Large-Scale Spectral Clustering on Graphs

Jialu Liu Chi Wang Marina Danilevsky Jiawei Han
University of Illinois at Urbana-Champaign, Urbana, IL
{jliu64, chiwang1, danilev1, hanj}@illinois.edu

Abstract

Graph clustering has received growing attention in recent years as an important analytical technique, both due to the prevalence of graph data, and the usefulness of graph structures for exploiting intrinsic data characteristics. However, as graph data grows in scale, it becomes increasingly more challenging to identify clusters. In this paper we propose an efficient clustering algorithm for large-scale graph data using spectral methods. The key idea is to repeatedly generate a small number of "supernodes" connected to the regular nodes, in order to compress the original graph into a sparse bipartite graph. By clustering the bipartite graph using spectral methods, we are able to greatly improve efficiency without losing considerable clustering power. Extensive experiments show the effectiveness and efficiency of our approach.

1 Introduction

Graph data, composed of a large number of data objects interconnected with each other via meaningful relationships, has become increasingly prominent in real life. Examples include friendship graphs in Facebook, web pages connected by hyperlinks, and co-author graphs in bibliographic data. In many machine learning applications, pairwise similarities between data objects can be modeled using graphs, and the subsequent problem of data clustering can be viewed as a graph clustering problem.

Graph clustering aims to partition the nodes into densely connected subgraphs such that nodes within the same cluster have more connections than those in different clusters. Discovering clusters in graphs not only helps visualize and define hierarchies [Herman *et al.*, 2000], but is also meaningful for many real world problems, such as community detection [Fortunato, 2010; Smyth and White, 2005] and outlier detection [Gupta *et al.*, 2012]. In addition, clustering results can be used as building blocks for many other algorithms to reduce the graph and model complexity [Song *et al.*, 2008; Dalvi *et al.*, 2008] as graphs grow in size, it becomes difficult to use or interpret these methods without some form of summarization. However, discovering clusters in a graph itself is challenging as the size of the graph grows extremely large, which is especially common in today's era of "big data". Thus, it is urgent and meaningful to develop efficient and effective clustering algorithms for large-scale graphs.

In this paper, we propose such an algorithm using spectral methods. Spectral Clustering has been widely used for effective graph clustering [Shi and Malik, 1997]. Many previous studies have examined accelerating spectral clustering. Most of these [Shinnou and Sasaki, 2008; Yan *et al.*, 2009; Sakai and Imiya, 2009; Chen and Cai, 2011] have been devoted to data represented in a feature space instead of a graph. Others are designed to achieve efficiency by finding numerical approximations to eigenfunction problems [Fowlkes *et al.*, 2004; Chen *et al.*, 2006; Liu *et al.*, 2007] or adapting standard eigensolvers to distributed architecture [Chen *et al.*, 2011; Miao *et al.*, 2008]. In contrast, we aim to mitigate the computational bottleneck by reducing the size of the graph, while still providing high-quality clustering results, as compared to standard spectral methods. Specifically, we generate meaningful *supernodes* which are connected to the original graph. Correspondingly, we obtain a *bipartite* structure which preserves the links between original graph nodes and the new supernodes. In this representation, we expect these supernodes to behave as cluster indicators that may guide the clustering of nodes in the original graph. Furthermore, the supernode clustering and regular node clustering should mutually help induce each other. In this way, the clustering of the original graph can be solved by clustering the bipartite graph. By controlling the number of supernodes and enforcing the sparsity of the generated bipartite graph, we are able to efficiently achieve this goal.

It is worthwhile to highlight several key aspects of the proposed approach:

1. As far as we know, this is the first exploration of a large-scale graph clustering algorithm which uses spectral methods on a transformation of the graph structure. The proposed design greatly reduces time requirements without considerably impacting performance.
2. Our method scales well. Most of the computations are simply matrix multiplications and thus can be easily implemented and efficiently processed in distributed or multi-core architecture.
3. We propose two alternative approaches in this paper. One uses a one-step generation of supernodes, and the other adopts an iterative framework to regenerate su-

pernodes based on the clustering result obtained from the previous step.

2 Related Work

The general spectral clustering method [Ng *et al.*, 2001; Shi and Malik, 1997] was first shown to work on data represented in feature space. As we are mainly interested in graph data, we need one more step to construct an adjacency matrix which takes $O(n^2p)$ time where n and p represent the number of data points and the number of features respectively. Calculating the eigenvalue-decomposition of the corresponding Laplacian matrix is the real computational bottleneck, requiring $O(n^3)$ time in the worst case. Therefore, applying spectral clustering for large-scale data becomes impossible for many applications.

