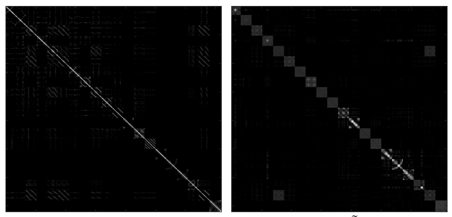


Figure 9: A visual comparison of S and $\tilde{Z}$ for *COIL20*

(O3): ROSC is winner 12 times and is first runner-up 6 times. Moreover, for those cases that ROSC does not win, its score is very close to that of the winner. For example, ROSC ranks 4th in (RI-*isolet_5CLASS*). Its score (0.9001), however, is quite close to that of the winner DPIE (0.9123). We thus see that ROSC is generally the best performer among all the 10 methods and it is very robust across the 7 datasets tested.

(O4): For *COIL20*, ROSC outperforms other methods by very wide margins. For example, in terms of purity, ROSC scores 0.9398, which is much better than NJW (0.4115), DPIE (0.3496), and FUSE (0.4177). From Table 3, we see that *COIL20* is highly multi-scale. The results thus highlight the ability of ROSC in clustering multi-scale data. From the cluster size distribution, we see that there are clusters in *COIL20* that are much bigger than others. Objects in these big clusters can thus be very distant from each other and hence their feature similarities are small. Their correlations are alternatively captured by ROSC using the TKNN graph and the derived reachability matrix. To see the effectiveness of this approach, we compare the scores of ROSC against those of ROSC-R, which does not use the reachability matrix to regularize Z. From Tables 4, 5, 6, we see big differences in the scores of ROSC and ROSC-R for the rows of *COIL20*. This verifies the importance of the TKNN graph in regularizing Z.

[Grouping effect] We showed in Lemma **??** that the rectified matrix Z^* and hence $\tilde{Z}$ have the desired grouping effect. Figure 9 visually compares the original similarity matrix S and $\tilde{Z}$ for *COIL20*. Each figure displays values in a matrix by pixel brightness. Rows and columns in the matrix are reordered by gold-standard clusters. From Figure 9(b), we see a clear well-defined block diagonal matrix with 20 sub-blocks. Compared with the fuzzy display of S in Figure 9(a), we see that $\tilde{Z}$ is much more effective in spectral clustering. This explains the excellent performance of ROSC in clustering *COIL20*.

[TKNN graph] We end this section with a further discussion on the TKNN graph regularizing matrix Z (via the reachability matrix $\mathcal{W}$). From Equation **??**, we see that the degree of such regularization is controlled by the parameter α_2. The larger α_2 is, the more dominant is the TKNN graph regularization in the objective function. Figures 10(a) and (b) show the performance scores of ROSC as α_2 varies for the dataset *COIL20* and *glass*, respectively. Note that the x-axes are shown in log scale. We note that trends of the curves in the two figures are quite different: For *COIL20*, the performance

Dataset	NJW	NCuts	PIC	PIC-k	DPIC	DPIE	ZP	FUSE	ROSC-R	ROSC
COIL20	0.4115	0.3926	0.2801	0.2801	0.2361	0.3496	0.5028	0.4177	0.4715	**0.9398 (1)**
seg_7class	0.5608	0.5403	0.3483	0.3566	0.3000	0.4756	0.5143	0.5912	0.6209	**0.6636 (1)**
glass	0.5234	0.5187	0.4976	0.5029	0.5245	0.5158	0.5374	0.5390	0.5748	**0.5760 (1)**
MNIST0127	0.5066	0.4970	0.4975	0.4924	0.5898	0.4395	0.5066	0.6436	0.6649	**0.6666 (1)**
isolet_5class	0.8120	0.7967	0.5863	0.5867	0.3033	**0.8572**	0.7767	0.7825	0.8495	0.8179 (3)
Yeast_4class	0.4819	0.4665	0.4428	0.4557	0.3831	0.4671	0.4819	**0.4999**	0.4877	0.4933 (2)
Yale_5class	0.5273	0.5091	0.4516	0.4596	0.4000	0.5225	0.5091	**0.5458**	0.5295	0.5455 (2)

