Data Clustering and Graph Partitioning via Simulated Mixing

Shahzad Bhatti* Carolyn Beck* Angelia Nedić*

Abstract

Spectral clustering approaches have led to well-accepted algorithms for finding accurate clusters in a given dataset. However, their application to large-scale datasets has been hindered by computational complexity of eigenvalue decompositions. Several algorithms have been proposed in the recent past to accelerate spectral clustering, however they compromise on the accuracy of the spectral clustering to achieve faster speed. In this paper, we propose a novel spectral clustering algorithm based on a mixing process on a graph. Unlike the existing spectral clustering algorithms, our algorithm does not require computing eigenvectors. Specifically, it finds the equivalent of a linear combination of eigenvectors of the normalized similarity matrix weighted with corresponding eigenvalues. This linear combination is then used to partition the dataset into meaningful clusters. Simulations on real datasets show that partitioning datasets based on such linear combinations of eigenvectors achieves better accuracy than standard spectral clustering methods as the number of clusters increase. Our algorithm can easily be implemented in a distributed setting.

1 Introduction

Data clustering is a fundamental problem in pattern recognition, data mining, computer vision, machine learning, bioinformatics and several other related disciplines. It has a long history and researchers in various fields have proposed numerous solutions. Several spectral clustering algorithms have been proposed [1, 2, 3, 4], which have enjoyed great success and have been widely used to cluster data. However, spectral clustering does not scale well to large-scale problems due to its considerable computational cost. In general, spectral clustering algorithms seek 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 of $O(n^3)$ [5], which is significant for large-scale problems.

In the past few years, efforts have been focused towards addressing scalability of spectral clustering. A natural way to achieve scalability is to perform spectral clustering on a sample of the given

*ISE & CSL, University of Illinois at Urbana Champaign, IL Email:{bhatti2, beck3, angelia}@illinois.edu

dataset and then generalize the result to the rest of data. For example, Fowlkes et al. [6] 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. They also reduce the dimension of the dataset using random projections. Another approach proposed by Yan et al. [8] works by first determining a smaller set of representative points using k-means (each centroid is a representative point) and then performing spectral clustering on the representative points. Finally, 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 on distributed computers.

These methods sacrifice the accuracy of spectral clustering to achieve fast implementation. In this paper, we perform spectral clustering without explicitly calculating eigenvectors. Rather we compute a linear combination of the right-eigenvectors weighted with corresponding eigenvalues. Moreover, unlike many traditional algorithms, our algorithm does not require a predefined number of clusters, k, as input. This algorithm can automatically detect and adapt to any number of clusters, based on a preselected tolerance. We apply our algorithm to large size stochastic block models to illustrate the scalability and demonstrate that it can handle large datasets where traditional spectral algorithms result in memory errors. We compare the accuracy and speed of our algorithm to the normalized cut algorithm [1] on real datasets 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. Throughout this paper 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 $\mathcal{G}$ is reserved to denote graphs. The norm $\|\cdot\|$ denotes $\|\cdot\|_2$ for vectors and for matrices it denotes spectral norm.

2 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. We will often use a short-hand notation i to denote the vector $\mathbf{v}_i$. 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 $s(\mathbf{v}_i, \mathbf{v}_j)$ often abbreviated by $s(i, j)$, where usually it is assumed that the function s is symmetric. Also, the similarity function s is non-negative if $\mathbf{v}_i \neq \mathbf{v}_j$

and is equal to zero if $\mathbf{v}_i = \mathbf{v}_j$. 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 $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; $\mathcal{V}$ 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 $\mathbf{v}_i$ represents a vertex/node in the graph. Two vertices i and j are connected by an edge if their similarity $s(i, j)$ is positive and the edge weight is given by $s(i, j)$. 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. Some commonly used similarity graphs for spectral clustering are noted below.

a) **Gaussian similarity graphs:** Gaussian similarity graphs are based on the distance between points. Typically every pair of vertices is connected by an edge and the edge weight is determined by the Gaussian function (radial basis function) $s(i, j) = exp(-\|\mathbf{v}_i - \mathbf{v}_j\|^2/2\sigma^2)$, where parameter σ controls how quickly the similarity fades away as the distance between the points increases. It is often helpful to remove the edges for edge weights below a certain threshold (say δ) to sparsify the graph. Thus the similarity function for this type of graph is

$$s(i,j) = \begin{cases} e^{-\frac{\|\mathbf{v}_i - \mathbf{v}_j\|^2}{2\sigma^2}} & \text{if } \|\mathbf{v}_i - \mathbf{v}_j\| > \delta, \\ 0 & \text{otherwise.} \end{cases}$$

b) **p-nearest neighbor graphs:** As the name suggests each vertex $\mathbf{v}_i$ is connected to its p nearest neighbors, where the nearness between i and j is measured by the distance $\|\mathbf{v}_i - \mathbf{v}_j\|$. 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 $\mathbf{v}_j$ is among the p nearest neighbors of $\mathbf{v}_i$ then it is not necessary for $\mathbf{v}_i$ to be among the p nearest neighbors of $\mathbf{v}_j$. We make the similarity measure symmetric by placing an edge between two vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ if either $\mathbf{v}_i$ is among the p nearest neighbors of $\mathbf{v}_j$ or $\mathbf{v}_j$ is among the p nearest neighbors of $\mathbf{v}_i$, that is

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

c) **ϵ-neighborhood graphs:** In an ϵ-neighborhood graph, we connect two vertices $\mathbf{v}_i$ and $\mathbf{v}_j$ if

the distance $\|\mathbf{v}_i - \mathbf{v}_j\|$ is less than ϵ giving us the following similarity function.

$$s(i,j) = \begin{cases} 1 & \text{if } \|\mathbf{v}_i - \mathbf{v}_j\| \leq \epsilon, \\ 0 & \text{otherwise.} \end{cases}$$

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 $\overline{W} = D^{-1}W$.

3 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 i moves towards (mixes with) vertex j by a distance proportional to the similarity $s(i,j)$. Thus the larger the similarity $s(i,j)$, the larger the distance vertices i and j move towards each other i.e., the greater the mixing. Moreover, a point $\mathbf{v}_i$ 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 describe the above idea more precisely, consider the following model, where each point $\mathbf{v}_i$ moves according to the following equation, starting at its original position at time $t = 0$:

$$\begin{aligned} \mathbf{v}_i^{t+1} &= \mathbf{v}_i^t + \alpha \sum_{j=1}^{n} \overline{w}_{ij}(\mathbf{v}_j^t - \mathbf{v}_i^t) \\ &= (1-\alpha)\mathbf{v}_i^t + \alpha \sum_{j=1}^{n} \overline{w}_{ij}\, \mathbf{v}_j^t. \end{aligned} \tag{1}$$

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 α bounded away from 1. We can use $\alpha = 1$ for graphs without a bipartite component. Assuming each point $\mathbf{v}_i$ is a row vector, we

express equation (1) in a matrix form:

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

The matrix V^t is a $n \times d$ matrix with row i representing the position of point $\mathbf{v}_i$ at time t. I is a $n \times n$ identity matrix, and we define $M = (1-\alpha)I + \alpha\overline{W}$. 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 equation (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 equation (2) can be constructed to determine the clusters, as given in the following algorithm.

Algorithm 1 Point-Based Resource Diffusion (PRD)

1: **Input:** Set of data points $\mathcal{V}$, number of clusters k
2: Represent the data points in the matrix V with point $\mathbf{v}_i$ being the ith row.
3: Compute $M = (1-\alpha)I + \overline{W}$.
4: **Loop over** t
5: $\quad V^{t+1} \leftarrow MV^t$
6: **until** Stopping criteria is met.
7: Find k clusters from rows of V^{t+1} using k-means algorithm.
8: **Output:** Clustering obtained in the final iteration.

