Large Scale Data Clustering and Graph Partitioning via Simulated Mixing

Shahzad Bhatti, Carolyn Beck, Angelia Nedić

Abstract—In this paper, we propose a new spectral clustering algorithm relying on a simulated mixing process over a graph. In contrast to existing spectral clustering algorithms, our algorithm does not necessitate the computation of eigenvectors. Alternatively, our algorithm determines the equivalent of a linear combination of eigenvectors of the normalized similarity matrix, which are weighted by the corresponding eigenvalues obtained by the mixing process on the graph. We use the information gained from this linear combination of eigenvectors directly to partition the dataset into meaningful clusters. Simulations on real datasets show that our algorithm achieves better accuracy than standard spectral clustering methods as the number of clusters increase.

I. INTRODUCTION

Data clustering presents one of the most fundamental problems in several disciplines, including pattern recognition, data mining, computer vision, machine learning and bioinformatics. Researchers from various fields have proposed numerous solutions to the data clustering problem; in particular, several spectral clustering algorithms have been proposed, examples include [1], [2], [3], [4]. These algorithms have been widely used to cluster data accurately. Unfortunately, spectral clustering does not scale well to large-scale problems. In general, spectral clustering algorithms attempt to find a low-dimensional embedding of the dataset by computing the eigenvectors of a Laplacian or similarity matrix. For a dataset with \( n \) instances, eigenvector computation has time complexity \( O(n^3) \) [5], which becomes intractable for large-scale problems.

Recently, research efforts have been focused on addressing the scalability of spectral clustering. One approach to achieve scalability is to perform spectral clustering on a random sample of the given dataset and then generalize the result to the full dataset. Specific approaches include the work by Fowlkes et al. [6], who find an approximate solution by first performing spectral clustering on a small random sample from the dataset and then using the Nystrom method; they extrapolate the solution to all the dataset. In [7], Sakai and Imiya also find an approximate spectral clustering by clustering a random sample of the dataset and use random projections to reduce the dimensions of the dataset. Another approach proposed by Yan et al. [8] is to determine a small set of representative points using \( k \)-means and then perform spectral clustering on the representative points. The original dataset is clustered by assigning each point to the cluster of its representative. In [9], Chen et al. deal with large-scale data by parallelizing both computation and memory use.

These methods trade-off the accuracy of spectral clustering for a fast implementation. In this paper, we perform spectral clustering without explicitly calculating eigenvectors. Alternatively, we compute a linear combination of the right-eigenvectors weighted with corresponding eigenvalues. Unlike many traditional algorithms, our algorithm does not require a predefined number of clusters, \( k \), as input, but can automatically detect and adapt to any number of clusters, based on a preselected tolerance. We evaluate our algorithm to large stochastic block models to illustrate the scalability. We compare the accuracy and speed of our algorithm to the normalized cut algorithm [1] and show that our approach achieves similar accuracy but at a faster speed. We also show that our algorithm is faster and more accurate than both the Nystrom method for spectral clustering[6] and the fast approximate spectral clustering [8].

Notation: We use boldface to distinguish between vectors and scalars. For example, \( \mathbf{v}_i \) is in boldface to identify a vector, while \( x_i \), a scalar, is not boldface. We use \( \mathbf{1} \) to denote the vector of ones. We denote matrices by capital letters, such as \( A \), and use \( a_{ij} \) to represent the entries of \( A \). Calligraphic font is used to denote sets with the single exception that \( G \) is reserved to denote graphs. The norm \( \| \cdot \| \) denotes \( \| \cdot \|_2 \) for vectors and spectral norm for matrices.

II. PROBLEM STATEMENT

Consider a set of \( n \) data points \( \mathcal{V} = \{ \mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n \} \) in a \( d \)-dimensional space. The goal is to find clusters in the dataset such that points in the same cluster are similar to each other, while points in differing clusters are dissimilar under some predefined notion of similarity. In particular, suppose pairwise similarity between points is given by some similarity function \( s(\mathbf{v}_i, \mathbf{v}_j) \), where usually it is assumed that the function \( s \) is symmetric. Also, the similarity function \( s \) is non-negative if \( i \neq j \) and is equal to zero if \( i = j \). A similarity matrix is an \( n \times n \) symmetric matrix \( W \) such that the entry \( w_{ij} \) is equal to the value of the similarity function \( s(\mathbf{v}_i, \mathbf{v}_j) \) between points \( \mathbf{v}_i \) and \( \mathbf{v}_j \).

