Distributed Spectral Decomposition in Networks by Complex Diffusion and Quantum Random Walk

Konstantin Avrachenkov*, Philippe Jacquet†, Jithin K. Sreedharan*
*INRIA, Sophia Antipolis, France
†Alcatel-Lucent Bell Labs, France
e-mails: k.avrachenkov@inria.fr, philippe.jacquet@alcatel-lucent.com, jithin.sreedharan@inria.fr

***Abstract*—In this paper we address the problem of finding top k eigenvalues and corresponding eigenvectors of symmetric graph matrices in networks in a distributed way. We propose a novel idea called complex power iterations in order to decompose the eigenvalues and eigenvectors at node level, analogous to time-frequency analysis in signal processing. At each node, eigenvalues correspond to the frequencies of spectral peaks and respective eigenvector components are the amplitudes at those points. Based on complex power iterations and motivated from fluid diffusion processes in networks, we devise distributed algorithms with different orders of approximation. We also introduce a Monte Carlo technique with gossiping which substantially reduces the computational overhead. An equivalent parallel random walk algorithm is also presented. We validate the algorithms with simulations on real-world networks. Our formulation of the spectral decomposition can be easily adapted to a simple algorithm based on quantum random walks. With the advent of quantum computing, the proposed quantum algorithm will be extremely useful.**

I. Introduction

Spectral properties of a graph or a network are of interest to diverse fields due to its strong influence in many practical algorithms. However the computational complexity associated to the estimation of eigenvalue spectrum and eigenvectors has been a demanding problem for a long time. In the context of network science, the design of *distributed* spectral decomposition methods is particularly important.

We consider any symmetric matrix associated to an undirected graph with n nodes and weighted edges. For instance, the matrix can be the adjacency matrix $\mathbf{A} = [a_{ij}], 1 \leq i, j \leq n$ in which a_{ij} denotes the weight of the edge between the nodes i and j. Due to symmetry the eigenvalues are real and can be ranked in decreasing order as $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n$. We study the largest k eigenvalues $\lambda_1, \ldots, \lambda_k$ and the corresponding eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$.

Though the ideas proposed in the paper are applicable to any symmetric graph matrix, we use the adjacency matrix $\mathbf{A}$ to illustrate the ideas in the rest of the paper.

A. Relevance of spectral decomposition

The existing works list many applications of the spectrum of graph matrices. The applications are matrix dependent and here we give a very limited list of uses for the adjacency matrix. In particular, the eigenvalues and eigenvectors can be needed globally or locally. Global applications require a central unit to collect eigenvalues and eigenvector components from all the nodes and then pass this global information to the algorithms behind the application. But in case of local applications, the underlying algorithms run separately at each node by making use of the respective component in the extreme eigenvectors, along with the knowledge of eigenvalues.

Following are some of the applications of the adjacency matrix related to graphs and networks.

1) Number of triangles: The spectral information of adjacency matrix can be used to obtain information about the global as well as local knowledge of the number of triangles (in other words, about global and local clustering). The total number of triangles in a graph is $1/6 \sum_{i=1}^{n} |\lambda_i|^3$ and the number of triangles that a node m participated in, is $1/2 \sum_{i=1} |\lambda_i|^3 \mathbf{u}_i(m)$ [1]. Hence if we calculate top k eigenvalues and eigenvector components locally at node m, we can approximate with good accuracy how much connected its neighbors are.

2) Dimensionality reduction, link prediction and Weak and strong ties: The knowledge of k top eigenvector components at a node maps it into a point in $\mathbb{R}^k$. Then new links can be suggested among unconnected nodes when the distance between them in the mapped vector space is small [2].

Weak ties occur when the endpoints of an edge are part of well connected nodes, but with very few common friends in between the endpoints. Strong ties happen in the opposite sense [3]. The k-dimensional vector associated to the endpoints can be used to find out weak or strong ties.

3) Finding near-cliques: Typical graph clustering works on the whole graph and will often assign isolated nodes to some clusters, and subsequently would fail to detect communities with good internal coherence. Therefore it is practically relevant to find communities which are like cliques and extract it from the main graph. The work in [4] shows that the spectrum of adjacency matrix can be useful for this. They propose the idea of EigenSpokes which is a phenomenon whereby straight lines are observed in the eigenvector-eigenvector scatter plots of the adjacency matrix. It is then derived that nodes which are placed near to each other on the EigenSpokes have the same set of neighbors and hence can be used to detect clique-like structures.

This work is supported by INRIA Alcatel-Lucent joint lab (ADR Network Science).

4) Spectral clustering: For the problem of finding clusters in a network, spectral clustering is a prominent solution among the studied techniques [5]. In its basic statement, the spectral clustering algorithm takes the first k normalized eigenvectors $\mathbf{u}_1, \ldots, \mathbf{u}_k$ of the adjacency or the Laplacian matrix. Let $\boldsymbol{\phi}(h)$ be a vector made of components of node h in $\mathbf{u}_1, \ldots, \mathbf{u}_k$. The k-means clustering algorithm is then applied to different $\boldsymbol{\phi}(h)$'s, and this will partition the nodes in the network. In fact which matrix to take is dependent on the objective function to optimize (*average association* in case of adjacency matrix instead of *normalized cut* in Laplacian) [6]. The main bottleneck in spectral clustering is the computation of extreme eigenvectors, which we try to accomplish here with much less complexity. Recently another promising spectral clustering method based on non-backtracking matrix is introduced in [7].

B. Related work and contributions of the paper

We provide a straightforward interpretation of eigenvalue spectrum and eigenvectors in terms of peaks in the frequency domain of complex exponential of $\mathbf{A}$ and exploit it for developing distributed algorithms. To the best of our knowledge, the first distributed network spectral decomposition algorithm was proposed in [2]. The most challenging part of the algorithm in [2] is the distributed orthonormalization at each step of the algorithm. This a difficult operation, which the authors solve by communicating information via random walks. Clearly if the graph has a small conductance (a typical case for many large graphs), this operation will take extremely long time at each step of the algorithm. Our first distributed algorithm based on the diffusion of complex fluid across the network, an implementation of *complex power iterations*, do not require orthonormalization. In [8], [9] the authors use techniques from signal processing which are in the same spirit of our approach. However their approach either need to use two time steps or two hop-neighbourhoods for each iteration, while our algorithms work with one time step and one-hop neighbourhood. The approach of [8], [9] distort the values of eigenvectors and eigenvalues and the correction needed is not evident. Moreover, since the methods in [8], [9] are based on classic Fourier transforms, the eigenvalues might not get detected because of spurious picks in the spectrum. Our approach overcomes this problem by using Gaussian smoothing. Our algorithms can also be immediately implemented via light weight gossiping and random walks with complex rewards. A recent gossip algorithm based on reinforcement learning was introduced in [10], but it computes only the principal eigenvector. From the analysis of the our diffusion technique, we observe that our algorithms are scalable of the order of maximum degree. Finally our method has a very interesting relation to the quantum random walks, which with the advancement of quantum computing can make our approaches very efficient.