In recent years, many works have been devoted to accelerating the spectral clustering algorithm. Among them, [Fowlkes *et al.*, 2004] adopts the classical Nyström method, which was originally proposed to find numerical approximations to eigenfunction problems. It chooses samples randomly to obtain small-size eigenvectors and then extrapolates these solutions. [Shinnou and Sasaki, 2008] reduces the original data set to a relatively small size before running spectral clustering. Similar to this idea, in [Yan *et al.*, 2009], all data points are collapsed into centroids through k-means or random projection trees so that eigenvalue-decomposition only needs to be applied on the centroids. [Sakai and Imiya, 2009] uses random projection in order to reduce data dimensionality. Random sampling has also been applied to reduce the size of data points within the eigenvalue-decomposition step. [Chen *et al.*, 2006; Liu *et al.*, 2007] introduce early stop strategies to speed up eigenvalue-decomposition based on the observation that well-separated data points will converge to the final embedding more quickly. In [Chen and Cai, 2011], landmark points are first selected among all the data points to serve as a codebook. After encoding all data points based on this codebook, acceleration can be achieved using the new representation. The authors in [Khoa and Chawla, 2012] work on resistance distance embedding, which employs a similar idea to spectral clustering and exhibits quite similar clustering capability.

3 Revisit of Spectral Clustering

In this section, we briefly introduce the spectral clustering algorithm [Shi and Malik, 1997; Ng *et al.*, 2001]. Suppose that we are given an undirected and weighted graph $G = (V, E)$. Assume $|V| = n$ represents the number of nodes and $|E| = m$ represents the number of edges or links[1]. We can then use a non-negative weighted $n \times n$ *adjacency matrix* W to describe G, where $W = \{W_{ij}\}_{i,j=1,\ldots,n}$. Based on W, the *Laplacian matrix* L is defined as follows:

$$L = D - W$$

where D is a diagonal matrix whose entries are column (or row, since W is symmetric) sums of W. Based on this Laplacian matrix, spectral clustering aims to find k orthonormal column vectors $X_1, X_2, \ldots, X_k$ with the objective function:

$$\min_X \; Tr(X^T D^{-1/2} L D^{-1/2} X) \quad \text{s.t. } X^T X = I$$

where $X \in \mathbb{R}^{n \times k}$ is a matrix consisting of the column vectors and k is number of clusters.

It turns out that these vectors are indeed the eigenvectors corresponding to k smallest eigenvalues obtained from the eigenvalue-decomposition (EVD) on $D^{-1/2} L D^{-1/2}$. After row normalization of these eigenvectors, one can apply any classical clustering algorithms such as k-means to partition these low-dimensional *embeddings*. It is worth noting that the complexity of EVD on an $n \times n$ graph is $O(n^3)$ without considering the sparsity or calculating limited pairs of eigenvalues and eigenvectors.

4 Large-Scale Spectral Clustering on Graphs

Now we introduce our *Efficient Spectral Clustering on Graphs (ESCG)* for large-scale graph data. The basic idea of our approach is designing an efficient way to coarsen the graph by generating *supernodes* linked to the nodes in the original graph. A bipartite graph between the nodes in G and the generated supernodes is then constructed to replace G, so that the original high-dimensional EVD can be avoided.

4.1 Generation of Supernodes

Given the initial graph G of n nodes, we want to generate a set of d supernodes to coarsen the graph under the condition that $d \ll n$. Inspired by the intuition behind *simultaneous* or *co-clustering* [Dhillon, 2001], which says that clustering results of two related object types can be mutually enhanced, we expect that a partition of supernodes can induce a partition of the observed nodes, whereas a partition of the observed nodes can imply a partition of supernodes. Therefore, we first develop a simple and efficient algorithm to establish an initial clustering on graph G. Then we generate supernodes based on this initial clustering.

Our proposed approach works as follows: Given a graph G, we randomly pick d seeds in the graph and compute shortest paths from these seeds to the rest of the nodes. We then partition all the nodes into d disjoint subsets represented by the seeds: each node chooses the representative seed with the shortest distance.