The data points together with the similarity function form a weighted undirected similarity graph \( G = (\mathcal{V}, \mathcal{E}) \); \( \mathcal{V} \) is the set of nodes (vertices) of the graph, \( \mathcal{E} \) is the set of edges. We note that the problem of finding clusters in a dataset can be converted into a graph partitioning problem. We will make this more precise in the sequel. In particular, in our case
each data point $v_i$ represents a vertex/node in the graph. Two vertices $v_i$ and $v_j$ are connected by an edge if their similarity $s(v_i, v_j)$ is positive and the edge weight is given by $s(v_i, v_j)$. Different similarity measures lead to different similarity graphs. The objective of constructing a similarity graph is to model the local neighborhood relationships to capture the geometric structure of the dataset using the similarity function. In our simulation results, we use $p$-nearest neighbor similarity.

As the name suggests each vertex $v_i$ is connected to its $p$ nearest neighbors, where the nearness between $v_i$ and $v_j$ is measured by the distance $||v_i - v_j||$. This similarity measure results in a graph which is not necessarily symmetric in its similarity function, since the nearness relationship is not symmetric. In particular, if $v_j$ is among the $p$ nearest neighbors of $v_i$, then it is not necessary for $v_j$ to be among the $p$ nearest neighbors of $v_i$. We make the similarity measure symmetric by placing an edge between two vertices $v_i$ and $v_j$ if either $v_i$ or $v_j$ is among the $p$ nearest neighbors of the other, i.e., $v_i$ or $v_j$ are among the $p$ nearest neighbors of $v_i$, that is

$$s(v_i, v_j) = \begin{cases} 1 & \text{if either } v_i \text{ or } v_j \text{ is one of the } p \text{ nearest neighbors of the other,} \\ 0 & \text{otherwise.} \end{cases}$$

Thus finding clusters in a dataset is equivalent to finding partitions in the similarity graph such that the sum of edge weights between partitions is small and the partitions themselves are dense subgraphs. The degree of each vertex $i$ of the graph is given by $d_i = \sum_{j=1}^{n} w_{ij}$. A degree matrix $D$ is a diagonal matrix with its diagonal elements given by $d_i$ for $i = 1, \ldots, n$. We assume that the degree of each vertex is positive. This in turn allows us to define the normalized similarity matrix by $W = D^{-1}W$.

### III. MIXING PROCESSES

As a visualization of our clustering approach, consider a mixing process in which one imagines that every vertex in the graph moves towards (mixes with) other vertices in discrete time steps. At each time step, vertex $v_i$ moves towards (mixes with) vertex $v_j$ by a distance proportional to the similarity $s(v_i, v_j)$. Thus the larger the similarity $s(v_i, v_j)$, the larger the distance vertices $v_i$ and $v_j$ move towards each other i.e., the greater the mixing. Moreover, a point $v_i$ will move away from the points which have weak similarity with it. Thus, similar points will move towards each other making dense clusters and dissimilar points will move away from each other increasing the separability between clusters. Clusters in this transformed distribution of points then can easily be identified by the $k$-means algorithm.

To make this idea more precise, consider the model where each point $v_i$ moves according to the following equation, starting at its position at time $t = 0$:

$$v_i^{t+1} = v_i^t + \alpha \sum_{j=1}^{n} w_{ij} (v_j^t - v_i^t)$$

$$= (1 - \alpha)v_i^t + \alpha \sum_{j=1}^{n} w_{ij} v_j^t. \tag{1}$$

The parameter $\alpha \in [0, 1]$ is the step size, which controls the speed of movement (or mixing rate) in each time interval. Observe that if the underlying graph has a bipartite component and $\alpha = 1$, then in each time step all points on one side of this component would move to the other side and vice versa. Therefore, points in this component would not actually mix even after a large number of iterations (for details see [10]). For such graphs we must have $\alpha$ bounded away from 1. We can use $\alpha = 1$ for graphs without a bipartite component. Assuming each point $v_i$ is a row vector, we express (1) in a matrix form:

$$V^{t+1} = ((1 - \alpha)I + \alpha W)V^t = MV^t. \tag{2}$$

The matrix $V^t$ is a $n \times d$ matrix with row $i$ representing the position of point $v_i$ at time $t$. $I$ is a $n \times n$ identity matrix, and we define $M = (1 - \alpha)I + \alpha W$. Note that the matrix $M$ is essentially the transition matrix of a lazy random walk with probability of staying in place given by $1 - \alpha$. Since $M$ also captures the similarity of the data points, one would expect that for $t$ large enough, the process in (2) would reveal the data clusters, since $M$ will mix the data points according to their similarities. Using this intuition, one can expect that a heuristic algorithm based on (2) can be constructed to determine the clusters.

This approach has two limitations: a) it does not scale well with the dimension $d$ of the data points (typical number of computations $O(n^2d)$ per iteration), and b) it fails to identify clusters contained within other clusters, e.g. in the case of two concentric circular clusters, points in both clusters will move towards the center and become one cluster, losing the geometric structure inherent in the data. To overcome these limitations, we associate an agent $x_i$ to each point $v_i$ and carry out calculations in the agent space. Agents are generated by choosing $n$ points uniformly at random from an interval $[0, b]^1$. We rewrite (2) in the agent space as:

$$x^{t+1} = ((1 - \alpha)I + \alpha W)x^t = Mx^t. \tag{3}$$

We refer to this iterative equation as the Mixing Process. In the following section we analyze this process using the properties of the random walk matrix $M$.

### IV. ANALYSIS OF THE MIXING PROCESS

The matrix $M$ captures the similarity structure of the data, and the idea behind using the iterative process (3) is that, after some sufficient number of iterations, the entries of the vector $x^{t+1}$ will reveal clusters on a real line, which will be representative of the clusters in the data. The fact is that the process (2) and (3) both mix with the same speed, which is governed by the random walk $M$. Thus, the hope is that through the process in (3) we determine the weakly coupled components in the matrix $M$, which can lead us to the data clusters of the points $v_1, \ldots, v_n$.

The Mixing Process shares a resemblance to the power iteration method. Unlike power iteration, we here use the

\[1\text{Note, } b \text{ is a scaling parameter and does not change the resulting clustering. For the sake of simplicity we use a probability vector for the analysis in the subsequent section.}\]
mixing process to discover strongly coupled components of $M$, which translate to data clusters.

A. Properties of the matrix $M$

We first show that the matrix $M$ is diagonalizable, which allows for a more straightforward analysis. By definition,

$$M = (1 - \alpha)I + \alpha W = D^{-1/2}(I - \alpha L)D^{1/2},$$

where $L = I - D^{-1/2}WD^{-1/2}$ is the normalized Laplacian of the graph $\mathcal{G}$. Let $\phi_i$ be a right eigenvector of $L$ with eigenvalue $\lambda_i$, then $D^{-1/2}\phi_i$ is a right eigenvector of $M$ with eigenvalue $\mu_i = 1 - \alpha \lambda_i$, that is

$$MD^{-1/2}\phi_i = (1 - \alpha D^{-1/2}LD^{1/2})D^{-1/2}\phi_i = (1 - \alpha \lambda_i)D^{-1/2}\phi_i.$$  

This gives us a useful relationship between the spectra of the random walk matrix $M$ and the normalized Laplacian $L$. It is well known that the eigenvalues of a normalized Laplacian lie in the interval $[0, 2]$, see for example [10]. Thus, if $0 = \lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n \leq 2$ are the eigenvalues of $L$, then the corresponding eigenvalues of $M$ are $1 = \mu_1 \geq \mu_2 \geq \ldots \geq \mu_n \geq 1 - 2\alpha$. It is worth noting that we are considering the right eigenvector of the random walk matrix $M$. One should not confuse this with the left eigenvector.