Algorithm 1 has two limitations: a) it does not scale well with the dimension d of the data points, because the number of computations in each iteration is $O(n^2 d)$, and b) it fails to identify clusters contained within other clusters. For example, 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. Hence it becomes impossible to discern these clusters using the k-means algorithm in Step 7. To overcome these limitations, we associate an *agent* x_i to each point $\mathbf{v}_i$ and carry out calculations in the *agent space*. Agents are generated by choosing n points uniformly at random from a bounded interval $[0, b]$[1]. We rewrite equation (2) in the agent space as:

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

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.

[1]Note, b 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.

4 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 $\mathbf{x}^{t+1}$ 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 $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

The Mixing Process shares a resemblance to the power iteration. Unlike the power iteration, we here use the mixing process to discover strongly coupled components of M, which translate to data clusters.

4.1 Properties of the matrix M

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

$$\begin{aligned} M &= (1-\alpha)I + \alpha\overline{W} \\ &= I - \alpha(I - D^{-1}W) \\ &= I - \alpha D^{-1/2} L D^{1/2} \\ &= D^{-1/2}(I - \alpha L) D^{1/2}, \end{aligned} \tag{4}$$

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

$$\begin{aligned} M D^{-1/2}\boldsymbol{\phi}_i &= (I - \alpha D^{-1/2} L D^{1/2}) D^{-1/2}\boldsymbol{\phi}_i \\ &= (1 - \alpha\lambda_i) D^{-1/2}\boldsymbol{\phi}_i. \end{aligned}$$

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 $\mathbb{R}^n$ and we can express L in the following

form:

$$L = \sum_{i=i}^{n} \lambda_i \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T. \tag{5}$$

Using (4) and (5), we obtain

$$\begin{aligned} M^t &= \left(D^{-1/2}(I - \alpha L)D^{1/2}\right)^t \\ &= D^{-1/2}\left(\sum_{i=i}^{n}(1 - \alpha\lambda_i)^t \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T\right) D^{1/2}. \end{aligned} \tag{6}$$

We will exploit this relationship in our proofs later.

4.2 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 $\mathbf{v}_i$ and $\mathbf{v}_j$ 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 $\cup_{i=1}^{k}\mathcal{V}_i = \mathcal{V}$ and $n = \sum_{i=1}^{k} n_i$. For the ease of exposition, we also assume that the $\mathbf{v}_i$'s are numbered in such a way that points $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_{n_1}$ are in cluster $\mathcal{V}_1$, the points $\mathbf{v}_{n_1+1}, \mathbf{v}_{n_1+2}, \ldots, \mathbf{v}_{n_1+n_2}$ 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 $\boldsymbol{\chi}_j$ of the jth component $\mathcal{G}_j$ is defined as

$$\boldsymbol{\chi}_j(i) = \begin{cases} 1 & \text{if } \mathbf{v}_i \in \mathcal{G}_j, \\ 0 & \text{otherwise.} \end{cases}$$

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.,

$$\begin{aligned} W^* &= diag(W_1, W_2, \ldots, W_k), \\ L^* &= diag(L_1, L_2, \ldots, L_k). \end{aligned}$$

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 $\boldsymbol{\phi}_1^*$, $\boldsymbol{\phi}_2^*$, $\ldots$, $\boldsymbol{\phi}_k^*$. Each of these eigenvectors is given by $\boldsymbol{\phi}_j^* = D^{1/2}\boldsymbol{\chi}_j / \|D^{1/2}\boldsymbol{\chi}_j\|$. In the following theorem, we prove that if we have an ideal graph then the iterate sequence $\{\mathbf{x}^t\}$ generated by the Mixing Process (3)

converges to a linear combination of characteristic vectors $\boldsymbol{\chi}_j$'s of the k components of the graph. The $\boldsymbol{\chi}_j$'s are also eigenvectors of M^*, where $M^* = (1-\alpha)I - D^{-1}W^*$, corresponding to first k eigenvalues $\mu_1 = \mu_2 = \ldots = \mu_k = 1$.

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

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

where $c_i = \frac{\boldsymbol{\chi}_i^T D\mathbf{x}^0}{\mathbf{1}^T D\boldsymbol{\chi}_i}$ and d_j is the degree of the j^{th} node.

Proof. We begin by noting that the iterative process (3) is equivalent to $\mathbf{x}^t = M^t\mathbf{x}^0$. Using (6), we have

$$\|M^{*t}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| = \|D^{-1/2}\left(\sum_{i=1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\right)D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\|.$$

Separating the first k terms in the sum and using the fact that eigenvector $\boldsymbol{\phi}_i^* = \frac{D^{1/2}\boldsymbol{\chi}_i}{\|D^{1/2}\boldsymbol{\chi}_i\|}$, for $i = 1, \ldots, k$, the above equation can be simplified as

$$\begin{aligned}
\|M^{*t}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &= \|D^{-1/2}\left(\sum_{i=1}^{k}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\right)D^{1/2}\mathbf{x}^0 \\
&\quad + D^{-1/2}\left(\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\right)D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| \\
&= \|\sum_{i=1}^{k} D^{-1/2}\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}D^{1/2}\mathbf{x}^0 + D^{-1/2}\left(\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\right)D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| \\
&= \|\sum_{i=1}^{k} D^{-1/2}\frac{D^{1/2}\boldsymbol{\chi}_i}{\|D^{1/2}\boldsymbol{\chi}_i\|}\frac{(D^{1/2}\boldsymbol{\chi}_i)^T}{\|D^{1/2}\boldsymbol{\chi}_i\|}D^{1/2}\mathbf{x}^0 \\
&\quad + D^{-1/2}\left(\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\right)D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\|.
\end{aligned}$$

We can simplify the first term in the norm as

$$\begin{aligned}
D^{-1/2}\frac{D^{1/2}\boldsymbol{\chi}_i}{\|D^{1/2}\boldsymbol{\chi}_i\|}\frac{(D^{1/2}\boldsymbol{\chi}_i)^T}{\|D^{1/2}\boldsymbol{\chi}_i\|}D^{1/2}\mathbf{x}^0 &= \frac{\boldsymbol{\chi}_i\ \boldsymbol{\chi}_i^T D^{1/2}D^{1/2}\mathbf{x}^0}{\|D^{1/2}\boldsymbol{\chi}_i\|^2} \\
&= \frac{\boldsymbol{\chi}_i^T D\mathbf{x}^0}{\|D^{1/2}\boldsymbol{\chi}_i\|^2}\boldsymbol{\chi}_i,
\end{aligned}$$

where the term $\|D^{1/2}\boldsymbol{\chi}_i\|^2$ is equal to the sum of the degrees of vertices in component $\mathcal{G}_i$ (commonly called volume of $\mathcal{G}_i$) which can also be expressed by $\mathbf{1}^T D\boldsymbol{\chi}_i$, to have

$$\begin{aligned}\|M^{*t}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &= \|\sum_{i=1}^{k} \frac{\boldsymbol{\chi}_i^T D\mathbf{x}^0}{\mathbf{1}^T D\boldsymbol{\chi}_i}\boldsymbol{\chi}_i + D^{-1/2}\left(\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\right)D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| \\ &= \|D^{-1/2}\left(\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\right)D^{1/2}\mathbf{x}^0\|.\end{aligned}$$

In the last equation we have used $c_i = \frac{\boldsymbol{\chi}_i^T D\mathbf{x}^0}{\mathbf{1}^T D\boldsymbol{\chi}_i}$. Now using the properties of the norm, we can separate the terms, giving us

$$\begin{aligned}\|M^{*t}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &\leq \|D^{-1/2}\|\ \|\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T}\|\ \|D^{1/2}\|\ \|\mathbf{x}^0\| \\ &\leq \max_{i>k}|1-\alpha\lambda_i|^t\frac{\max_j\sqrt{d_j}}{\min_j\sqrt{d_j}}, \qquad \text{since } \|\mathbf{x}^0\| \leq 1.\end{aligned}$$