C. Organization

The paper is organized as follows. In Section II we review some classical challenges in computing the spectrum of graphs. Section III presents the central idea of the paper, complex power iterations, and explains various approximations. In Section IV we present in detail two distributed approaches for network spectral decomposition. Then, in Section V we explain that our approach can be efficiently realized using a quantum algorithm. In Section VI we analyse the error terms and provide recommendations for the choice of parameters. Numerical results presented in Section VII demonstrate the scalability of the approach. In Section VIII we discuss an extension to non-symmetric matrices.

II. Challenges in classic techniques

Here we illustrate two problems (among many) faced by existing techniques with the help of two classic algorithms.

The first one, power iteration, consists of computing the iterative power $\mathbf{b}_\ell = \mathbf{A}^\ell \mathbf{b}_0$ for the increasing integer ℓ with $\mathbf{b}_0$ as an initial vector. Using the spectral decomposition of $\mathbf{A}$, we have

$$\mathbf{A}^\ell = \sum_j \lambda_j^\ell \mathbf{u}_j \mathbf{u}_j^\intercal \ .$$

We adopt the convention $\|\mathbf{u}_j\| = 1$. Depending of λ_1 being greater or smaller than one, the iteration $\mathbf{b}_\ell = \mathbf{A}^\ell \mathbf{b}_0$ for $\ell \geq 1$ will exponentially decrease or increase without proper step normalization. The normalization introduced is

$$\mathbf{b}_{\ell+1} = \frac{1}{\|\mathbf{b}_\ell\|} \mathbf{A}\mathbf{b}_\ell \ . \tag{1}$$

Notice that $\mathbf{b}_\ell$ converges to $\lambda_1 \mathbf{u}_1$ when $\ell \to \infty$. The problem is that this method cannot be readily applied to the search of the other eigenvalues because the first eigenvalue screens the exponentially decreasing secondary eigenvalues: $\mathbf{b}_\ell = \lambda_1 \mathbf{u}_1 + O\left((\frac{\lambda_2}{\lambda_1})^\ell\right)$.

In order to compute the other eigenvalues one can use the inverse iteration methods based on the formula

$$\mathbf{b}_{\ell+1} = \frac{1}{\|\mathbf{b}_\ell\|} (\mathbf{A} - \mu \mathbf{I})^{-1} \mathbf{b}_\ell, \tag{2}$$

for an arbitrary real number $\mu < \lambda_1$. The iteration will converge to $\frac{1}{\lambda_j - \mu}\mathbf{u}_j$ where j is the index of the eigenvalue closest to μ. The search consists of approaching the eigenvalue by tuning the parameter μ. The difficulty of the method is in computing the inverse matrix (or solution of the linear system) for each selected values μ which is computationally costly. Furthermore, the use of normalization at each iterative step, in (1) as well as in (2), will make it difficult an adaptation to the distributed context, i.e., the normalization requires frequent pausing of the iteration in order to compute and disseminate the normalization factor.

In the next section we propose a new method called complex power iterations. From an initial vector $\mathbf{b}_0$ and given integer k, the method will return the k first eigenvalues $\lambda_1, \ldots, \lambda_k$ and the vectors $\langle \mathbf{b}_0 \mathbf{u}_j \rangle \mathbf{u}_j$ for $j = 1, \ldots, k$. Notice that the $\mathbf{u}_j$'s can be retrieved via normalization, but it will turn out that this is an unnecessary step.

III. Complex power iterations

Now we introduce the main idea in the paper, the complex power iterations, which we use to compute the spectrum of the matrix of a network in an efficient way. We consider an undirected graph. For convenience we summarize the important notation used in this paper in Table I. We derive a technique

Notation	Meaning
G	Graph
(V, E)	Node set and edge set
n	Number of nodes
$\mathbf{A}$	Adjacency matrix
a_{ij}	weight of the connection from nodes i to j in $\mathbf{A}$
$\lambda_1, \ldots, \lambda_k$	Top k eigenvalues in descending order
$\mathbf{u}_1, \ldots, \mathbf{u}_k$	Eigenvectors corresponding to $\lambda_1, \ldots, \lambda_k$
$\mathcal{N}_j$	Neighbor list of node j without including self loop
D_j	Degree of node j without including self loop
Δ	$\max\{D_1, \ldots, D_n\}$
$\mathbf{a}(k)$	kth component of a column vector $\mathbf{a}$
$\|\mathbf{a}\|$	Euclidean norm of vector $\mathbf{a}$
$\mathbf{x}_\ell$	Approximation of $\exp(i\varepsilon\ell\mathbf{A})$ multiplied by $\mathbf{b}_0$
$\mathbf{x}_\ell$	Approximation of $\exp(i\varepsilon\ell\mathbf{A})$ multiplied by $\mathbf{b}_0$

TABLE I
List of important notations

that is analogous to the frequency spectrum in the time-frequency analysis and apply it to graph spectrum analysis. In this perspective, the domain of eigenvalues corresponds to the frequency domain.