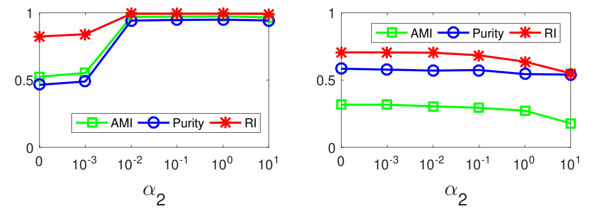
Table 4: Purity scores, real datasets

Dataset	NJW	NCuts	PIC	PIC-k	DPIC	DPIE	ZP	FUSE	ROSC-R	ROSC
COIL20	0.4718	0.4258	0.2989	0.2781	0.2507	0.3642	0.5702	0.4448	0.5291	**0.9682 (1)**
seg_7class	0.5043	0.4603	0.2339	0.2385	0.0915	0.3954	0.4298	0.5049	0.5255	**0.5730 (1)**
glass	**0.3469**	0.3465	0.3162	0.3193	0.2807	0.2683	0.3426	0.2589	0.3137	0.3204 (4)
MNIST0127	0.4353	0.4241	0.3623	0.3822	0.3714	0.2059	0.4219	0.4125	**0.4920**	0.4826 (2)
isolet_5class	**0.7595**	0.7204	0.5280	0.5292	0.0489	0.7481	0.7379	0.6516	0.7356	0.7524 (2)
Yeast_4class	0.1173	0.1052	0.1081	0.1165	0.0214	0.1318	0.1138	**0.1816**	0.1503	0.1582 (2)
Yale_5class	0.3121	0.3321	0.2357	0.2320	0.1468	0.3305	0.2788	**0.3495**	0.3201	0.3448 (2)

Table 5: AMI scores, real datasets

Dataset	NJW	NCuts	PIC	PIC-k	DPIC	DPIE	ZP	FUSE	ROSC-R	ROSC
COIL20	0.7303	0.6245	0.4940	0.4481	0.7737	0.6114	0.8534	0.7424	0.8264	**0.9923 (1)**
seg_7class	0.8242	0.7962	0.4830	0.5000	0.7212	0.7162	0.8208	0.8210	0.8357	**0.8549 (1)**
glass	0.6890	0.6880	0.6808	0.6851	0.6556	0.6281	0.6949	0.6693	0.7036	**0.7054 (1)**
MNIST0127	0.5683	0.5459	0.5941	0.5887	0.6598	0.4648	0.6018	0.7022	0.7382	**0.7388 (1)**
isolet_5class	0.9058	0.8942	0.7288	0.7296	0.6792	**0.9123**	0.8993	0.8695	0.9016	0.9001 (4)
Yeast_4class	0.6046	0.5929	0.5643	0.5733	0.5770	0.5037	0.6201	0.6346	**0.6405**	**0.6405 (1)**
Yale_5class	0.7626	0.7519	0.6772	0.6843	0.6846	0.7542	0.7600	0.7363	0.7578	**0.7667 (1)**

Table 6: Rand index scores, real datasets



(a) COIL20 **(b) glass**

Figure 10: ROSC's performance scores vs. α_2

drops when α_2 is very small, while for *glass*, it is the other way round. The reason is that *COIL20* is highly multi-scale (see Table 3). In particular, there are very big clusters whose objects could be at large distances from each other. The TKNN graph has the effect of capturing the reachability similarities of these objects despite their weak feature similarities, and hence maintain their strong correlations. At very small α_2, the regularization effect is given too small a weight for it to be effective, leading to smaller performance scores. For *glass*, the dataset is much less multi-scale. The performance of ROSC stays very stable over a range of α_2 values until α_2 becomes very large, in which case, reachability similarity over-dominates feature similarity, resulting in a mild dip in performance.

6 CONCLUSIONS

In this paper we studied the effectiveness of spectral clustering methods in handling multi-scale data. We discussed the traditional approaches of locally scaling the similarity matrix and power-iteration-based techniques. We described the methods ZP and FUSE, which were previously proposed to cluster multi-scale data. We pointed out that these existing approaches focus on measuring the correlations of objects via feature similarity. However, for data with various sizes and densities, objects that belong to the same big cluster could be at substantial distances from each other. Feature similarity could fail in this case. In view of this, we proposed ROSC, which computes an affinity matrix $\tilde{Z}$ that takes into account both feature similarity and reachability similarity. In particular, $\tilde{Z}$ is obtained by regularizing a primitive affinity matrix with a TKNN graph. We mathematically proved that $\tilde{Z}$ has the desired grouping effect, which makes it a very effective replacement of the similarity matrix S used in spectral clustering. We conducted extensive experiments comparing the performance of ROSC against 9 other methods. Our results show that ROSC provides very good performances across all the datasets, both real and synthetic, we tested. In particular, for cases where the datasets are highly multi-scale, ROSC substantially outperforms other competitors. ROSC is therefore a very robust solution for clustering multi-scale data.

7 ACKNOWLEDGMENTS

This research is supported by Hong Kong Research Grants Council GRF HKU 17254016.

REFERENCES