□

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

$$\begin{aligned}\|M^{*t}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &\leq (1-\alpha\lambda_{k+1})^t\frac{\max_j\sqrt{d_j}}{\min_j\sqrt{d_j}} \\ &\leq e^{-\alpha t\lambda_{k+1}}\frac{\max_j\sqrt{d_j}}{\min_j\sqrt{d_j}}.\end{aligned}$$

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

$$e^{-\alpha t\lambda_{k+1}}\frac{\max_j\sqrt{d_j}}{\min_j\sqrt{d_j}} \leq \xi. \tag{7}$$

Specifically, taking the log and simplifying, we have

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

4.3 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. For example, for

$$W = \left[\begin{array}{ccc:ccc} 0 & 20 & 50 & 1 & 2 & 1 \\ 20 & 0 & 30 & 0 & 1 & 1 \\ 50 & 30 & 0 & 1 & 0 & 1 \\ \hdashline 1 & 0 & 1 & 0 & 25 & 40 \\ 2 & 1 & 0 & 25 & 0 & 30 \\ 1 & 1 & 1 & 40 & 30 & 0 \end{array}\right],$$

the matrices W^* and E satisfying the above constraints are given by

$$W^* = \left[\begin{array}{ccc:ccc} 4 & 20 & 50 & 0 & 0 & 0 \\ 20 & 2 & 30 & 0 & 0 & 0 \\ 50 & 30 & 2 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 0 & 2 & 25 & 40 \\ 0 & 0 & 0 & 25 & 3 & 30 \\ 0 & 0 & 0 & 40 & 30 & 3 \end{array}\right], \text{ and } E = \left[\begin{array}{ccc:ccc} -4 & 0 & 0 & 1 & 2 & 1 \\ 0 & -2 & 0 & 0 & 1 & 1 \\ 0 & 0 & -2 & 1 & 0 & 1 \\ \hdashline 1 & 0 & 1 & -2 & 0 & 0 \\ 2 & 1 & 0 & 0 & -3 & 0 \\ 1 & 1 & 1 & 0 & 0 & -3 \end{array}\right].$$

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 λ_i's of L can be written as

$$\lambda_i = \lambda_i^* + \tilde{\lambda}_i,$$

where λ_i^* is the eigenvalue of the ideal normalized Laplacian L^*, and $\tilde{\lambda}_i$ depend continuously on

the entries of $\bar{E} \triangleq D^{-1/2}ED^{-1/2}$. Similarly the eigenvectors $\boldsymbol{\phi}_i$'s of L can be expressed as

$$\boldsymbol{\phi}_i = \boldsymbol{\phi}_i^* + \tilde{\boldsymbol{\phi}}_i,$$

where $\boldsymbol{\phi}_i^*$ is the eigenvector of the ideal normalized Laplacian L^* and $\tilde{\boldsymbol{\phi}}_i$ depend continuously on the entries of $\bar{E}$. Note that the pair $(\tilde{\lambda}_i, \tilde{\boldsymbol{\phi}}_i)$ is not necessarily an eigenvalue/eigenvector pair of $\bar{E}$. We assume that $\|E\|$ and consequently $\|\bar{E}\|$ are small enough so that $|\tilde{\lambda}_i|$ and $\|\tilde{\boldsymbol{\phi}}_i\|$ are also small.

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

$$\|M^t\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| \leq \left(\sum_{i=1}^{k}\left(2\|\tilde{\boldsymbol{\phi}}_i\| + \|\tilde{\boldsymbol{\phi}}_i\|^2\right) + \max_{\ell>k}|1-\alpha\lambda_\ell|^t\right)\frac{\max_j\sqrt{d_j}}{\min_j\sqrt{d_j}},$$

where $c_i = (1-\alpha\tilde{\lambda}_i)^t\frac{\boldsymbol{\chi}_i^T D\mathbf{x}^0}{\mathbf{1}^T D\boldsymbol{\chi}_i}$ and d_j is the degree of the j^{th} node.

Proof. Using (6) and separating the first k terms, we get

$$\begin{aligned}
\|M^t\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &= \|D^{-1/2}\sum_{i=1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| \\
&= \|D^{-1/2}\sum_{i=1}^{k}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T D^{1/2}\mathbf{x}^0 \\
&\quad + D^{-1/2}\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\|.
\end{aligned}$$

Substituting $\boldsymbol{\phi}_i = \boldsymbol{\phi}_i^* + \tilde{\boldsymbol{\phi}}_i$ and $\lambda_i = \lambda_i^* + \tilde{\lambda}_i$ in the above equation and using the fact that $\lambda_i^* = 0$ for $i = 1, \ldots, k$, we have

$$\begin{aligned}
\|M^t\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &= \|D^{-1/2}\sum_{i=1}^{k}(1-\alpha\tilde{\lambda}_i)^t\left(\boldsymbol{\phi}_i^*\boldsymbol{\phi}_i^{*T} + \boldsymbol{\phi}_i^*\tilde{\boldsymbol{\phi}}_i^T + \tilde{\boldsymbol{\phi}}_i\boldsymbol{\phi}_i^{*T} + \tilde{\boldsymbol{\phi}}_i\tilde{\boldsymbol{\phi}}_i^T\right)D^{1/2}\mathbf{x}^0 \\
&\quad + D^{-1/2}\sum_{i=k+1}^{n}(1-\alpha\lambda_i)^t\boldsymbol{\phi}_i\boldsymbol{\phi}_i^T D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| \\
&= \|\sum_{i=1}^{k}(1-\alpha\tilde{\lambda}_i)^t D^{-1/2}\frac{D^{1/2}\boldsymbol{\chi}_i(D^{1/2}\boldsymbol{\chi}_i)^T}{\|D^{1/2}\boldsymbol{\chi}_i\|^2}D^{1/2}\mathbf{x}^0 \\
&\quad + \sum_{i=1}^{k}(1-\alpha\tilde{\lambda}_i)^t D^{-1/2}\left(\boldsymbol{\phi}_i^*\tilde{\boldsymbol{\phi}}_i^T + \tilde{\boldsymbol{\phi}}_i\boldsymbol{\phi}_i^{*T} + \tilde{\boldsymbol{\phi}}_i\tilde{\boldsymbol{\phi}}_i^T\right)D^{1/2}\mathbf{x}^0
\end{aligned}$$

$$+ D^{-1/2} \sum_{i=k+1}^{n} (1-\alpha\lambda_i)^t \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T D^{1/2}\mathbf{x}^0 - \sum_{i=1}^{k} c_i \boldsymbol{\chi}_i \|, \tag{8}$$

where

$$\begin{aligned}(1-\alpha\tilde{\lambda}_i)^t D^{-1/2} \frac{D^{1/2}\boldsymbol{\chi}_i (D^{1/2}\boldsymbol{\chi}_i)^T}{\|D^{1/2}\boldsymbol{\chi}_i\|^2} D^{1/2}\mathbf{x}^0 &= (1-\alpha\tilde{\lambda}_i)^t \frac{\boldsymbol{\chi}_i \ \boldsymbol{\chi}_i^T D^{1/2} D^{1/2}\mathbf{x}^0}{\|D^{1/2}\boldsymbol{\chi}_i\|^2} \\ &= (1-\alpha\tilde{\lambda}_i)^t \frac{\boldsymbol{\chi}_i^T D\mathbf{x}^0}{\mathbf{1}^T D\boldsymbol{\chi}_i} \boldsymbol{\chi}_i \\ &= c_i \boldsymbol{\chi}_i.\end{aligned}$$