Although the matrix $M$ is not symmetric, $L$ is a symmetric positive semi-definite matrix, thus its normalized eigenvectors form an orthonormal basis for $\mathbb{R}^n$ expressed as $L = \sum_{i=1}^n \lambda_i \phi_i \phi_i^T$.

B. The ideal case

For the sake of analysis, it is worthwhile to consider the ideal case, in which all points form tight clusters that are well-separated. By well-separated, we mean that if points $v_1$ and $v_2$ lie in different clusters, then their similarity $w_{ij} = 0$. Suppose that the data consists of $k$ clusters $\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k$ with $n_1, n_2, \ldots, n_k$ points, respectively, such that $\bigcup_{i=1}^k \mathcal{V}_i = \mathcal{V}$ and $n = \sum_{i=1}^k n_i$. For the ease of exposition, we also assume that the $v_i$’s are numbered in such a way that points $v_1, v_2, \ldots, v_{n_i}$ are in cluster $\mathcal{V}_1$, the points $v_{n_1+1}, v_{n_1+2}, \ldots, v_{n_1+n_2}$ are in cluster $\mathcal{V}_2$ and so on.

The underlying graph in the ideal case consists of $k$ connected components $\mathcal{G}_1, \mathcal{G}_2, \ldots, \mathcal{G}_k$, where each component $\mathcal{G}_j$ consists of vertices in the corresponding cluster $\mathcal{V}_j$. We represent this ideal graph by $\mathcal{G}^*$, its normalized Laplacian by $L^*$ and its similarity matrix by $W^*$. The $n$-dimensional characteristic vector $\chi_j$ of the $j$th component $\mathcal{G}_j$ is defined as

$$\chi_j(i) = \begin{cases} 1 & \text{if } v_i \in \mathcal{G}_j, \\ 0 & \text{otherwise}. \end{cases}$$

The ideal similarity matrix $W^*$ and, consequently the ideal normalized Laplacian $L^*$ of a graph with $k$ connected components, are both block-diagonal with the $j$th block representing the component $\mathcal{G}_j$, i.e.,

$$W^* = \text{diag}(W_1, W_2, \ldots, W_k),$$

$$L^* = \text{diag}(L_1, L_2, \ldots, L_k).$$

Since $L^*$ is block-diagonal, its spectrum is the union of spectra of $L_1, L_2, \ldots, L_k$. The eigenvalue $\lambda_1 = 0$ of $L^*$ has multiplicity $k$ with $k$ linearly independent normalized eigenvectors $\phi^*_1, \phi^*_2, \ldots, \phi^*_k$. Each of these eigenvectors is given by $\phi^*_j = D^{1/2} \chi_j ||D^{1/2} \chi_j||$. In the following theorem, we show that the iterate sequence $\{x^t\}$ generated by the Mixing Process (3) converges to a linear combination of characteristic vectors $\chi_j$ of $k$ components for an ideal graph. The $\chi_j$’s are also eigenvectors of $M^*$, where $M^* = (1 - \alpha)I - D^{-1}W^*$, corresponding to first $k$ eigenvalues $\mu_1 = \ldots = \mu_k = 1$.

**Theorem 1.** Suppose that we have an ideal dataset which consists of $k$ clusters as defined previously and let $x^0$ be any vector such that each $x^0_i > 0$ and $(x^0)^T 1 = 1$, then

$$\|M^* x^0 - \sum_{i=1}^k c_i \chi_i\| \leq \max_{i, k} |1 - \alpha \lambda_i| t \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}},$$

where $c_i = \frac{\chi_i^T D x^0}{\chi_i^T D x^0}$ and $d_j$ is the degree of the $j$th node.