From the eigendecomposition of the symmetric matrix $\mathbf{A}$, $\mathbf{A} = \sum_j \lambda_j \mathbf{u}_j \mathbf{u}_j^\intercal$, we have $e^{i\mathbf{A}t} = \sum_j e^{it\lambda_j} \mathbf{u}_j \mathbf{u}_j^\intercal$. Notice that the function $e^{i\mathbf{A}t}$ is pseudo periodic and its harmonics correspond exactly to the spectrum of the matrix. The advantage of complex exponential is that the whole spectrum can be recovered via the classic Fourier transform, contrary to the expression $e^{\mathbf{A}t}$ (dropping the imaginary unit i) where the effect of the other eigenvalues λ_j for $j > 1$ decays exponentially when $t \to \infty$. Indeed, we formally have

$$\frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{i\mathbf{A}t} e^{-it\theta} dt = \sum_{j=1}^{n} \delta_{\lambda_j}(\theta) \mathbf{u}_j \mathbf{u}_j^\intercal \ , \tag{3}$$

with δ_{λ_j} being the Dirac function translated of the quantity λ_j. However this expression is not easy to handle numerically because any truncated or discretization version of the integral will generate too large fake harmonic oscillations that will hide the Dirac peaks (we show an example of this in Section VII). To overcome this problem we use the spectral smoothing via convolution with a Gaussian function of variance $v > 0$:

$$\begin{aligned} &\frac{1}{2\pi} \int_{-\infty}^{+\infty} e^{i\mathbf{A}t} e^{-t^2 v/2} e^{-it\theta} dt \\ &= \sum_{j=1}^{n} \frac{1}{\sqrt{2\pi v}} \exp\left(-\frac{(\lambda_j - \theta)^2}{2v}\right) \mathbf{u}_j \mathbf{u}_j^\intercal \ , \end{aligned} \tag{4}$$

Notice that the above expression converges to (3) when $v \to 0$. In order to ease visualization, a scalar product is taken with an initial vector $\mathbf{b}_0$. Figure 1 shows a sample plot produced from (4) at some node m, by varying θ. The detection of the eigenvalues corresponds to locating the peaks and the

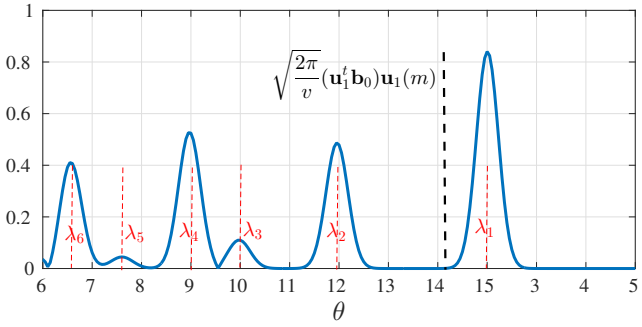

Fig. 1. Spectral plot at node m.

quantities $\sqrt{\frac{2\pi}{v}}(\mathbf{b}_0^\intercal \mathbf{u}_j)\mathbf{u}_j(m)$ corresponds to the values at these peaks as we will see later.

The key for the computation of (4) is the determination of the factor $e^{i\mathbf{A}t}$ which does not come naturally from the expression of matrix $\mathbf{A}$. We fix a number $\varepsilon > 0$ and we use the discretization $e^{i\mathbf{A}\ell\varepsilon} = (\mathbf{I} + i\varepsilon\mathbf{A})^\ell (1 + O(\varepsilon^2 \ell))$, where $\mathbf{I}$ is the identity matrix, for the calculation of left-hand side in (4),

$$\begin{aligned} &\int_{-\infty}^{+\infty} e^{i\mathbf{A}t} e^{-t^2 v/2} e^{-it\theta} dt \\ &= \varepsilon \Re \left(\mathbf{I} + 2 \sum_{\ell=1}^{d_{\max}} (\mathbf{I} + i\varepsilon\mathbf{A})^\ell e^{-i\ell\varepsilon\theta} e^{-\ell^2\varepsilon^2 v/2} \right) \\ &\quad + O(\varepsilon^2 d_{\max}) \end{aligned} \tag{5}$$

The quantity $\ell\varepsilon$ in the sum plays the role of variable t in the integral. Applying the expression to an initial vector $\mathbf{b}_0$, we define,

$$\mathbf{f}_\theta = \varepsilon \Re \Big(\mathbf{b}_0 + 2 \sum_{\ell=1}^{d_{\max}} e^{-i\ell\varepsilon\theta} e^{-\ell^2\varepsilon^2 v/2} \mathbf{x}_\ell \Big), \tag{6}$$

where $\mathbf{x}_\ell$ is used to approximate $e^{i\varepsilon\ell\mathbf{A}}\mathbf{b}_0$. For instance, in (5), $(\mathbf{I} + i\varepsilon\mathbf{A})^\ell$ is taken as an estimate of $e^{i\varepsilon\ell\mathbf{A}}$.

We notice that the expression of $\mathbf{f}_\theta$ does not use any discretisation over the θ variable or on the v variable and turns out to be analytical functions of these variable. Therefore the search of peaks corresponding to the eigenvalues turns out to be finding the zeroes of the derivative of $\mathbf{f}_\theta$ under the condition that $\mathbf{f}_\theta$ is above a certain threshold.

This process and the way to tune the best values of ε, v and $d_{\max}$ will be discussed in Section VI-C. In the next section we refine some higher order approximation of $e^{i\varepsilon\ell\mathbf{A}}$ which can be used to obtain more accurate expressions.

A. Higher Order Approximations

The approximation $\mathbf{x}_\ell$ of $e^{i\ell\varepsilon\mathbf{A}}\mathbf{b}_0$ in (6) can be made more accurate by using Runge-Kutta (RK) methods. Indeed, we have $\mathbf{x}_{\ell+1} = (\mathbf{I} + i\varepsilon\mathbf{A})\mathbf{x}_\ell$. We also notice that $e^{i\mathbf{A}t}\mathbf{b}_0$ is the solution of the differential equation $\dot{\mathbf{x}} = (i\mathbf{A})\mathbf{x}$ with initial condition $\mathbf{x}(0) = \mathbf{b}_0$. We use Runge-Kutta (RK) methods [11] to solve this differential equation numerically. With $\mathbf{x}(\ell\varepsilon)$ approximated by $\mathbf{x}_\ell$, the iteration in such a method can be defined as follows,

$$\mathbf{x}_{\ell+1} = \mathbf{x}_\ell + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4, \tag{7}$$

where $\mathbf{x}_0 = \mathbf{b}_0$, $k_1 = \varepsilon(i\mathbf{A}\mathbf{x}_\ell)$, $k_2 = \varepsilon i\mathbf{A}(\mathbf{x}_\ell + 1/2\, k_1)$, $k_3 = \varepsilon i\mathbf{A}(\mathbf{x}_\ell + 1/2\, k_2)$ and $k_4 = \varepsilon i\mathbf{A}(\mathbf{x}_\ell + k_3)$. It is observed that (7) is equivalent to $\mathbf{x}_\ell = \left(\mathbf{I} + (i\varepsilon\mathbf{A}) + \ldots + \frac{(i\varepsilon\mathbf{A})^4}{4!}\right)^\ell \mathbf{b}_0$. This is the RK method of order-4. This equivalence can be easily generalized to any order r RK methods as,

$$\mathbf{x}_\ell = \left(\sum_{j=0}^{r} \frac{(i\varepsilon\mathbf{A})^j}{j!}\right)^\ell \mathbf{b}_0. \tag{8}$$

In this paper we use order $r = 1, 2$ and 4.