Substituting the above expression in (8) and using the triangle inequality, we have

$$\begin{aligned}\|M^t\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &= \| \sum_{i=1}^{k} (1-\alpha\tilde{\lambda}_i)^t D^{-1/2} \left(\boldsymbol{\phi}_i^* \tilde{\boldsymbol{\phi}}_i^T + \tilde{\boldsymbol{\phi}}_i \boldsymbol{\phi}_i^{*T} + \tilde{\boldsymbol{\phi}}_i \tilde{\boldsymbol{\phi}}_i^T \right) D^{1/2}\mathbf{x}^0 \\ &\quad + D^{-1/2} \sum_{i=k+1}^{n} (1-\alpha\lambda_i)^t \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T D^{1/2}\mathbf{x}^0 \| \\ &\leq \| \sum_{i=1}^{k} (1-\alpha\tilde{\lambda}_i)^t D^{-1/2} \left(\boldsymbol{\phi}_i^* \tilde{\boldsymbol{\phi}}_i^T + \tilde{\boldsymbol{\phi}}_i \boldsymbol{\phi}_i^{*T} + \tilde{\boldsymbol{\phi}}_i \tilde{\boldsymbol{\phi}}_i^T \right) D^{1/2}\mathbf{x}^0 \| \\ &\quad + \| D^{-1/2} \sum_{i=k+1}^{n} (1-\alpha\lambda_i)^t \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T D^{1/2}\mathbf{x}^0 \| \\ &\leq \sum_{i=1}^{k} |1-\alpha\tilde{\lambda}_i|^t \|D^{-1/2}\| \left(\|\boldsymbol{\phi}_i^* \tilde{\boldsymbol{\phi}}_i^T\| + \|\tilde{\boldsymbol{\phi}}_i \boldsymbol{\phi}_i^{*T}\| + \|\tilde{\boldsymbol{\phi}}_i \tilde{\boldsymbol{\phi}}_i^T\| \right) \|D^{1/2}\| \ \|\mathbf{x}^0\| \\ &\quad + \|D^{-1/2}\| \ \| \sum_{i=k+1}^{n} (1-\alpha\lambda_i)^t \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \| \ \|D^{1/2}\| \ \|\mathbf{x}^0\|. \end{aligned} \tag{9}$$

Since $\|\mathbf{x}^0\| \leq 1,\ \|\boldsymbol{\phi}_i^*\| = 1,\ |1-\alpha\tilde{\lambda}_i| \leq 1$, we can further simplify inequality (9) as

$$\begin{aligned}\|M^t\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| &\leq \sum_{i=1}^{k} |1-\alpha\tilde{\lambda}_i|^t \|D^{-1/2}\| \left(2\|\tilde{\boldsymbol{\phi}}_i\| + \|\tilde{\boldsymbol{\phi}}_i\|^2 \right) \|D^{1/2}\| \\ &\quad + \|D^{-1/2}\| \ \| \sum_{i=k+1}^{n} (1-\alpha\lambda_i)^t \boldsymbol{\phi}_i \boldsymbol{\phi}_i^T \| \ \|D^{1/2}\| \\ &\leq \sum_{i=1}^{k} \left(2\|\tilde{\boldsymbol{\phi}}_i\| + \|\tilde{\boldsymbol{\phi}}_i\|^2 \right) \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}} + \max_{\ell>k} |1-\alpha\lambda_\ell|^t \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}} \\ &= \left(\sum_{i=1}^{k} \left(2\|\tilde{\boldsymbol{\phi}}_i\| + \|\tilde{\boldsymbol{\phi}}_i\|^2 \right) + \max_{\ell>k} |1-\alpha\lambda_\ell|^t \right) \frac{\max_j \sqrt{d_j}}{\min_j \sqrt{d_j}}. \end{aligned} \tag{10}$$

□

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

$$\|M^t\mathbf{x}^0 - \sum_{i=1}^{k} c_i\boldsymbol{\chi}_i\| \leq \left(\sum_{i=1}^{k}\left(2\|\tilde{\boldsymbol{\phi}}_i\| + \|\tilde{\boldsymbol{\phi}}_i\|^2\right) + (1-\alpha\lambda_{k+1})^t\right)\frac{\max_j\sqrt{d_j}}{\min_j\sqrt{d_j}}.$$

Observe that $\lambda_i = \tilde{\lambda}_i$ 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 $(1-\alpha\lambda_k) = (1-\alpha\tilde{\lambda}_k)^t \geq 1-\delta$ 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., $\|M^t\mathbf{x}^0 - \sum_{i=1}^{k} c_i\chi_i\|$ will be small. Small perturbation assumption also leads to $\|\tilde{\boldsymbol{\phi}}_i\|$ 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 distinguish the clusters from the vector $M^t\mathbf{x}^0$ using the classical k-means algorithm because the perturbation $\tilde{\boldsymbol{\phi}}_i$ will accompany the k eigenvectors in $M^t\mathbf{x}^0$. Thus, we devise a recursive bi-partitioning mechanism to find the clusters.

4.4 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 [11] 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 $\mathbf{x}^t$. Thus a simple search for the largest *gap* in the sorted $\mathbf{x}^t$ can reveal the clusters. This cluster separating *gap* is directly proportional to b, since we initialize $\mathbf{x}^0$ by choosing n points uniformly at random from an interval $[0, b]$. Furthermore, it is inversely proportional to the size n of the dataset. Thus we define the *gap* between two consecutive elements of sorted $\mathbf{x}^t$ as:

$$gap(i) = \begin{cases} x_{i+1}^t - x_i^t & \text{if } \; x_{i+1}^t - x_i^t \geq \frac{b}{2n}, \\ 0 & \text{otherwise.} \end{cases} \tag{11}$$

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 $\mathbf{x}$ in this call corresponds a cluster. This leads us to Algorithm 2 (RARD - Theoretical).

Algorithm 2 Recursive Agent-Based Resource Diffusion (RARD) - Theoretical

Input: Matrix $M = (1-\alpha)I + \alpha D^{-1}W$ and a tolerance ϵ
procedure $C = \text{RARD}(M,\ \epsilon)$
    $n \leftarrow rowsize(M)$
    Initialized $\mathbf{x}^0$ by choosing n points uniformly at random from $[0, b]$.
    repeat
        $\mathbf{x}^{t+1} \leftarrow M\mathbf{x}^t$
    until $(1-\alpha\lambda_{k+1})^t \leq \epsilon$
    Sort($\mathbf{x}^{t+1}$); find the largest *gap* using Equation (11).
    if *gap* is not found **then**
        return
    end if
    bi-partition the indexing set of $\mathbf{x}^{t+1}$ based on largest *gap* into i_1 and i_2.
    $C \leftarrow \text{RARD}(M(i_1, i_1),\ \epsilon), \qquad C \leftarrow \text{RARD}(M(i_2, i_2),\ \epsilon)$
end procedure
Output: Clustering C.

In practice, we do not know the eigenvalues of M. Thus, in our implementation we start the procedure with an initial tolerance ϵ_0 on mixing of $\mathbf{x}^t$. When the tolerance is achieved, we search for a nonzero *gap* in the vector sort($\mathbf{x}^{t+1}$). 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 $\epsilon_{\min}$. This algorithm is Algorithm 3 (RARD - Implemented). Since our algorithm finds clusters in a recursive fashion by bi-partitioning the dataset in each recursive step, Theorem 2 can be reduced to the following corollary.