To solve the shortest path problem, we first transform the edge weight demonstrating the similarity into distance:

$$M_{ij} = -\log \frac{W_{ij}}{\max W} + \varepsilon$$

where ε is a very small number that functions as the additional decay along the path. We incorporate the decay in order to prevent the possible distance between two nodes from being 0. The logarithmic transformation is adopted because the summation of M_{ij}'s along the paths in the graph can be viewed as the multiplication of the edge weights, which makes sense for estimating the distance for any pair of nodes.

After this step, the range of the distance value on each edge should be within $[\varepsilon, +\infty)$. Dijkstra's algorithm is then adopted to compute the shortest paths from seeds to the rest of the graph, which takes $O(md + nd \log n)$ time.

[1] In the following sections, edges and links are used interchangeably.

After running Dijkstra's algorithm, we are able to partition all nodes in G into d disjoint subsets by comparing their shortest paths to the seeds. We assign each node to the partition with the closest seed. Note that this step can be implemented in parallel.

Given these d disjoint subsets, d supernodes in total are assigned accordingly. The linkage between supernodes and regular nodes in G is described by a binary matrix $R \in \mathbb{R}_{d\times n}$. Each element in R is set to be 1 if there is a link between supernode and regular node. It should be noted that R is also a sparse matrix whose Frobenius norm is equal to $\sqrt{n}$. At this point, we know that R contains the edges between supernodes and regular nodes while W describes the edges within G. We therefore absorb W into R to finish transforming G into the bipartite graph. In a *bipartite* graph, the nodes can be divided into two disjoint groups, such that edges only exist between nodes in different groups. Denote the associated edge weight matrix $\hat{W} \in \mathbb{R}_{d\times n}$ of the bipartite graph as follows:

$$\hat{W} = RW. \tag{1}$$

The design of this step can also be explained with the fact that R is essentially a one-to-many bipartite graph whereas $\hat{W}$ is a many-to-many graph. Only in the latter case is each supernode related to many regular nodes and vice versa. Obviously, this many-to-many relationship can be further modeled to achieve the goal mentioned at the beginning of this subsection, i.e., to propagate clustering information between nodes located on both sides of the bipartite graph. It is also worth noting that in order to obtain better clustering power, intuitively d should be set larger so that the information loss in going to $\hat{W}$ from W is smaller.

4.2 Spectral Clustering on Reduced Graphs

Through the transformation to the bipartite graph, we significantly reduce the size of the full edge weight matrix of G from $n \times n$ to $d \times n$. In this section, we introduce how to convert the EVD of the graph Laplacian mentioned in the previous section into a singular value decomposition (SVD) problem, such that the overall time complexity is $O(md + nd\log n + nd^2)$, which is a significant reduction from $O(n^3)$ since $d \ll n$.

To begin, we give the adjacency matrix of the bipartite graph described above:

$$W' = \begin{bmatrix} 0 & \hat{W}^T \\ \hat{W} & 0 \end{bmatrix}.$$

Here, we use the "prime" notation to denote the adjacency matrix, which is a square matrix of size $(n+d)\times(n+d)$. Hence we also have representations for L' and D' in this bipartite model:

$$L' = \begin{bmatrix} D_1 & -\hat{W}^T \\ -\hat{W} & D_2 \end{bmatrix}, \text{and} \quad D' = \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix}$$

where D_1 and D_2 are two diagonal matrices whose entries are column and row sums of $\hat{W}$, respectively.

The generalized eigenvalue-decomposition (GEVD) of L' and D' can be written as

$$\begin{bmatrix} D_1 & -\hat{W}^T \\ -\hat{W} & D_2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \lambda \begin{bmatrix} D_1 & 0 \\ 0 & D_2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}. \tag{2}$$

Algorithm 1 Efficient Spectral Clustering on Graphs (ESCG)

Input: Adjacency matrix $W \in \mathbb{R}_{n\times n}$ for graph G, number of clusters k, desired number of supernodes d

Output: Clustering of nodes in G

1: Randomly sample d seeds and apply Dijkstra's algorithm to compute shortest paths.
2: Partition nodes into d disjoint subsets with shortest paths.
3: Generate d supernodes and build their connections to d disjoint subsets respectively with matrix $R \in \mathbb{R}_{d\times n}$.
4: Compute matrix $\hat{W}$ indicating the adjacency for the transformed bipartite graph by Eq. 1.
5: Compute $Z = D_2^{-1/2}\hat{W}D_1^{-1/2}$ where D_1 and D_2 are diagonal matrices of column and row sums of $\hat{W}$.
6: Obtain the largest k eigenvalues and corresponding eigenvectors from ZZ^T.
7: Generate right singular vectors X of Z as in Eq. 4 .
8: Form the matrix $U = D_1^{-1/2}X$.
9: Treat each row of U as a node in $\mathbb{R}^k$, partition these nodes into k clusters via k-means algorithm.