IV. Complex Diffusion

The previous analysis is based on a centralised approach where we can manage to make matrix multiplications. The real challenge in the networking context is to make the computations distributed. Thus, in order to compute (6), we propose and compare the following three techniques:

(i) **Centralized approach:** Here we assume that the adjacency matrix $\mathbf{A}$ is fully known to a centralized unit. We use order-1 (5), order-2 or order-4 (7) technique to compute the approximation.

(ii) **Complex diffusion:** It is a distributed and asynchronous approach in which only local information is available at each node, which is the list of its neighbors. Here, each node communicates with all its neighbors at each time epoch.

(iii) **Monte Carlo techniques:** This is also a distributed technique with only local information, but with much reduced complexity than Complex diffusion as each node communicates with only one neighbor. Monte Carlo techniques can be implemented either using Monte Carlo Gossiping or using parallel random walks.

The matrix $\mathbf{X}$ (size $n \times (d_{\max}+1)$) approximates $e^{i\ell\varepsilon\mathbf{A}}$ for $0 \le \ell \le d_{\max}$,

$$\begin{aligned}\mathbf{X} &= \left[\mathbf{x}_0 \quad \mathbf{x}_1 \quad \ldots \quad \mathbf{x}_{d_{\max}}\right] \\ &\approx \left[\mathbf{b}_0 \quad e^{i\varepsilon\mathbf{A}}\mathbf{b}_0 \quad \ldots \quad e^{id_{\max}\varepsilon\mathbf{A}}\mathbf{b}_0\right]\end{aligned}$$

The above three methods employ different techniques to compute $\mathbf{X}$. At any node m, once the corresponding row in $\mathbf{X}$ is computed, then $\mathbf{f}_\theta(m)$ (cf. (6)) can be calculated independent of other nodes, and thus spectral plot and the extreme eigenvalues and eigenvectors are obtained.

A. Complex Diffusion

The key of the computation in (6) is the calculation of the sequence $\mathbf{x}_\ell$ or the associated polynomial $\mathbf{x}(z) = \sum_{\ell=0}^{d_{\max}} z^\ell \mathbf{x}_\ell$. Complex Diffusion uses the idea of fluid diffusion in networks to compute the coefficients of this polynomial in z. The algorithms proposed in this section are distributed in the sense that each node needs only to know and to communicate with its neighbors. They can be made asynchronous as well since there is no central clock to control the fusion-diffusion process.

We first consider the complex diffusion based on the order-1 approximation, i.e., $\mathbf{x}_\ell = (\mathbf{I} + i\varepsilon\mathbf{A})^\ell \mathbf{b}_0$. For order-1 calculations, the node m will start with an initial fluid $\mathbf{b}_0(m)$ and a copy of this fluid is diffused to all of its neighbors with weight $i\varepsilon a_{m,h}, h \in \mathcal{N}(m)$. A copy is also diffused to itself with weight $1+i\varepsilon a_{mm}$. The technique is detailed in Algorithm IV.1. At each node, we compute the polynomial $x(z)$ which corresponds to the equivalent row in $\mathbf{X}$. Then vector $\mathbf{x}(z)$ is made of the polynomials $x(z)$ computed on each node m. In the algorithm, the procedure $\textsc{Send}(h, f)$ transmits fluid f to node h and $\textsc{Receive}(h)$ collects fluid from h.

Algorithm IV.1: $\textsc{ComplexDiffusion}(m, \mathbf{b}_0)$

$C(z) \leftarrow \mathbf{b}_0(m)$
$d \leftarrow 0$
while $(d \le d_{\max})$
do
  $x(z) \leftarrow x(z) + C(z)$
  for each $h \in \mathcal{N}(j)$
    do $\{\textsc{Send}(h, a_{m,h}\varepsilon i C)$
  $C(z) \leftarrow (1 + i\varepsilon a_{j,j})C(z)z$
  for each $h \in \mathcal{N}(j)$
    do $\{C(z) \leftarrow C(z) + z\textsc{Receive}(h)$
  $d \leftarrow d + 1$
return $(x(z))$

The total number of fluid diffusion-fusion cycles at each node should be $d_{\max}$ in case the diffusions are synchronised. For asynchronous diffusions the diffusion will stop when all quantities C have only monomials of degree larger than $d_{\max}$, thus equal to zero after truncation. This will occur when all paths of length $d_{\max}$ have been processed in sequence. This can be easily detected if we assume a maximum time $t_{\max}$ for a diffusion-fusion cycle on each node, the process should stop after any laps of duration $t_{\max}d_{\max}$ with no diffusion-fusion.

At first sight, the quantities $x(z)$'s would need to be collected only at the end of the parallel computations. In fact, even this is not needed since the computation of $\mathbf{f}_\theta(m)$ and the peak detection can be made locally as long as the initial vector $\mathbf{b}_0$ is not orthogonal to the $\mathbf{u}_j$, and the quantity $(\mathbf{b}_0^\intercal \mathbf{u}_j)\mathbf{u}_j(m)$ be returned.

The polynomial technique explained in Algorithm IV.1 can also be implemented in vector form. We can extend the technique to higher order approximations. The pseudo-code in Algorithm IV.2 implements the order-2 complex diffusion. The use of parameter $C_2(z)$ is the artefact for the diffusion of matrix $(i\varepsilon\mathbf{A})^2$. Indeed, the fluid must be retransmitted towards relay before being added to $x(z)$. This is the reason why the number of iteration must be raised to $2d_{\max}$. The generalisation to a diffusion of order-r, is straightforward since it consists of the diffusion of matrix $(i\varepsilon\mathbf{A})^r$. To this end we add additional parameters $C_2(z)$ by a vector $C_r(z)$ with $r-1$ components $C_r^2, \ldots C_r^r$ and the procedure $\textsc{Send}$ transmits $i\varepsilon a_{m,h}(C(z)+C_r^r(z))$ and $\textsc{Send}$-r consists of transmitting the vector $[C(z), C_r^2(z, \ldots, C_r^{r-1}(z)]$.