Corollary 1. *Suppose the dataset contains 2 clusters and let* $\mathbf{x}^0$ *be any vector such that each* $x_i^0 > 0$ *and* $(\mathbf{x}^0)^T\mathbf{1} = 1$*, then we have*

$$\|M^t\mathbf{x}^0 - \sum_{i=1}^{2} c_i\boldsymbol{\chi}_i\| \leq \left(\sum_{i=1}^{2} 2\|\tilde{\boldsymbol{\phi}}_i\| + \|\tilde{\boldsymbol{\phi}}_i\|^2 + (1-\alpha\lambda_3)^t\right)\frac{\max_j\sqrt{d_j}}{\min_j\sqrt{d_j}},$$

where $c_i = (1-\alpha\tilde{\lambda}_i)^t\frac{\boldsymbol{\chi}_i^T D\mathbf{x}^0}{\mathbf{1}^T D\boldsymbol{\chi}_i}$ *and* d_j *is the degree of the* j^{th} *node.*

Algorithm 3 Recursive Agent-Based Resource Diffusion (RARD) - Implemented

Input: Matrix $M = (1-\alpha)I + \alpha D^{-1}W$ and initial tolerance ϵ_0
procedure $C = \text{RARD}(M,\ \epsilon_0)$
    $n \leftarrow rowsize(M)$
    Initialized $\mathbf{x}^0$ by choosing n points uniformly at random from $[0, b]$.
    Initialize $\epsilon \leftarrow \epsilon_0$
    repeat
        repeat
            $\mathbf{x}^{t+1} \leftarrow M\mathbf{x}^t, \qquad y^{t+1} \leftarrow \|\mathbf{x}^{t+1} - \mathbf{x}^t\|$
        until $|y^{t+1} - y^t| \leq \epsilon$
        Sort($\mathbf{x}^{t+1}$); find the largest *gap* using Equation (11).
        if $\epsilon \leq \epsilon_{\min}$ **or** $t \geq t_{\max}$ **then**
            return
        end if
        $\epsilon \leftarrow \epsilon/2$
    until A nonzero *gap* is found
    bi-partition the indexing set of $\mathbf{x}^{t+1}$ based on largest *gap* into i_1 and i_2.
    $C \leftarrow \text{RARD}(M(i_1, i_1),\ \epsilon_0), \qquad C \leftarrow \text{RARD}(M(i_2, i_2),\ \epsilon_0)$
end procedure
Output: Clustering C.

4.5 Time complexity

Each iteration of Algorithm 1 involves multiplication of two matrices of size $n \times n$ and $n \times d$, which requires $O(\bar{n}d)$ operations, where $\bar{n}$ is the number of nonzero entries in the matrix W. Algorithm 1 also calls the k-means algorithm, whose running time for one iteration is $O(nkd)$. So the complexity of Algorithm 1 is $O(\bar{n}dt_{\max}) + O(nkd)$, where $t_{\max}$ is the maximum number of iterations.

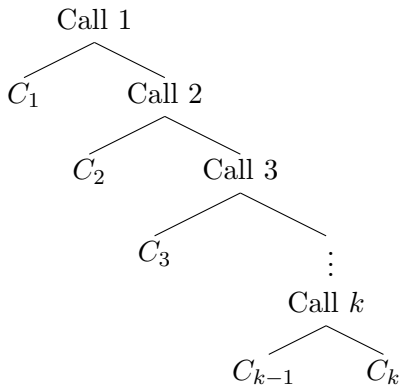


Figure 1: A depiction of the worst case.

Algorithm 3 is a recursive algorithm. For the sake of intuitive analysis, we assume that $n = 2^j$ for $j \geq 1$ and that all clusters are of equal size i.e., each cluster has n/k points. we further assume that

$k = 2^\ell$ for $\ell \geq 1$. In each recursive call, the algorithm performs a maximum of $t_{\max}$ sparse matrix-vector multiplications which require $O(\bar{n}t_{\max})$ operations. It also makes two more recursive calls and sorts n numbers except for the base case. Thus each non-base call takes $O(\bar{n}t_{\max}) + O(n \log n)$ operations, where $O(n \log n)$ is the complexity of sorting n numbers. Let $T(n, k, t_{\max})$ be the time required to find k clusters in a dataset of size n by Algorithm 3, then we have

$$\begin{aligned}
T(n, k, t_{\max}) &= 2T\left(\frac{n}{2}, k, t_{\max}\right) + c\bar{n}t_{\max} + n \log n \\
&= 4T\left(\frac{n}{4}, k, t_{\max}\right) + c\bar{n}t_{\max} + c\frac{\bar{n}}{2}t_{\max} + n \log n + \frac{n}{2}\log\frac{n}{2} \\
&\;\;\vdots \\
&= 2^\ell T\left(\frac{n}{2^\ell}, k, t_{\max}\right) + c\bar{n}t_{\max}\sum_{s=0}^{\ell-1}\frac{1}{2^s} + n\sum_{s=0}^{\ell-1}\frac{1}{2^s}\log\frac{n}{2^s} \\
&= 2^\ell c\frac{\bar{n}}{2^\ell}t_{\max} + c\bar{n}t_{\max}\sum_{s=0}^{\ell-1}\frac{1}{2^s} + n\log n\sum_{s=0}^{\ell-1}\frac{1}{2^s} - n\sum_{s=0}^{\ell-1}\frac{s}{2^s} \\
&= c\bar{n}t_{\max} + (c\bar{n}t_{\max} + n\log n)\sum_{s=0}^{\ell-1}\frac{1}{2^s} - n\sum_{s=0}^{\ell-1}\frac{s}{2^s} \\
&= O(\bar{n}t_{\max}) + O(\bar{n}t_{\max}) + O(n\log n) - O(n) \\
&= O(\bar{n}t_{\max}) + O(n\log n),
\end{aligned}$$

where c is a constant and we have used the fact that $\sum_{s=0}^{\ell} 1/2^s \leq 2$ and $\sum_{s=0}^{\ell} s/2^s \leq 2$. Similarly, if we assume a different split of the data in each recursive call, such as 1/3 and 2/3, or 1/4 and 3/4 etc., it easily follows from the above analysis that the running time of the Algorithm 3 remains $O(\bar{n}t_{\max}) + O(n \log n)$. The worst case arises when the dataset comprises a big cluster $\mathcal{V}_k$ of size (say) $n/2$ and rest of the dataset constitutes the other $k-1$ clusters $\mathcal{V}_1, \mathcal{V}_2, \ldots \mathcal{V}_{k-1}$. In such a scenario, if a smaller cluster from one of the $k-1$ clusters, say $\mathcal{V}_1$, splits from the data in the first recursive call, and in the second call on the dataset containing the big cluster, another smaller cluster $\mathcal{V}_2$ splits from the data and so on. Continuing in this way, suppose that the big cluster is the only cluster left in the last recursive call as shown in Figure 1, then we have made k recursive calls on a dataset of size at least $n/2$ making the running time of the algorithm $O(\bar{n}kt_{\max}) + O(nk \log n)$. However, if $\mathcal{V}_k$ splits from the dataset early on in the recursion, the running time remains $O(\bar{n}t_{\max}) + O(n \log n)$. Thus, unless the dataset contains a big cluster encompassing a dominant fraction of the dataset, the running time of the Algorithm 3 is $O(\bar{n}t_{\max}) + O(n \log n)$. We use p nearest neighbors to compute the similarities between points, which results in $O(pn)$ nonzero entries in W. Since p is a constant typically between 4 and 10, we conclude that number of nonzero entries in W is $O(n)$. Thus for the p-nearest neighbor similarity function, the complexity of Algorithm 3 is $O(nt_{\max}) + O(n \log n)$.