**Proof.** See [11].

Note that we can always choose $\alpha \in [0, 1]$ such that $\lambda_{k+1} = \arg \max_{i > k} |1 - \alpha \lambda_i|$. Thus the preceding inequality can be written as

$$\|M^* x^0 - \sum_{i=1}^k c_i \chi_i\| \leq (1 - \alpha \lambda_{k+1}) t \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}} \leq e^{-\alpha \lambda_{k+1}} \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}}.$$

For any $\xi > 0$, there exists some $t > 0$ such that

$$e^{-\alpha \lambda_{k+1}} \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}} \leq \xi.$$

Specifically, taking the log and simplifying, we have

$$\frac{1}{\alpha \lambda_{k+1}} \ln \left( \frac{\max_j \sqrt{d_j}}{\xi \min_j \sqrt{d_j}} \right) \leq t.$$

C. The general case

In practice, the graph under consideration may not have $k$ connected components, but rather $k$ nearly connected components i.e., $k$ dense subgraphs sparsely connected by bridges (edges). We can obtain $k$ connected components from such a graph by removing only a small fraction of edges. This means that matrices $W$ and $L$ have non-zero off-diagonal blocks, but both matrices have dominant blocks on the diagonal. The general case is thus a perturbed version of the ideal case.

Let $W = W^* + E$ be the similarity matrix for a dataset $\mathcal{V}$, where $W^*$ is the similarity matrix corresponding to the true clusters (an ideal similarity matrix), which is block-diagonal and symmetric. We obtain $W^*$ by replacing the off-diagonal block elements of $W$ with zeros and adding the sum of the off-diagonal block weights in each row to the diagonal elements. This results in the matrices $W$ and $W^*$ having the same degree matrix $D$. The matrix $E$ is then a
symmetric matrix with row and column sums equal to zero with the $i^{th}$ diagonal entry given by $e_{ii} = -\sum_{j=1}^{n} e_{ij}$. The off-diagonal entries of $E$ are the same as the entries in the off-diagonal blocks of $W$. Using the definition of $L$, one can then show that the following result holds.

**Lemma 1.** If $W = W^* + E$ is a similarity matrix for a dataset $\mathcal{V}$, then the normalized Laplacian of the corresponding graph is $L = L^* - D^{-1/2}ED^{-1/2}$, where $L^*$ is the normalized Laplacian of the ideal graph corresponding to the true clusters.

Since eigenvalues and eigenvectors are continuous functions of entries of a matrix, the eigenvalues $\lambda_i$’s of $L$ can be written as $\lambda_i = \lambda_i^* + \delta_i$, where $\lambda_i^*$ is the eigenvalue of the ideal normalized Laplacian $L^*$, and $\delta_i$ depend continuously on the entries of $E \cong D^{-1/2}ED^{-1/2}$. Similarly the eigenvectors $\phi_i$’s of $L$ can be expressed as $\phi_i = \phi_i^* + \delta_i$, where $\phi_i^*$ is the eigenvector of the ideal normalized Laplacian $L^*$ and $\delta_i$ depend continuously on the entries of $E$. Note that the pair $(\lambda_i, \phi_i)$ is not necessarily an eigenvalue/eigenvector pair of $E$. We assume that $\|E\|$ and consequently $\|E\|$ are small enough so that $|\lambda_i|$, $|\phi_i|$ are also small.

**Theorem 2.** Suppose that we have a dataset which consists of $k$ clusters and let $x^0$ be any vector such that each $x_i^0 > 0$ and $(x^0)^T 1 = 1$, then we have

$$\|M^t x^0 - \sum_{i=1}^{k} c_i x_i \| \leq \left( \sum_{i=1}^{k} (2\|\phi_i\| + \|\phi_i\|^2) \right) \max_{\ell > k} |1 - \alpha \lambda_\ell| \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}}.$$  

where $c_i = (1-\alpha \lambda_i)^t x_i^0 \frac{Dx_i^0}{1^T D x_i}$ and $d_j$ is the degree of the $j^{th}$ node.