In parallel each node can compute their function $\mathbf{f}(\theta)$ (cf. (6)) and detect the peaks. When a node has detected the values λ_ℓ for those peaks, with $\ell = 1, \ldots, k$ and the corresponding values $\mathbf{f}(\lambda_\ell)$ are broadcasted in a specific packet with a specific

Algorithm IV.2: DIFFUSIONORDER2$(m, \mathbf{b}_0)$

$C(z) \leftarrow \mathbf{b}_0(m)$
$C_2(z) \leftarrow 0$
$d \leftarrow 0$
while $(d \leq 2d_{\max})$
do
- $x(z) \leftarrow x(z) + C(z)$
- **for each** $h \in \mathcal{N}(m)$
  - **do**
    - SEND$(h, a_{m,h}\varepsilon i(C(z) + \frac{1}{2}C_2(z)))$
    - SEND-2$(h, a_{m,h}\varepsilon iC(z))$
- $C_2(z) \leftarrow i\varepsilon a_{m,m}C(z)$
- $C(z) \leftarrow (1 + i\varepsilon a_{m,m})zC(z)$
- **for each** $h \in \mathcal{N}(m)$
  - **do**
    - $C(z) \leftarrow C(z) + z$RECEIVE(h)
    - $C_2(z) \leftarrow C_2(z) +$ RECEIVE-2(h)
- $d \leftarrow d + 1$

return $(x(z))$

sequence number. The packet will be repeated all along the network. At the end of the process, all the nodes in the network can reconstitute the vectors $\mathbf{f}(\lambda_\ell)$ and perform any algorithm based on eigenvectors. It is very possible that the values of the λ_ℓ will not perfectly match from one node to another, but this is not crucial since, in most cases, the structure of the vectors $\mathbf{f}(\lambda_\ell)$ can accept some inaccuracy.

Complexity of the algorithm

In case of inverse power iteration method, assuming $d_{\max}$ iterations for different μ values (best case), we get the net computation cost (in terms of packets exchanged) as $|E|n^2 + (n|E| + |E|)d_{\max}$ (The first term due to diffusion of coefficients of $(\mathbf{A} - \mu\mathbf{I})^{-1}$ in the whole network, and second term due to the iterations and final collection in a central node). If we consider the delay, many packets may fly in parallel, and a diffusion will have a delay proportional to D, the diameter of the graph. Therefore the net delay will be proportional to $D + 2Dd_{\max}$. Notice that this will correspond to a throughput proportional to $n\frac{|E|}{D}$ in each iteration. But in case of complex diffusion the total number of packets exchanged is $|E|d_{\max} + n|E|$ and the net delay is $d_{\max} + D$. The throughput here is $\frac{|E|}{D}$, much smaller by a factor n than the throughput of the power iteration.

B. Complex Gossiping

In the order-1 computation (5), the terms $(\mathbf{I} + i\varepsilon\mathbf{A})^\ell\mathbf{b}_0$ for $0 \leq \ell \leq d_{\max}$ can also be calculated by the following Monte Carlo approach. We have

$$\mathbf{x}_{k+1} = (\mathbf{I} + i\varepsilon\mathbf{A})\mathbf{x}_k, \quad \mathbf{x}_0 = \mathbf{b}_0.$$

Then

$$\mathbf{x}_{k+1} = \mathbf{x}_k + i\varepsilon\mathbf{D}\mathbf{P}\mathbf{x}_k,$$

where $\mathbf{D}$ is the diagonal matrix with entries as the degrees of the nodes $(D_1, \ldots, D_n)$, and $\mathbf{P} = \mathbf{D}^{-1}\mathbf{A}$ is the transition probability matrix of a random walk on graph defined by matrix $\mathbf{A}$. For m and h nodes in the graph, we denote p_{mh} the probability transition from m to h, namely equal to $\frac{a_{m,h}}{D_m}$. We have the identity on the mth component of $\mathbf{x}_{k+1}$,

$$\mathbf{x}_{k+1}(m) = \mathbf{x}_k(m) + i\varepsilon D_m\mathbb{E}(\mathbf{x}_k(\xi_m)), \tag{9}$$

with ξ_m as a randomly picked neighbour of node-m. Similar local updating rule applies for other nodes as well. The expectation in (9) can be calculated using Monte Carlo approach. Interestingly we observe in simulation results that small number of iterations provides good accuracy for the Monte Carlo simulation. The algorithm is iterated several times and then averaged in order to smooth the variations due to the random selection of neighbor. The algorithm is as follows:

Algorithm IV.3: COMPLEXGOSSIPING$(m, \mathbf{b}_0)$

$d \leftarrow 1$
$x(z) \leftarrow \mathbf{b}_0(m)$
while $(d \leq d_{\max})$
do
- **for each** $j \in \mathcal{N}(m)$
  - **do**
    - **if** REQUESTWAITINGFROM(j)
      - **then**
        - $d' \leftarrow$ REQUESTLEVEL(j)
        - **if** $(d' < d)$ **then**
          - SEND$(j, i\varepsilon$ COEFF$(x(z), z^{d'}))$
- $\xi_m \leftarrow$ RANDOMNEIGHBOR(m)
- SENDREQUEST$(\xi_m, d-1)$
- $x(z) \leftarrow x(z) +$
  z^d COEFF$(x(z), z^{d-1}) + D_m z^d$RECEIVE$(\xi_m)$
- $d \leftarrow d + 1$

return $(x(z))$

The procedure COEFF$(x(z), z^d)$ returns coefficient of the term z^d in $x(z)$. REQUESTWAITINGFROM(j) is a boolean valued procedure indicating if a request is waiting from node j, REQUESTLEVEL(j) is the degree of the coefficient required by node j and SENDREQUEST(j, d) is the procedure which sends to node j the request to fetch the coefficient of degree d. Notice that with the procedure SENDREQUEST, the local process will wait until the neighbor ξ_m will respond. Though this will introduce delays, it is limited to waiting time of a single neighbor to deliver. But such a scheme will avoid the use of synchronous clock in the system.

The gossiping algorithm introduced here is a variant of the diffusion algorithms mentioned in the previous section, i.e., instead of taking fluid from all the neighbors, the gossiping technique collects the fluid from only one random neighbor. The algorithm can also be extended to order-2 and order-4.