5 Simulation results

We begin with a toy example to illustrate the mechanics of Algorithm 3. Suppose that we have the following normalized similarity matrix for a dataset with three clusters. The intra-cluster similarities of the three clusters are represented by red, green and blue colors.

$$\begin{bmatrix}
0 & .5 & .45 & .025 & .025 & 0 & 0 & 0 & 0 & 0 \\
.4 & 0 & .55 & 0 & 0 & 0 & .05 & 0 & 0 & 0 \\
.3 & .7 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & .01 & 0 & 0 & .3 & .4 & .28 & 0 & .01 & 0 \\
0 & 0 & 0 & .4 & 0 & .3 & .3 & 0 & 0 & 0 \\
0 & 0 & .1 & .25 & .25 & 0 & .4 & 0 & 0 & 0 \\
.01 & 0 & 0 & .4 & .3 & .27 & 0 & .02 & 0 & 0 \\
0 & .01 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5 & 0.49 \\
0 & 0 & 0 & .02 & 0 & 0 & 0 & 0.49 & 00 & 0.49 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.7 & 0.3 & 00
\end{bmatrix}$$

The first call to the RARD procedure in Algorithm 3 starts by picking 10 points uniformly at random from $[0, 100]$. Stopping criterion is met after 27 iterations giving us the following vector. A bipartition of the dataset based on the *gap* criteria separates the blue cluster from the other two clusters.

$$\begin{bmatrix}
31.92 \\
32.15 \\
31.92 \\
32.64 \\
32.60 \\
32.55 \\
32.71 \\
39.57 \\
39.53 \\
39.63
\end{bmatrix}$$

Algorithm 3 then makes two recursive calls to the RARD procedure on the re-normalized sub-matrices corresponding to the two partitions. The recursive call on the first partition containing blue and green clusters results in a bipartition of the data into two clusters giving us the following vector, whereas the recursive call on the second partition containing the blue cluster does not find any *gap* satisfying the stopping criteria, so the algorithm identifies it as a cluster and returns out

of the recursion.

$$\begin{bmatrix} 77.63 \\ 78.03 \\ 78.07 \\ 67.06 \\ 66.91 \\ 67.78 \\ 66.98 \end{bmatrix} \begin{bmatrix} 21.12 \\ 20.98 \\ 21.28 \end{bmatrix}$$

The recursive call on the red and green clusters, then makes two further calls to the RARD procedure, one on the red and one on the green cluster. These two calls do not find a bipartition in the data and return out of the recursion identifying red and green clusters.

In the following sections, we demonstrate the efficiency of the proposed algorithm on a variety of synthetic and real datasets. In particular, we show that Algorithm 3 can identify clusters of complex shapes and varying sizes in Section 5.2 and compare its accuracy with the normalized cut algorithm [1]. We have used p-neighbor similarity measure to construct the similarity graph for all the experiments. In Section 5.3, we portray the scalability and speed of Algorithm 3 by applying it to large-scale stochastic block models. Finally in Section 5.4, we run Algorithm 3 on two real datasets and compare its accuracy and speed with normalized cut algorithm, fast approximate spectral clustering [8] and Nystrom method [6]. 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.

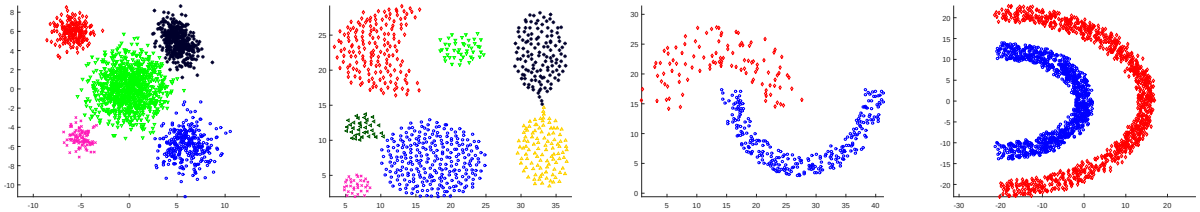
5.1 Performance evaluation

Mutual information is a symmetric measure to quantify the information shared between two distributions. It is widely used as a measure to calculate the shared information between two clusterings. Let $\mathcal{V}$ denote the cluster labels and $\mathcal{V}'$ be the clustering obtained by an algorithm. Their mutual information is defined as follows:

$$MI(\mathcal{V}, \mathcal{V}') = \sum_{\mathcal{V}_i \in \mathcal{V},\ \mathcal{V}'_j \in \mathcal{V}'} Pr(\mathcal{V}_i, \mathcal{V}'_j)\ \log\left(\frac{Pr(\mathcal{V}_i, \mathcal{V}'_j)}{Pr(\mathcal{V}_i)Pr(\mathcal{V}'_j)}\right)$$

where $Pr(\mathcal{V}_i)$ and $Pr(\mathcal{V}'_j)$ are the probabilities that an arbitrary point belongs to cluster $\mathcal{V}_i$ in clustering $\mathcal{V}$ and $\mathcal{V}'_j$ in clustering $\mathcal{V}'$ respectively, i.e.,

$$Pr(\mathcal{V}_i) = \frac{|\mathcal{V}_i|}{n} \quad \text{and} \quad Pr(\mathcal{V}'_j) = \frac{|\mathcal{V}'_j|}{n}$$



(a) Mixture of Gaussians (b) Clustering aggregation (c) Two crescents (d) Half ellipses

Figure 2: Clustering result of Algorithm 3 on synthetic datasets.

Table 1: Results of 50 runs of Algorithm 3 on synthetic datasets. Parameter settings $\alpha = 1$ and $\epsilon_0 = 10^{-2}, 10^{-3}, 10^{-3}, 10^{-4}$ top to bottom. NCut shows the average normalized cut defined in Equation (12) over 50 simulations with standard error. NMI is the mean normalized mutual information in 50 runs with standard error. Runtime shows the average computation time of a simulation.

Datasets	**NCut**	**NMI**	**Runtime**
Mixture of Gaussians	0.0052 ± 0.0000	97.12 ± 0.03	0.687
Clustering aggregation	0.0323 ± 0.0000	99.58 ± 0.03	0.106
Two crescents	0.0052 ± 0	100 ± 0	0.092
Half ellipses	0 ± 0	100 ± 0	0.969

$Pr(\mathcal{V}_i, \mathcal{V}'_j)$ is the joint probability that an arbitrary point lies in both clusters $\mathcal{V}_i$ and $\mathcal{V}'_j$ in clusterings $\mathcal{V}$ and $\mathcal{V}'$ respectively, i.e.,

$$Pr(\mathcal{V}_i, \mathcal{V}'_j) = \frac{|\mathcal{V}_i \cap \mathcal{V}'_j|}{n}$$

For the ease of interpretation, we use normalized mutual information defined as:

$$NMI(\mathcal{V}, \mathcal{V}') = \frac{MI(\mathcal{V}, \mathcal{V}')}{\sqrt{H(\mathcal{V})\, H(\mathcal{V}')}}$$

where $H(\mathcal{V})$ is the entropy of $\mathcal{V}$ given by:

$$H(\mathcal{V}) = -\sum_{\mathcal{V}_i \in \mathcal{V}} Pr(\mathcal{V}_i) \log(Pr(\mathcal{V}_i))$$

It is easy to verify that $0 \leq NMI(\mathcal{V}, \mathcal{V}') \leq 1$. NMI is 1 when the two clusterings are identical and 0 when the clusterings are independent.

Table 2: Computation time (in seconds) of Algorithm 3 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	$15,000$			$30,000$			$60,000$		
k	5	10	15	5	10	15	5	10	15
RARD	4.19	3.51	3.23	14.47	11.59	10.46	78.21	53.27	45.44
Normalized Cut	69.46	121.14	250.07	Out of Memory					

5.2 Synthetic datasets

Four two-dimensional synthetic datasets have been used to compare Algorithm 3 to the normalized cut algorithm [1]. The details of the datasets are given in Appendix A. Table 1 portrays the results of RARD algorithm. These results depict that our recursive implementation is very accurate in identifying clusters of complex shapes and different sizes. Small standard errors emphasize that RARD algorithm has little dependence on the initial vector $\mathbf{x}^0$.

5.3 Scalability

We apply Algorithm 3 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 2. Runtimes shown are average times for 50 different SBMs. Observe that RARD algorithm is significantly faster than normalized cut. It also consumes less memory as compared to normalized cut. For each model, the RARD 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 clusters. Figure 3 shows the number of simulations where RARD recovered correct clusters in 50 different SBMs while varying q, the probability of placing an edge between two nodes in different clusters.