**Proof.** See [11].

Note that we can always choose $\alpha$ such that $(1-\alpha \lambda_{k+1}) = \max_{\ell > k} |1 - \alpha \lambda_\ell|$ and the above expression then becomes

$$\|M^t x^0 - \sum_{i=1}^{k} c_i x_i \| \leq \left( \sum_{i=1}^{k} (2\|\phi_i\| + \|\phi_i\|^2) \right) \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}}.$$  

Observe that $\lambda_i = \hat{\lambda}_i$ for $i = 1, \ldots, k$. Thus, assuming that the perturbation is small, the first $k$ eigenvalues of the Laplacian are close to zero. If the eigengap $(\lambda_{k+1} - \lambda_k)$ is large enough, then for some $t > 0$, we will have both $(1-\alpha \lambda_{k+1})^t \geq 1 - \delta$ for a small $\delta > 0$, and $(1-\alpha \lambda_{k+1})^t \leq \epsilon$ for a small $\epsilon > 0$. This results in the effective vanishing of the term $(1-\alpha \lambda_{k+1})^t$ in the above expression after a sufficient number of iterations and $c_i$’s being bounded away from zero. According to Theorem 2, we will then have an approximate linear combination of the $k$ characteristic vectors of the graph i.e., $\|M^t x^0 - \sum_{i=1}^{k} c_i x_i \|$ will be small. Small perturbation assumption also leads to $||\phi_i||$ being relatively small. Note that this eigengap condition is equivalent to Assumption A1 in [3].

It is worth noting that as the number of clusters $k$ grows, it becomes increasingly difficult to discern all the clusters simultaneously from the vector $M^t x^0$ because the perturbation $\delta_i$ will accompany the $k$ eigenvectors in $M^t x^0$. Thus, we devise a recursive bi-partitioning mechanism to find the clusters which divides the data into two clusters at a time.

**D. Clustering algorithm**

Our analysis in the previous section suggests that points in the same cluster mix quickly whereas points in different clusters mix slowly. Simon and Ando’s [12] theory of nearly completely decomposable systems also demonstrates that states in the same subsystem achieve local equilibria long before the system as a whole attains a global equilibrium. Therefore, an efficient clustering algorithm should stop when a local equilibrium is achieved. We can then distinguish the clusters based on mixing of the points. The two clusters in this case correspond to aggregation of elements of $x^t$. Thus a simple search for the largest gap in the sorted $x^t$ can reveal the clusters. This cluster separating gap is directly proportional to $b$, since we initialize $x^0$ by choosing $n$ points uniformly at random from an interval $[0, b]$, and is inversely proportional to the size $n$ of the dataset. Thus we define the gap between two consecutive elements of sorted $x^t$ as:

$$gap(i) = \begin{cases} x_{i+1}^t - x_i^t & \text{if } x_{i+1}^t - x_i^t \geq \frac{b}{2}\pi, \\ 0 & \text{otherwise.} \end{cases}$$  

(6)

In each recursive call, the algorithm terminates upon finding the largest gap and bi-partitions the data based on this gap. If the algorithm fails to find a nonzero gap in a recursive call, then the indexing set of $x$ in this call corresponds a cluster.

In our implementation we start the procedure with an initial tolerance $\epsilon_0$ on mixing of $x^t$. When the tolerance is achieved, we search for a nonzero gap in the vector sort($x^{t+1}$). If a gap is found the dataset is bi-partitioned based on the largest gap. In a recursive fashion, the bi-partitioning procedure is then applied to both resulting partitions. On the other hand, if the gap is not found we decrease the tolerance and reevaluate for a gap after the new tolerance is attained. A cluster is formed if the procedure can not find a gap using either a maximum number of iterations $t_{\max}$ or a minimum tolerance $\epsilon_{\min}$.

**E. Connection to normalized cuts**

Shi and Malik proposed an optimal criteria known as normalized cut (Ncut) to partition a graph [1]. They showed that minimizing the 2-way Ncut to obtain a bipartition of the graph is equivalent to solving the generalized eigenvalue problem $(D - W)y = \lambda y D$ for the second smallest eigenvalue also known as Fiedler value. Since $D$ is invertible, the generalized eigenvalue problem is equivalent to solving $(I - D^{-1}W)y = \lambda y$. Thus, minimizing a 2-way Ncut is approximated by finding the second smallest eigenvector of $I - D^{-1}W$. In this paper, we are looking for a linear
Algorithm 1 Recursive Agent-Based Mixing (RAM)