V. QUANTUM RANDOM WALK TECHNIQUES

We have described in the previous section methods to solve a discretisation of equation

$$\frac{\partial}{\partial t}\mathbf{b}_t = i\mathbf{A}\mathbf{b}_t.$$

This equation is very similar to the classic Schrödinger equation

$$i\hbar\frac{\partial}{\partial t}\mathbf{\Psi}_t = \mathcal{H}\mathbf{\Psi}_t,$$

where $\mathbf{\Psi}_t$ is the wave function represented by a vector of complex numbers, $\hbar$ is the Planck constant and $\mathcal{H}$ is the Hamiltonian of the system (which is a Hermitian matrix). If $\mathcal{H}$ is formed from graph matrices like $\mathbf{A}$, we have the wave function of a particle wandering on a network, which is called quantum random walk. In this section we will first show that the evolution of the wave function of a quantum random walk can be simulated via several parallel classic random walks with complex reward and use it in order to extract the spectrum information. Then we propose a pure quantum random walk algorithm.

A. Quantum random walk algorithm via classic random walks

This is a distributed technique in which the nodes require only local information, but with reduced complexity compared to Complex gossiping as only one node communicates with another neighbor node at each cycle, for each random walks.

Assume we consider order-1 approximation. At each iteration, we run the following algorithm:

1) Each node-m has a vector of length $d_{\max}+1$ (mth row of $\mathbf{X}$) with $\mathbf{x}_\ell(m)$ representing mth entry of $(\mathbf{I}+i\varepsilon\mathbf{A})^\ell\mathbf{b}_0$.
2) A set of classic random walks start from each node initially. All the random walks associated to node-m has the initial fluid $\mathbf{b}_0(m)$.
3) All the random walks start moving to one of their neighbor, which is selected uniformly at random.
4) At time step $k>2$, at each node m, only the first arrived random walk wins and the fluid carried by this random walk (F) will be used for updating $\mathbf{x}_k(m)$. The fluid update will happen at node-m when it also receives the fluid from its own previous level $\mathbf{x}_{k-1}(m)$. Then
$$\mathbf{x}_k(m)=\mathbf{x}_{k-1}(m)+i\varepsilon D_m F.$$
After the update of $\mathbf{x}_k(m)$, all the random walks at node-m (the one which won and the ones which lost) will update their fluid as $\mathbf{x}_k(m)$. In case no random walks arrive at a particular node at level k after a certain time, it will take the fluid from its previous level $k-1$ only.
5) The random walks keep moving and all of them stop when the time step k reaches $d_{\max}$.

Like the gossiping technique, the above algorithm is also iterated several times to get the desired convergence. The pseudocode of a more generalized technique is given in Algorithm V.1.

The procedure MOVEWAITING() is the set of the notification of random walk moves to node m, MOVELEVEL(M) is the level of the move notification M, and SENDMOVE(j,d,c) is the process of moving random walk to the level d at node j carrying the value c, and MOVEVALUE(M) is the value carried the random walk notification M.

B. Pure Quantum Random Walk algorithm

In this section, we elaborate the connection of our technique to quantum computing. We make use of quantum random walks (QRW) on graphs to massively distribute our spectrum calculations. Compared to the classic random walks, in which walker can exist in only one state at a time, QRW moves simultaneously over all the states by exploiting the idea of superposition in quantum mechanical systems [12]. The quantum mechanical system we assume to perform the spectral decomposition is described as follows.

Algorithm V.1: PARALLELRANDOMWALK($m,\mathbf{b}_0$)

```
d ← 0
x(z) ← b_0(m)
while (d ≤ d_max)
do { for each M ∈ MOVEWAITING()
     do { d' ← MOVELEVEL(M)
          if (d' > d) then
          { for d'' ← d+1 to d'
            do {x(z) ← x(z) + z^{d''} x(z)}
            d ← d'
            x(z) ← x(z) + z^d D_m MOVEVALUE(M)
          ξ_m ← RANDOMNEIGHBOR(m)
          SENDMOVE(ξ_m, d+1, εi COEFF(x(z), z^d)) }
return (x(z))
```

We focus on continuous time QRW on a graph in which the position of the walker depends on the Hamiltonian $\mathcal{H}$ which is taken as $\mathcal{H}=\mathbf{A}+\Delta\mathbf{I}$ where Δ is the maximum degree. The translation by Δ is necessary in order to make the energy values positive and will only cost a translation of the spectrum.

The walker is represented by a qubit made out of a chain of L atoms where each of them is spin oriented, either up (1) or down (0). Therefore the qubit has a capacity of L bits and can describe all the 2^L binary codewords of length L, corresponding to integers between 0 and $d_{\max}-1$, with $d_{\max}=2^L$. For $0\le k\le d_{\max}-1$, we denote $|k\rangle$ the state of the qubit corresponding to the integer k. At initialization, state of the qubit is uniform on all codewords: $(1/\sqrt{d_{\max}})\sum_{k=0}^{d_{\max}-1}|k\rangle$.

We consider a *splitting chain* made of L polarized gates such that the state $|k\rangle$ is delayed by $k\varepsilon$ time units. To achieve this, for $0\le r\le L$ the rth gate let the qubit with spin 0 pass or delay the spin 1 by $2^r\varepsilon$ via a delay line. At the end of *splitting chain*, the qubit is *injected* in the graph on each node ℓ. The wave function $\mathbf{\Psi}_t^{d_{\max}}$ of the walker at time t, a complex valued vector on the vertices of the graph, formed from such a process satisfies

$$\mathbf{\Psi}_t^{d_{\max}}=\frac{1}{\sqrt{d_{\max}}}\sum_{k=0}^{d_{\max}-1}e^{i(t-k\varepsilon)\mathcal{H}}\mathbf{\Psi}_0|k\rangle. \tag{10}$$

Here $\mathbf{\Psi}_0$ is the wave function of the walker when it is inserted in the graph, for instance when the walker is introduced on node ℓ: $\mathbf{\Psi}_0(m)=\delta_\ell(m)$ for any node m. At time $t\ge\varepsilon d_{\max}$, we take the qubits on any node m, $\mathbf{\Psi}_t^{d_{\max}}(m)$. We apply on it the quantum Fourier transform (QFT) as described in the Shor's algorithm [13]. Essentially this implements a discrete Fourier transform (DFT) approximation of the continuous Fourier transform in (3). The QFT outputs the DFT coefficients $\{y_k\}$ as $\sum_{k=0}^{d_{\max}-1}y_k|k\rangle$. During measurement of the QRW, kth

index is obtained with probability $|y_k|^2$, and this will be an eigenvalue point shifted by Δ (along with appropriate scaling of frequencies in discrete domain to continuous domain). Thus multiple run-measurement process of the QRW produces the different eigenvalues. The empirical probability of $\lambda_j + \Delta$ can be calculated via measurements and will be proportional to $|\mathbf{u}_j(m)|^2$.