Under the representation of $x = D_1^{1/2}u$ and $y = D_2^{1/2}v$, Eq. 2 becomes

$$D_2^{-1/2}\hat{W}D_1^{-1/2}x = (1-\lambda)y$$
$$D_1^{-1/2}\hat{W}^T D_2^{-1/2}y = (1-\lambda)x.$$

For convenience, we use Z to denote $D_2^{-1/2}\hat{W}D_1^{-1/2}$ and the relationship between x and y is then seen more clearly:

$$Zx = (1-\lambda)y, \quad Z^T y = (1-\lambda)x. \tag{3}$$

It is easy to verify that x retains the clustering information of original nodes in G, and y retains that of the supernodes, which are inherited from u and v, respectively. Therefore, Eq. 3 justifies the intuition that the partitions of the original nodes and those of the supernodes can induce each other through linear operations. In fact, x and y are mutually affected by Z, which is essentially a normalized edge weight matrix between the original nodes and the supernodes. However, x and y do not directly indicate the cluster membership of each node in the bipartite graph since they themselves are normalizations of u and v where u and v are real relaxations into discrete partitions, as in spectral clustering.

In addition, Eq. 3 demonstrates that x and y are the left and right singular vectors corresponding to the largest singular values of Z. Thus, the GEVD of L' and D' that was previously mentioned is not necessary to save computation time. Note that the size of $D_2^{-1/2}\hat{W}D_1^{-1/2}$ remains the same as $\hat{W}$, i.e., $d \times n$. Thus, a better way to compute x is to first estimate y because d is much smaller than n,.

Denote the SVD of Z as follows:

$$Z = Y\Sigma X^T$$

where $\Sigma = Diag(\sigma_1, \sigma_2, \cdots, \sigma_d)$ and $Y \in \mathbb{R}_{d\times d}$ ($X \in \mathbb{R}_{n\times d}$) are left (right) singular vectors. Moreover, X and Y are two matrices formed by pairs of column vectors x and y satisfying Eq. 3. It is easy to check that the left singular

Algorithm 2 Regeneration of Supernodes

Input: Embedding matrix $U \in \mathbb{R}_{n\times k}$
Output: Sparse matrix R describing the links between supernodes and regular nodes in G

1: Create a sparse matrix $R \in \mathbb{R}_{(2k-2)\times n}$
2: **for** $i = 2$ **to** k **do**
3: Compute mean $\overline{U}_{\cdot,i}$ of vector $U_{\cdot,i}$.
4: Set $R_{2\times i-3,j}$ to 1 if $U_{j,i} \leq \overline{U}_{\cdot,i}$ for all $j \in 1, \cdots, n$.
5: Set $R_{2\times i-2,j}$ to 1 if $U_{j,i} > \overline{U}_{\cdot,i}$ for all $j \in 1, \cdots, n$.
6: **end for**

vectors of Z are exactly the eigenvectors of ZZ^T and the corresponding eigenvalues are $\sigma_1^2, \sigma_2^2, \cdots, \sigma_d^2$. Thus, by performing eigenvalue-decomposition of ZZ^T, we can directly obtain Y and Σ^2. As these singular values are all nonnegative real numbers and Y is a unitary matrix, we can get

$$X^T = \Sigma^{-1} Y^T Z. \tag{4}$$

As the final step, k-means is usually adopted to cluster the "top" k column vectors of normalized X. Other clustering approaches such as hierarchical k-means or approximate k-means may also be used to achieve better efficiency. Alg. 1 summarizes the algorithm of *Efficient Spectral Clustering on Graphs* (ESCG). Note that in order to obtain k column vectors from SVD, the number of supernodes should be at least k to ensure that the rank of matrix Z is large enough.

4.3 Regeneration of Supernodes

In the aforementioned approach, supernodes are connected to regular nodes according to the shortest paths to randomly selected seeds. The supernodes thus behave like cluster indicators responsible for propagating knowledge of the original graph. However, once we get the clustering result using spectral techniques discussed above, we may use this knowledge to form non-random supernodes and improve the final results.