1: Input: Matrix $M$ and initial tolerance $\epsilon_0$
2: procedure $C = \text{RAM}(M, \epsilon_0)$
3: $n \leftarrow \text{rowsize}(M)$
4: Initialize $x^0_i$ by choosing $n$ points uniformly at random from $[0, b]$.
5: Initialize $\epsilon \leftarrow \epsilon_0$
6: repeat
7: repeat
8: $x^{t+1}_i \leftarrow M x^t_i$, $y^{t+1}_i \leftarrow \|x^{t+1}_i - x^t_i\|$ until $|y^{t+1}_i - y^t_i| \leq \epsilon$
9: $\text{Sort}(x^{t+1}_i)$; find the largest gap using (6).
10: if $\epsilon \leq \epsilon_{\text{min}}$ or $t \geq t_{\text{max}}$ then
11: return
12: end if
13: $\epsilon \leftarrow \epsilon/2$
14: until A nonzero gap is found
15: bi-partition the indexing set of $x^{t+1}_i$ based on largest gap into $i_1$ and $i_2$.
16: $C \leftarrow \text{RAM}(\{i_1, i_1\}, \epsilon_0)$
17: $C \leftarrow \text{RAM}(\{i_2, i_2\}, \epsilon_0)$
18: end procedure
19: Output: Clustering $C$.

TABLE I: Computation time (in seconds) of Algorithm 1 on SBMs with $p = 0.5$ and $q = 0.01$. The time shown is averaged over 50 simulations on different SBMs. All partitions are exactly recovered.

| $n$   | $k$ | RAM   | Normalized Cut |
|-------|-----|-------|----------------|
| 15,000| 5   | 4.19  | 69.46          |
|       | 10  | 3.51  | 121.14         |
|       | 15  | 3.23  | 250.07         |
| 30,000| 5   | 14.47 | Out of Memory  |
|       | 10  | 11.59 | Out of Memory  |
|       | 15  | 10.46 | Out of Memory  |
| 60,000| 5   | 78.21 | Out of Memory  |
|       | 10  | 53.27 | Out of Memory  |
|       | 15  | 45.44 | Out of Memory  |

Fig. 1: Clustering result of Algorithm 1 on synthetic datasets.

Fig. 2: Correct clusterings recovered in 50 runs vs. $q$ (probability of an edge between two points in different clusters). Parameters $n = 15000$ and $p = 0.5$.

We implement all the algorithms in MATLAB 8.4.0 and conduct experiments on a machine with Intel Core i7 3.40GHz CPU and 16GB memory. Additional simulation results can be found in [11].

A. Synthetic datasets

Four two-dimensional synthetic datasets have been used to show accuracy Algorithm 1. Fig 1 depict that our recursive implementation is very accurate in identifying clusters of complex shapes and different sizes.

B. Scalability

We apply Algorithm 1 to stochastic block model (SBM) graphs to illustrate its scalability to large datasets with many clusters. We also compare the runtime with normalized cut algorithm [1]. In the basic form, a SBM with same size blocks (clusters) is defined by four parameters; $n$, the number of vertices; $k$, the number of blocks (clusters); $p$, the probability of an edge between two points in the same cluster and $q$, the probability of an edge between two points in different clusters. Running time of the algorithm on various size SBMs is shown in Table I. Runtimes shown are average times for 50 different SBMs. Observe that RAM algorithm is significantly faster than normalized cut. It also consumes less memory as compared to normalized cut. For each model, the RAM algorithm recovers all the clusters exactly in each run with $p = 0.5$ and $q = 0.01$. Each node shares roughly $pn/k$ edges within the cluster and $q(n - n/k)$ edges across the cluster. For example with $n = 30,000$ and $k = 10$, a node approximately has an edge with 1500 nodes in its cluster and 270 edges with nodes in other clusters. Total edges in this graph are roughly $0.5(1770 \times 30,000) = 26.55$ million. We also show the ability of the algorithm to recover correct clusters as we increase the number of edges across
TABLE II: Comparison of Average NMI with standard error and runtime in seconds (in parenthesis) over 50 simulations on real datasets. \( k \) denotes the number of clusters. For each \( k \), 50 runs are conducted on randomly chosen clusters (except for \( k = 10 \) for USPS and \( k = 20 \) for COIL20). N-cut represents the normalized cut algorithm [1]. FASC 1 and 2 are implemented with 10% and 5% representative points respectively. Runtime of FASC is significantly more than our algorithm because it uses k-means to get the representative points which has linear time complexity in terms of dimension of the dataset. NYSTROM 1 and 2 are approximations of Normalized cut algorithm as defined in [8].