The *splitting chain* technique with *injection* into the original graph can be further modified in order to introduce the Gaussian smoothing (4) which improves the accuracy of the spectral decomposition, but is not described here. Moreover, $\mathbf{\Psi}_0$ could be $(1/\sqrt{2})(\delta_{\ell'}(m) + \delta_\ell(m))$ so that $\text{sign}(\mathbf{u}_j(\ell)\mathbf{u}_j(\ell'))$ can be revealed.

VI. Parameter Analysis and Tuning

A. Order of Convergence

There are three factors governing the convergence rate:

1) Riemann integral approximation to left-hand side of the integral (4):

$$\varepsilon\Re\left(\mathbf{I} + 2\sum_{\ell=1}^{d_{\max}} e^{i\ell\varepsilon\mathbf{A}}\mathbf{b}_0 e^{-i\ell\varepsilon\theta}e^{-\ell^2\varepsilon^2 v/2}\right)$$
$$= \int_{-T}^{+T} e^{i\mathbf{A}t}\mathbf{b}_0 e^{-t^2v/2}e^{-it\theta}dt + O\left(\lambda_1\varepsilon^2 d_{\max}\|\mathbf{b}_0\|\right), \quad (11)$$

where $T = \varepsilon d_{\max}$. The factor $\lambda_1\|\mathbf{b}_0\|$ is an upperbound of the derivative of $e^{it\mathbf{A}}\mathbf{b}_0$. Notice that λ_1 can in turn be upper bounded by Δ the maximum weighted degree of the graph.

2) Approximating $e^{i\ell\varepsilon\mathbf{A}}$ by r-order Runge-Kutta method (with the equivalent expression (8)), we get

$$\varepsilon\Re\left(\mathbf{I} + 2\sum_{\ell=1}^{d_{\max}} e^{-i\ell\varepsilon\theta}e^{-\ell^2\varepsilon^2 v/2}\mathbf{x}_l\mathbf{b}_0\right)$$
$$= \varepsilon\Re\left(\mathbf{I} + 2\sum_{\ell=1}^{d_{\max}} e^{i\ell\varepsilon\mathbf{A}}\mathbf{b}_0 e^{-i\ell\varepsilon\theta}e^{-\ell^2\varepsilon^2 v/2}\right)$$
$$+O\left(\lambda_1\varepsilon^{r+2}(d_{\max})^2\|\mathbf{b}_0\|\right).$$

3) Error in truncated integral:

$$\int_{-T}^{T} e^{i\mathbf{A}t}\mathbf{b}_0 e^{-t^2v/2}e^{-it\theta}dt$$
$$= \int_{-\infty}^{\infty} e^{i\mathbf{A}t}\mathbf{b}_0 e^{-t^2v/2}e^{-it\theta}dt$$
$$+O\left(\sqrt{\frac{2\pi}{v}}\text{erf}\left(\sqrt{\frac{v}{2}}\varepsilon d_{\max}\right)\|\mathbf{b}_0\|\right)$$

where $\text{erf}(x)$ indicates the Gaussian error function.

It can be seen that convergence rate in (11) dominates. In addition, we should have $\varepsilon d_{\max}$ large while $\varepsilon^2 d_{\max}$ is small.

B. Choice of initial vector $\mathbf{b}_0$ and algebraic multiplicity

In order to compute the approximation (6) in a distributed manner in the network itself, each node select its own component of the initial vector $\mathbf{b}_0$. The components could be all equal to 1, $\mathbf{b}_0 = \mathbf{1}$, but in this case it may be orthogonal to eigenvectors. Indeed if the graph is regular then $\mathbf{1}$ is colinear to the main eigenvector, and therefore orthogonal to the other eigenvectors. To circumvent this problem, each node can randomly select a component so that the probability it results into an orthogonal vector be negligible.

Another interesting option is to select $\mathbf{b}_0$ as a vector of i.i.d. Gaussian random variables with zero mean and variance w. In this case

$$\mathbb{E}[\mathbf{b}_0^\intercal\mathbf{f}(\theta)] = w\sum_{j=1}^{n}\sqrt{\frac{2\pi}{v}}\exp(-\frac{(\lambda_j-\theta)^2}{2v}).$$

The left-hand side of the above expression can be calculated by Monte Carlo techniques and this will give equal peaks for all the eigenvalues. Hence, the eigenvalues and subsequently the eigenvector components can be deduced with very good accuracy. We call this technique *trace-technique*, since this method indeed is like taking trace of the original approximation matrix, right-hand side of (4).

In case of algebraic multiplicity k of a particular eigenvalue λ, *trace-technique* will give the peaks approximately as $kw\sqrt{2\pi/v}$ and $\mathbb{E}[\mathbf{b}_0\mathbf{f}_{\theta=\lambda}^\intercal]$ will be the projection matrix on eigenspace of λ, i.e., $\mathbb{E}[\mathbf{b}_0\mathbf{f}_{\theta=\lambda}^\intercal] = \sqrt{\frac{2\pi}{v}}w\sum_{\ell=1}^{k}\mathbf{u}_\ell\mathbf{u}_\ell^\intercal$.

C. Choice of parameters

1) Parameter v: The selection of v is governed by the fact that we need to discern distinct eigenvalues by locating the peaks in the spectral plot. When we need to locate top k eigenvalues, with 99.7% of the Gaussian areas not overlapping, $6v < \min_{1\le i\le k-1}|\lambda_i - \lambda_{i+1}|$. In general for a large graph, a sufficiently lower value of v will be enough to distinguish between the peaks. For larger $\varepsilon d_{\max}$, variation in v will not affect the plot apart from the resolution of the peaks, but for lower $\varepsilon d_{\max}$, lower value of v creates spurious ripples across the plot.

2) Parameter ε: From Fourier analysis literature, it is known that the discretization of the integral in (4) with ε leads to multiple copies of the spectrum. According to sampling theorem, in order to avoid aliasing among them, the choice of ε is controlled by the net bandwidth B of the spectrum as $\epsilon < 1/(2B)$. Here $B \approx |\lambda_1 - \lambda_n| + 6v$, including 99.7% of the Gaussian variations associated to λ_1 and λ_n. We have, $|\lambda_1 - \lambda_n| < 2\lambda_1 < 2\Delta$, with Δ being the maximum degree of the graph and a proof of the last inequality can be found in [14]. Hence choosing $\varepsilon < 1/(4\Delta + 12v)$ will ensure that sampling theorem is satisfied.