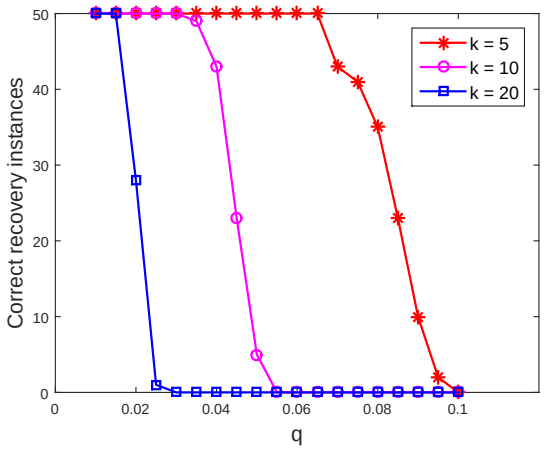


Figure 3: Correct clusterings recovered in 50 runs vs. q (probability of an edge between two points in different clusters). Parameters $n = 15000$ and $p = 0.5$.

Table 3: 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 is KASP algorithm as defined in [8]. 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 [6] with a random sample of 50% and 20% respectively.

(a) USPS

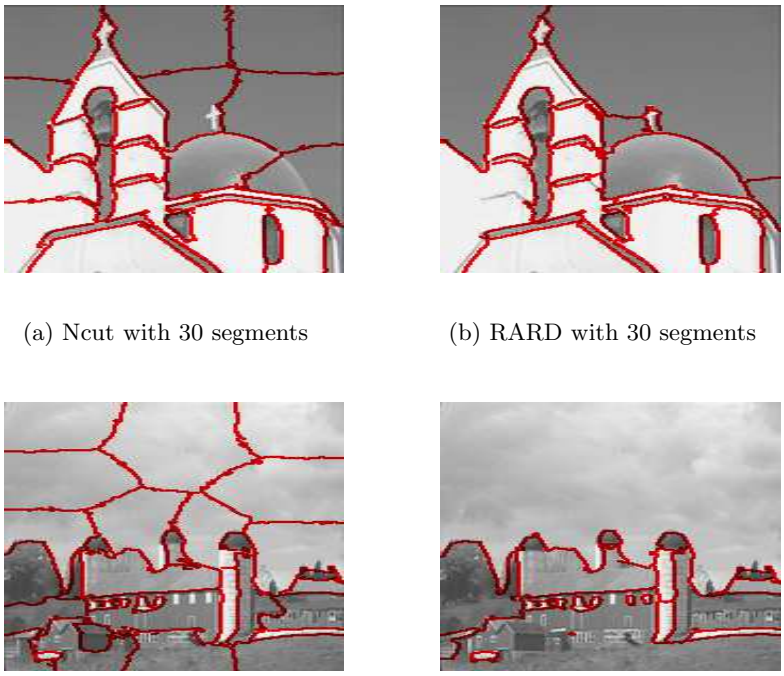
k	**NCUT**	**FASC 1**	**FASC 2**	**NYSTROM 1**	**NYSTROM 2**	**RARD**
4	$90.50 \pm 2.53(0.48)$	$84.80 \pm 1.26(4.46)$	$84.21 \pm 1.09(1.23)$	$74.36 \pm 1.47(1.57)$	$68.66 \pm 1.53(0.32)$	$88.07 \pm 1.26(0.37)$
6	$85.76 \pm 0.93(1.13)$	$80.88 \pm 0.87(6.28)$	$79.43 \pm 0.92(2.39)$	$73.06 \pm 1.02(2.21)$	$66.46 \pm 0.69(0.77)$	$85.23 \pm 0.87(0.58)$
8	$85.21 \pm 0.42(2.31)$	$79.28 \pm 0.67(8.93)$	$78.67 \pm 0.66(5.31)$	$69.93 \pm 0.63(4.54)$	$62.93 \pm 0.65(1.88)$	$84.56 \pm 0.41(0.82)$
10	$81.47 \pm 0.00(6.71)$	$78.18 \pm 0.42(12.30)$	$76.95 \pm 0.51(8.05)$	$67.20 \pm 0.27(6.60)$	$61.94 \pm 0.30(1.79)$	$82.33 \pm 0.22(1.14)$

(b) COIL20

k	**NCUT**	**FASC 1**	**FASC 2**	**NYSTROM 1**	**NYSTROM 2**	**RARD**
4	$98.68 \pm 0.96(0.15)$	$74.44 \pm 2.24(0.94)$	$72.82 \pm 2.04(0.41)$	$84.86 \pm 2.05(0.02)$	$77.31 \pm 2.51(0.01)$	$97.32 \pm 1.79(0.13)$
8	$97.15 \pm 0.90(0.38)$	$73.72 \pm 1.16(1.30)$	$71.35 \pm 1.27(0.82)$	$83.82 \pm 1.15(0.05)$	$73.25 \pm 1.53(0.04)$	$94.03 \pm 0.73(1.17)$
12	$94.58 \pm 0.79(0.72)$	$73.35 \pm 0.69(2.77)$	$71.01 \pm 0.69(1.84)$	$80.29 \pm 0.99(0.11)$	$69.88 \pm 1.12(0.07)$	$94.36 \pm 0.92(0.41)$
16	$92.28 \pm 0.69(1.23)$	$74.46 \pm 0.61(4.14)$	$70.66 \pm 0.45(2.44)$	$76.17 \pm 0.71(0.20)$	$67.05 \pm 0.62(0.11)$	$92.35 \pm 0.28(0.55)$
20	$91.93 \pm 0.00(1.98)$	$73.81 \pm 0.43(6.53)$	$70.92 \pm 0.41(3.59)$	$73.31 \pm 0.43(0.32)$	$63.61 \pm 0.46(0.16)$	$92.90 \pm 0.09(0.68)$

5.4 Real datasets

We empirically compare the accuracy and speed of our RARD algorithm with normalized cut algorithm [1], fast approximate spectral clustering [8] and Nystrom method [6] on two real datasets.



(a) Ncut with 30 segments

(b) RARD with 30 segments

(c) Ncut with 30 segments

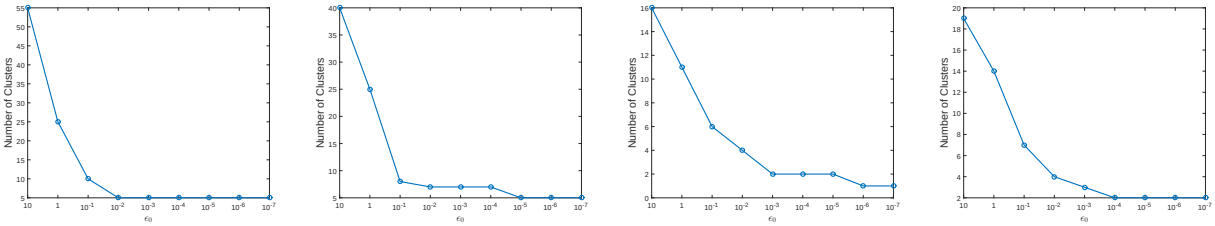
(d) RARD with 30 segments

Figure 4: Comparison of Ncut and RARD on image segmentation.

The USPS dataset has 7291 instances and length of the feature vector is 256 [12]. It has a total of 10 clusters. The COIL20 dataset consists of 1140 examples and has 1024 features with 20 clusters [13]. We use a p-nearest neighbor graph to construct the similarity matrix. For both dataset we use $p = 4$. ϵ_0 is set to 10^{-3} and 10^{-2} for USPS and COIL20 datasets respectively. We compare normalized mutual information and computation time of RARD algorithm with other algorithms. As demonstrated earlier RARD algorithm has the same accuracy as normalized cut algorithm. Our algorithm does not sacrifice accuracy as opposed to FASC and Nystrom method which also claim to improve the speed of spectral clustering. For large datasets RARD is faster than both FASC and Nystrom and does not compromise on the accuracy.

5.5 Image segmentation

We also test our RARD algorithm against normalized cut algorithm [1] on image segmentation on some standard images. We show that our algorithm performs significantly better than normalized cut algorithm when the number of segments is large. We follow the feature selection of Shi and Malik [1]. In particular, normalized cut fails to detect big segments and thus divides them into smaller sub-segments. On the other hand RARD can detect segments of varying sizes correctly.