We therefore propose an iterative way to regenerate the supernodes based on the current clustering results, aiming to repeatedly improve the clustering. In particular, it is natural to require each supernode to link to a set of densely connected nodes, which themselves form a better local cluster than the random sampling method discussed in Sec. 4.1. Also the process of discovering such local clusters must be efficient. Inspired by the fact that the column vectors of the embedding matrix U can be used to indicate partitions of nodes in the graph [Shi and Malik, 1997], supernodes can be guided to connect to nodes which form local clusters that are inferred from the element values in the column vectors of U.

In our approach, we generate $(2k-2)$ supernodes in total, corresponding to the right $k-1$ columns of the embeddings (the leftmost column is ignored since it contains constant elements.) For details please refer to Alg. 2, where we assume the first column vector to be filtered, for simplicity.

Alg. 3 gives the procedure of *Efficient Spectral Clustering on Graphs with Regeneration* (ESCG-R).

4.4 Computational Complexity Analysis

For ESCG, the supernodes generation step (lines 1-4 in Alg. 1) takes $O(m(d+1) + nd\log n)$ time, as it is simply

Algorithm 3 Efficient Spectral Clustering on Graphs with Regeneration (ESCG-R)

Input: Adjacency matrix $W \in \mathbb{R}_{n\times n}$ for graph G, number of clusters k, desired number of supernodes d, number of iterations t
Output: Clustering of nodes in G

1: **for** $i = 1$ **to** t **do**
2: **if** $i = 1$ **then**
3: Create matrix $R \in \mathbb{R}_{d\times n}$ as in Alg. 1 (lines 1 - 4).
4: **else**
5: Create matrix $R \in \mathbb{R}_{(2k-2)\times n}$ as in Alg. 2 .
6: **end if**
7: Same with lines 4-8 as in Alg. 1.
8: **end for**
9: Treat each row of U as a node in $\mathbb{R}^k$, partition these nodes into k clusters via k-means algorithm.

a combination of Dijkstra's algorithm and a matrix multiplication operation. ESCG then needs $O(nd)$ time to compute Z and $O(nd^2)$ time for ZZ^T. The EVD step[2] of ZZ^T (line 6) has time complexity $O(d^3)$, and obtaining the right singular vectors X takes $O(ndk)$ time. Summing these gives $O(m(d+1)+nd\log n+nd+nd^2+d^3+ndk)$. Since $d \ll n$ and $k \leq d$, we can preserve just the dominant components, yielding a total complexity of $O(md + nd\log n + nd^2)$.

Compared to ESCG, the main difference in ESCG-R is that the number for supernodes in later iterations (line 5) decreases from d to $2k-2$. The complexity of this step is $O(mk)$. As the loop (lines 1 to 12) takes t iterations, the total time complexity is $O(md + nd\log n + mkt + n(d^2 + k^2 t))$.

5 Experiments

In this section we present several experiments to show the effectiveness of our proposed approach. We begin with the description of the data sets used in our experiment.

5.1 Data Sets

Two synthetic data sets and two real world data sets are used in our experiments. The important statistics of these data sets are summarized below (see also Table 1):

- **Syn-1K** is a synthetic data set generated from k-NN graph of three circles similar to Fig. 1(a) and (b).
- **Syn-100K** is a synthetic data set generated according to [Brandes *et al.*, 2003] where each node has about 40 out-cluster and 120 within-cluster neighbors.
- **DBLP** is a collection of bibliographic information on major computer science journals and proceedings[3]. We use a subset of the DBLP records that belong to four areas: artificial intelligence, information retrieval, data mining and database. We generate a graph of authors linked by the co-conference relationship.
- **IMDB** is an international organization whose objective is to provide useful and up-to-date movie information[4]. We create a graph of movies linked by the co-director

[2]We consider EVD using QR decomposition for simplicity.
[3]http://www.informatik.uni-trier.de/ ley/db/
[4]http://www.imdb.com/

relationship. The genre information is considered to be the clustering label. We use four genres: documentary, music, comedy and adult.