FASC 1 and 2 are approximations of Normalized cut algorithm as defined in [8].

For both dataset we use \( q \) as normalized cut algorithm.

C. Real datasets

We empirically compare the accuracy and speed of our RAM algorithm with normalized cut algorithm [1], fast approximate spectral clustering [8] and Nyström method [6] on two real datasets. The USPS dataset has 7291 instances and length of the feature vector is 256 [14]. It has a total of 10 clusters. The COIL20 dataset consists of 1140 examples and has 1024 features with 20 clusters [15]. We use a \( p \)-nearest neighbor graph to construct the similarity matrix. For both dataset we use \( p = 4 \). \( \epsilon_0 \) is set to \( 10^{-3} \) and \( 10^{-2} \) for USPS and COIL20 datasets respectively. We compare normalized mutual information and computation time of RAM algorithm with other algorithms. As demonstrated earlier RAM algorithm has the same accuracy as normalized cut algorithm. Our algorithm does not sacrifice accuracy as opposed to FASC and Nyström method which also claim to improve the speed of spectral clustering. For large datasets RAM is faster than both FASC and Nyström and does not compromise on the accuracy.

VI. CONCLUSIONS

We have proposed a fast spectral clustering algorithm based on a mixing process, which does not explicitly compute the eigenvectors of a similarity matrix. Our algorithms are simple to implement and computationally efficient. We have demonstrated the scalability and accuracy of the algorithm by applying it to large stochastic block models and have also shown that our algorithm has the same accuracy as normalized cut algorithm.

REFERENCES

[1] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[2] Chris HQ Ding, Xiaofeng He, Hongyuan Zha, Ming Gu, and Horst D Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of the IEEE International Conference on Data Mining*, pages 107–114, 2001.

[3] Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in Neural Information Processing Systems 14*, pages 849–856, 2002.

[4] Francis R. Bach and Michael I. Jordan. Learning spectral clustering. In *Advances in Neural Information Processing Systems 16*, pages 305–312, 2004.

[5] Peter Arbenz and Daniel Kressner. Lecture notes on solving large scale eigenvalue problems. In *Parallel Computing: Theory and Practice*, pages 201–220. CRC Press, 2004.

[6] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the Nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225, 2004.

[7] Tomoya Sakai and Atsushi Imiya. Fast spectral clustering with random projection and sampling. In *Machine Learning and Data Mining in Pattern Recognition*, volume 5632 of *Lecture Notes in Computer Science*, pages 372–384, 2009.

[8] Donghui Yan, Ling Huang, and Michael I. Jordan. Fast approximate spectral clustering. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 907–916, 2009.

[9] Wen-Yen Chen, Yangqiu Song, Hongjie Bai, Chih-Jen Lin, and Edward Y. Chang. Parallel spectral clustering in distributed systems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(3):568–586, 2011.

[10] Fan RK Chung. *Spectral graph theory*, volume 92. American Mathematical Soc., 1997.

[11] Shahzad Bhatti, Carolyn Beck, and Angelia Nedic. Data clustering and graph partitioning via simulated mixing. *arXiv preprint arXiv:1603.04918*, 2016.

[12] Herbert A Simon and Albert Ando. Aggregation of variables in dynamic systems. *Econometrica: Journal of the Econometric Society*, pages 111–138, 1961.

[13] Alexander Strehl and Jiydeep Ghosh. Cluster ensembles: A knowledge reuse framework for combining multiple partitions. *The Journal of Machine Learning Research*, 3:583–617, 2003.

[14] Yann Le Cun, Bernhard E Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne E Hubbard, and Lawrence D Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in Neural Information Processing Systems 2*, pages 396–404, 1990.

[15] Sameer A Nene, Shree K Nayar, and Hiroshi Murase. Columbia object image library (coil-20). Technical report, CUCS-005-96, 1996.