(a) Mixture of Gaussians (b) Clustering aggregation (c) Two crescents (d) Half ellipses

Figure 5: ϵ_0 vs Ncut and number of clusters in Algorithm 3

The resulting segmentations are shown in Figure 4.

5.6 Parameter selection

Our algorithm proposed in this paper require only one parameter ϵ_0. We start with some initial value of ϵ_0 and monitor the number of clusters produced by RARD algorithm. We then decrease ϵ_0 by a constant factor and observe the number clusters detected by RARD algorithm. In this fashion, we search for two consecutive ϵ_0 with the same number of clusters, i.e., we set ϵ_0, when the number of clusters become stable. Figure 5 exhibits this procedure for synthetic datasets. If we already know the number of clusters k in the dataset, then the value ϵ_0 is chosen corresponding to the value of k. Gaussian similarity measure require a parameter σ which is difficult to tune. Automatically selecting σ is a challenging problem and has received interest in some recent studies. More detailed analysis of this subject is beyond the scope this work. Interested readers can refer to [14].

6 Connection to normalized cuts

Shi and Malik proposed a graph partitioning criteria known as normalized cut (NCut) [1]. A k-way NCut is defined as

$$\text{NCut}(\mathcal{V}_1, \mathcal{V}_2, \ldots, \mathcal{V}_k) = \sum_{i=1}^{k} \frac{cut(\mathcal{V}_i, \mathcal{V} - \mathcal{V}_i)}{a(\mathcal{V}_i)}, \tag{12}$$

where $cut(\mathcal{A}, \mathcal{B}) = \sum_{i \in \mathcal{A}, j \in \mathcal{B}} w_{ij}$ and $a(\mathcal{A}) = \sum_{i \in \mathcal{A}, j \in \mathcal{V}} w_{ij}$. They showed that minimizing the 2-way normalized cut to obtain a bipartition of the graph is equal to minimizing the following Rayleigh quotient.

$$\min_{\mathbf{y}} \frac{\mathbf{y}^T (D - W)\mathbf{y}}{\mathbf{y}^T D \mathbf{y}}$$

where D is the degree matrix. The vector $\mathbf{y}$ of length n satisfies $\mathbf{y}^T D\mathbf{1} = 0$ and $y_i \in \{1, -b\}$ with b some constant in $(0, 1)$. This problem is NP-Hard [15] and is approximated by relaxing $\mathbf{y}$ to take on real values. This approximation leads to solving the generalized eigenvalue problem $(D - W)\mathbf{y} = \lambda D\mathbf{y}$ for the second smallest eigenvalue also known as the Fiedler value. Since D is invertible the generalized eigenvalue problem is equivalent to solving

$$(I - D^{-1}W)\mathbf{y} = \lambda\mathbf{y}. \tag{13}$$

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 combination of the eigenvectors of $D^{-1}W$. Observe that eigenvectors of both these systems are same, however the eigenvalues are translated by 1.

7 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. It rather finds an eigenvalue weighted linear combination of eigenvectors of the normalized similarity matrix. Our algorithms are simple to implement and computationally efficient. We have demonstrated the scalability and accuracy of the RARD algorithm by implementing it on large stochastic block models with tens of thousands of nodes and hundreds of millions of edges. We have also shown that our RARD algorithm has the same accuracy as normalized cut algorithm. Thus our algorithm does not compromise on accuracy to achieve faster speed unlike other algorithms which claim to speed-up spectral clustering such as fast approximate spectral clustering [8] and Nystrom method [6].

Acknowledgments

This research is supported in part by NSF grant CNS 13-30077 and DMS 1312907.

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. *D-MATH, EHT Zurich*, 2012.

[6] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the Nystrom 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] Herbert A Simon and Albert Ando. Aggregation of variables in dynamic systems. *Econometrica: Journal of the Econometric Society*, pages 111–138, 1961.

[12] 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 backpropagation network. In *Advances in Neural Information Processing Systems 2*, pages 396–404. 1990.

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

[14] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. In *Advances in Neural Information Processing Systems 17*, pages 1601–1608. 2005.

[15] Charu C Aggarwal and Chandan K Reddy. *Data clustering: algorithms and applications*. CRC Press, 2013.

[16] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. Clustering aggregation. *ACM Transactions on Knowledge Discovery from Data*, 1(1):4, 2007.

A Details of Datasets

A.1 Synthetic Datasets

- *Mixture of Gaussians:* We consider a mixture of five Gaussians random variables X_1, X_2, X_3, X_4 and X_5 with different densities. The five mean vectors and covariance matrices are

$$\mu_1 = \begin{bmatrix} -5 \\ -5 \end{bmatrix}, \quad \mu_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \mu_3 = \begin{bmatrix} 6 \\ -6 \end{bmatrix}, \quad \mu_4 = \begin{bmatrix} -6 \\ 6 \end{bmatrix}, \quad \mu_5 = \begin{bmatrix} 5 \\ 5 \end{bmatrix}$$

$$\Sigma_1 = \begin{bmatrix} .5 & 0 \\ 0 & .5 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 3.5 & 0 \\ 0 & 3.5 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},$$

$$\Sigma_4 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \quad \Sigma_5 = \begin{bmatrix} 1 & -.5 \\ -.5 & 1.5 \end{bmatrix}$$

  A sample of 2000 points is taken from these distributions with 100, 1000, 300, 200 and 400 samples from X_1, X_2, X_3, X_4 and X_5 respectively. We have used $\sigma = 0.5$ for RBF similarity.

- *Clustering aggregation:* This dataset set is taken from Clustering aggregation paper [16]. The authors show that single link, complete link, average link, ward's method and k-means fail to recover the correct clusters in this data set. It has 7 clusters and 788 total points. Parameter σ is kept at 1 for RBF similarity function.

- *Two crescents:* This dataset has two clusters with a total of 384 points. The value of parameter σ in the RBF similarity function is kept equal to 1.5.

- *Half ellipses:* This dataset contains a total of 2000 points with 1000 points in each cluster. Parameter $\sigma = 2.5$ is used for RBF similarity.

A.2 Real Datasets

- *USPS* [12]: This dataset contains 7291 grayscale images of digits $0-9$ scanned from envelopes by United States Postal Service. Each feature vector consists of normalized grayscale values of $16 \times 16 = 256$ pixels.

- *COIL20* [13]: This dataset consists of 32×32 grayscale images of 20 different objects. 72 different images of each object are taken from 72 different angles as the objects are rotated on a table, i.e., after every 5 degree rotation an image is taken. A feature vector consists of 1024 normalized grayscale values i.e., one value for each pixel.

Figure 6: A sample of USPS dataset

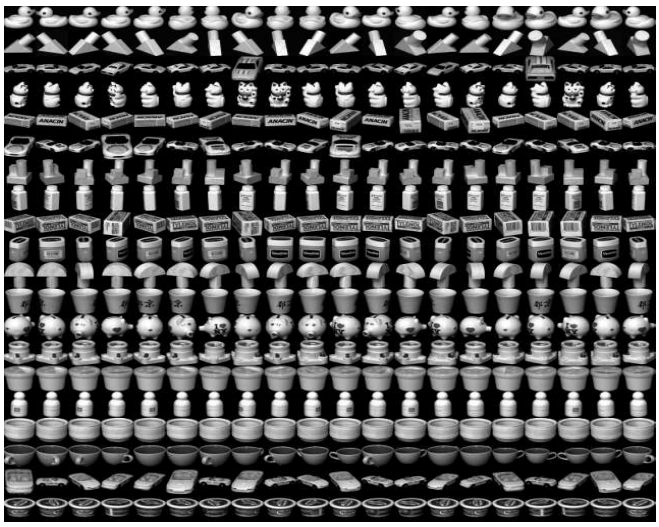


Figure 7: A sample of COIL20 dataset