Table 1: Statistics of the four data sets

data set	nodes	edges	clusters	sparsity
Syn-1K	1,000	10,000	3	0.01
Syn-100K	100,000	8.2M	10	0.0016
DBLP	28,702	62.4M	4	0.1515
IMDB	30,731	257K	4	5×10^{-4}

5.2 Algorithms for Comparison

To demonstrate the effectiveness and efficiency of our method, we compared with the following clustering algorithms which can be applied to graph data directly:

- Shortest Paths (**SP**) algorithm. To show our method can significantly improve the clustering result from the initial partition obtained from shortest path algorithm, we report its performance by selecting k random seeds.
- Efficient Spectral Clustering on Graphs (**ESCG**), the method proposed in this paper. There is one parameter in the ESCG algorithm: the number of supernodes d. In our experiments, we empirically set it to 30. The model selection is discussed in the Section 5.6.
- Efficient Spectral Clustering on Graphs with Regeneration (**ESCG-R**), the second method proposed in this paper, which regenerates supernodes. The number of iterations is empirically set to be 5 and related experiments can be found in the Section 5.6.
- Standard Spectral clustering (**SC**) algorithm. We implemented the algorithm in [Ng *et al.*, 2001].
- Resistance Embedding Spectral Clustering (**RESC**) from [Khoa and Chawla, 2012], an efficient spectral clustering method that can be applied to graph data. We implement the algorithm and set the parameter K_{RP} to be 50 as suggested by the authors.
- **Nyström** [Fowlkes *et al.*, 2004], a method that aims to find numerical approximations to eigenfunction problems. We use the Matlab implementation online[5].

In order to randomize the experiments, we conduct 20 test runs with different random initializations and report the average performance.

5.3 Evaluation Metric

The clustering result is evaluated by comparing the obtained label of each sample using clustering algorithms with that provided by the data set. The accuracy metric (AC) [Chen and Cai, 2011] is used to measure the performance.

We also report the running time of each method. All coding was implemented in MATLAB R2011b and run on a desktop with Intel Core I7 2600 and 16GB memory.

5.4 Experimental Results

Table 2 shows the clustering performance of the different algorithms on all four data sets. We can see that ESCG always outperforms SP - which is in fact a module of ESCG for generating supernodes. ESCG-R achieves better performance than ESCG since the process of regenerating supernodes based on the embeddings of ESCG makes use of more knowledge from the original graph.

Table 2: Clustering accuracy on the four data sets (%)

Data sets	Syn-1K	Syn-100K	DBLP	IMDB
SP	73.6	37.6	58.0	46.3
ESCG	**100**	97.0	70.6	49.5
ESCG-R	**100**	**100**	**82.1**	51.8
SC	**100**	**100**	78.2	55.2
RESC	**100**	10.6	27.9	**58.2**
Nyström	40.0	17.1	78.4	40.1

Table 3: Running time on the four data sets (s)

Data sets	Syn-1K	Syn-100K	DBLP	IMDB
SP	0.001	0.33	1.72	0.02
ESCG	**0.023**	**1.36**	**3.81**	**0.17**
ESCG-R	0.041	7.38	9.23	0.49
SC	0.228	5.06	27.7	74.4
RESC	0.166	201	2394	24.9
Nyström	0.033	19.1	3.89	4.41

On the Syn-1K data set, since the k-NN graph is relatively clean, ESCG, ESCG-R, SC and RESC all achieve 100% accuracy. Despite SP's significant efficiency, its accuracy is only 73.6% which is 26.4% less than all other methods. Although both ESCG and ESCG-R use SP to generate supernodes, they both significantly improve on SP's performance in the clustering task. Nyström performs poorly on this data set since Syn-1K is sparse and finding approximations of eigenvalue-decomposition using the sampling technique on a sparse matrix is very difficult. To further study this data set, we add noise into the graph, as visualized in Fig. 1 (a) and (b). The results of our proposed methods and other algorithms are shown in Fig. 1 (c) to (h). Based on the initial clustering from shortest path calculations, ESCG significantly refines the result and ESCG-R further improves performance.

On the Syn-100K data set, SP is surprisingly unable to find a good partition on the graph, and only achieves 37.6% in terms of accuracy. This is due to the fact that each node has 40 out-cluster links and in this case, shortest path length may not be a good feature for clustering. However, ESCG is still able to achieves 97% accuracy, again demonstrating its effectiveness over SP. With regeneration of supernodes, ESCG-R finally reaches 100% accuracy, the same as SC. It is worth noting that RESC performs on the level of random guessing on this data set. This may be due to the random projection step within the algorithm since too many edges are intra-cluster. Nyström once again performs poorly due to graph sparsity.

On the DBLP data set, ESCG-R achieves the best performance, beating the second best Nyström by 3.9%. Nyström shows its power on the DBLP data set because this graph is much more dense. RESC posts a similarly poor performance on this data set as on the Syn-100K data set.

On the IMDB data set, all algorithms are relatively close. Among them, RESC has 58.2% accuracy and SC has 55.2%. Our proposed ESCG and ESCG-R perform a little worse, with accuracies of 49.5% and 51.8%, respectively. Due to

[5]http://alumni.cs.ucsb.edu/∼wychen/

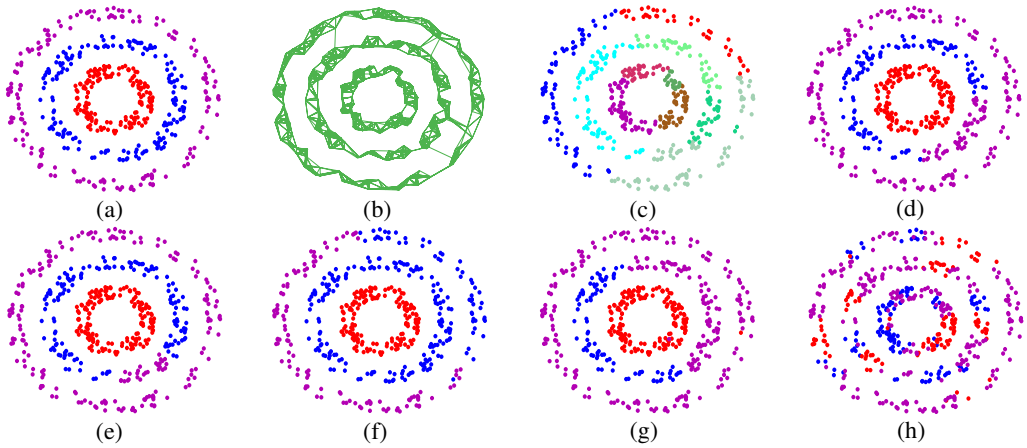

Figure 1: Toy example on the synthetic data set Syn-1K. Top 4 figures, from left to right: (a) ground-truth partition; (b) k-NN graph; (c) initial clustering results by shortest path, when $d = 10$; (d) clustering results by ESCG. Bottom 4 figures, from left to right: clustering results by (e) ESCG-R after 5 iterations; (f) spectral clustering; (g) RESC; (h) Nyström method.

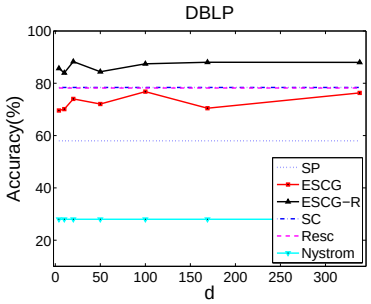

Figure 2: Performance of ESCG/ESCG-R w.r.t. parameter d.

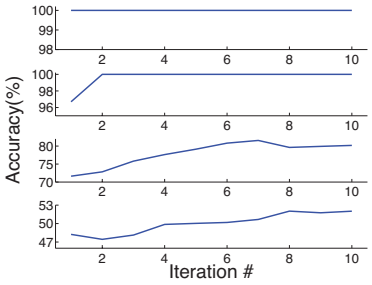

Figure 3: ESCG-R w.r.t. # iteration on the four data sets.

the graph sparseness, Nyström only shows 40.1% accuracy, nearly 20% lower than the best.

In conclusion, we find that all the scalable clustering methods in our experiment including ours cannot always beat SC, demonstrating its strength and stability. However, our proposed methods, ESCG and ESCG-R, are relatively stable and effective, outperforming other efficient algorithms.

5.5 Complexity Study

Table 3 lists the running time for different algorithms on all four data sets. For the convenience of testing, we do not incorporate the running time for k-means because this step is shared by all of the algorithms except SP. As shown in the table, ESCG takes the least amount of time to obtain the embeddings with the exception of SP, which is essentially one module in our algorithm. When the graph is dense, we note that RESC becomes extremely slow since it needs to construct a diagonal matrix of all edges and to apply random projection and some additional operations on the matrix. ESCG-R adopts an iterative framework and therefore should always take more time than ESCG. However, due to the small size of the regenerated supernodes, the time complexity does not increase much, as demonstrated in the table.

5.6 Parameter Study

In this subsection, we study the parameters d and t in our proposed algorithms. As shown in Fig. 2, in order to obtain better clustering power, d should be set a bit larger than k so that the information loss of $\hat{W}$ from W is smaller. Thus, in the experiments, we set d to be 30 which can ensure the efficiency while not sacrificing too much clustering power. Fig. 3 shows the performance of ESCG-R w.r.t. varying number of iterations t on the four data sets. It can be seen that the performance generally improves as the number of iterations grows.

6 Conclusion

In this paper we tackle the scalability issue of spectral clustering methods on large-scale graphs. We propose a method to reduce graph size, based on effectively compressing the graph information into a smaller number of "supernodes". Clustering supernodes with spectral methods is less expensive, and the clustering results can also be propagated back to the original graph with low cost. We reduce the computational complexity of spectral clustering significantly, from $O(n^3)$ to $O(md + nd \log n + nd^2)$. Although graph compression naturally induces inaccuracy, empirical studies demonstrate that our method can considerably decrease the necessary runtime while posting a tolerably small loss in accuracy.

7 Acknowledgment

This work was supported in part by the U.S. National Science Foundation grants IIS-0905215, U.S. Army Research Laboratory under Cooperative Agreement No. W911NF-09-2-0053 (NS-CTA).

References

[Brandes *et al.*, 2003] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. *Algorithms-ESA 2003*, pages 568–579, 2003.

[Chen and Cai, 2011] Xinlei Chen and Deng Cai. Large scale spectral clustering with landmark-based representation. In *AAAI*, 2011.

[Chen *et al.*, 2006] Bo Chen, Bin Gao, Tie-Yan Liu, Yu-Fu Chen, and Wei-Ying Ma. Fast spectral clustering of data using sequential matrix compression. In *ECML*, pages 590–597, 2006.

[Chen *et al.*, 2011] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and E. Y. Chang. Parallel Spectral Clustering in Distributed Systems. *TPAMI*, 33(3):568–586, 2011.

[Dalvi *et al.*, 2008] B.B. Dalvi, M. Kshirsagar, and S. Sudarshan. Keyword search on external memory data graphs. *PVLDB*, 1(1):1189–1204, 2008.

[Dhillon, 2001] I.S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD*, pages 269–274, 2001.

[Fortunato, 2010] Santo Fortunato. Community detection in graphs. *Physics Reports*, 486(3-5):75 – 174, 2010.

[Fowlkes *et al.*, 2004] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nyström method. *TPAMI*, 26(2):214–225, 2004.

[Gupta *et al.*, 2012] Manish Gupta, Jing Gao, Yizhou Sun, and Jiawei Han. Integrating community matching and outlier detection for mining evolutionary community outliers. In *KDD*, pages 859–867, 2012.

[Herman *et al.*, 2000] Ivan Herman, Guy Melançon, and M. Scott Marshall. Graph visualization and navigation in information visualization: A survey. *TVCG*, 6(1):24–43, 2000.

[Khoa and Chawla, 2012] N. Khoa and S. Chawla. Large scale spectral clustering using resistance distance and spielman-teng solvers. In *Discovery Science*, pages 7–21, 2012.

[Liu *et al.*, 2007] Tie-Yan Liu, Huai-Yuan Yang, Xin Zheng, Tao Qin, and Wei-Ying Ma. Fast large-scale spectral clustering by sequential shrinkage optimization. In *ECIR*, pages 319–330, 2007.

[Miao *et al.*, 2008] G. Miao, Y. Song, D. Zhang, and H. Bai. Parallel spectral clustering algorithm for large-scale community data mining. In *SWSM*, 2008.

[Ng *et al.*, 2001] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *NIPS*, pages 849–856. 2001.

[Sakai and Imiya, 2009] Tomoya Sakai and Atsushi Imiya. Fast spectral clustering with random projection and sampling. In *MLDM*, pages 372–384, 2009.

[Shi and Malik, 1997] J. Shi and J. Malik. Normalized cuts and image segmentation. *TPAMI*, 22(8):888 –905, 1997.

[Shinnou and Sasaki, 2008] Hiroyuki Shinnou and Minoru Sasaki. Spectral clustering for a large data set by reducing the similarity matrix size. In *LREC*, 2008.

[Smyth and White, 2005] P. Smyth and S. White. A spectral clustering approach to finding communities in graphs. In *SDM*, volume 119, page 274, 2005.

[Song *et al.*, 2008] Yang Song, Ziming Zhuang, Huajing Li, Qiankun Zhao, Jia Li, Wang-Chien Lee, and C. Lee Giles. Real-time automatic tag recommendation. In *SIGIR*, pages 515–522, 2008.

[Yan *et al.*, 2009] Donghui Yan, Ling Huang, and Michael I. Jordan. Fast approximate spectral clustering. In *KDD*, pages 907–916, 2009.