3) Parameter $d_{\max}$: From Section III-A, T should $\to\infty$ and $T\varepsilon\to 0$. This implies as $d_{\max}\to\infty$, ϵ should be chosen as $1/d_{\max} < \epsilon < 1/\sqrt{d_{\max}}$, asymptotically.

Scalability: Combining the argument behind the selection of ε and $d_{\max}$ (with $d_{\max}$ chosen accordingly by fixing ε), we can say that $d_{\max}$ depends on maximum degree Δ, not on the number of nodes n. Thus, we expect that our approach is highly scalable.

VII. Numerical Results

We demonstrate the algorithms described above with numerical studies on real-world networks. First, in order to compare

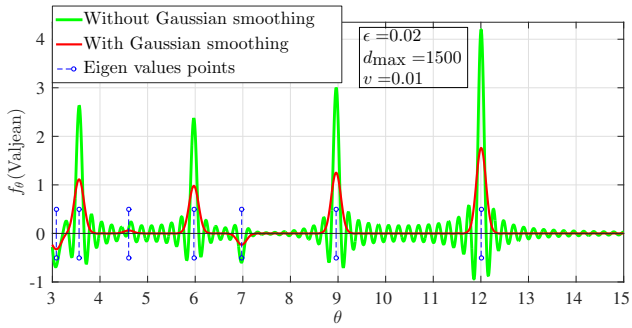

Fig. 2. Les Misérables graph: with and without Gaussian smoothing

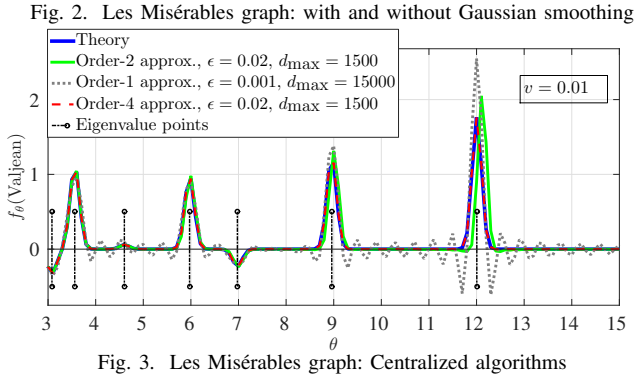

Fig. 3. Les Misérables graph: Centralized algorithms

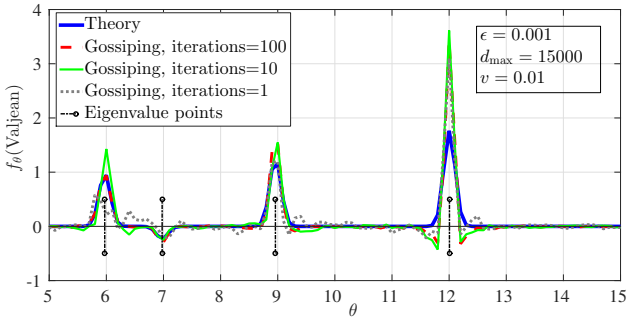

Fig. 4. Les Misérables graph: Monte Carlo gossiping

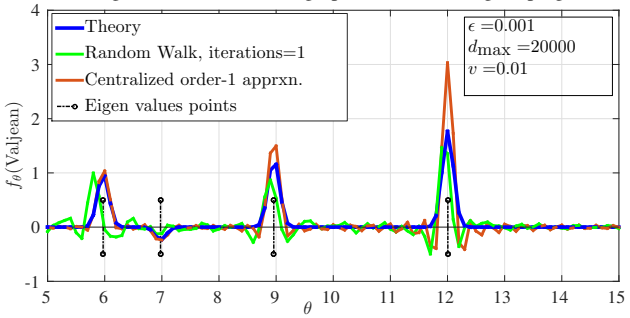

Fig. 5. Les Misérables graph: random walk

and show the effectiveness of the different techniques, we consider Les Misérables network, graph of characters of the novel Les Misérables. Later, we examine Enron email network, the email communication network among Enron employees and DBLP network which is a co-authorship network from the DBLP computer science bibilography. We have chosen these datasets so that their sizes differ in orders of magnitude. The datasets are taken from [15] where several parameters of the datasets can be found.

In the following simulation results, in order to show the decomposition at the node level, we consider one particular node in each of the networks examined. We select such a node as one of the top 2% highest degree nodes in the network. In the figures, we have also shown the actual eigenvalue points, which are cross-checked with *eigs* function in Matlab and *adjacency_spectrum* function in the Python module Networkx.

Les Misérables network

In Les Misérables network, nodes are the characters and edges are formed if two characters appear in the same chapter. The number of nodes is 77 and number of edges is 254. We look at the spectral plot in a specific node called Valjean, a character in the associated novel.

We first show in Figure 2 the smoothing effect Gaussian term brings in the finite sum approximation (6). Indeed, Gaussian smoothing technique eliminates spurious picks. Different centralized algorithms are shown in Figure 3. As shown in the figure, the order-1 algorithm takes ten times more $d_{\max}$ than order-2 and order-4. This is mainly because lower ε is needed for order-1 due to slower convergence and this in turn leads to higher $d_{\max}$. We also observe that order-4 matches nearly perfectly with the theoretical values.

The numerical results for the Monte Carlo gossiping technique explained in Section IV-B is shown in Figure 4. Interestingly, even one iteration of Monte Carlo averaging provides sufficient information about the eigenvalues and it can be observed that smaller number of iterations are needed for practical convergence of this algorithm.

Figure 5 presents the random walk implementation of the Monte Carlo gossiping. Here we used only one Monte Carlo averaging and four random walks are started from each node. We notice again that just with one iteration, it shows very good performance with respect to the order-1 centralized algorithm.

Enron email network

The nodes in this network are the email ID's of the employees in Enron and the edges are formed when two employees communicated through email. Since the graph is not connected, we take the largest connected component with $33,696$ nodes and $180,811$ edges. The node considered is the highest degree node in that component.

Figure 6 shows the complex diffusion with order-4 calculation and we find it is exactly matching with the theory. In Figure 7, the gossiping approach is shown which also performs very well.

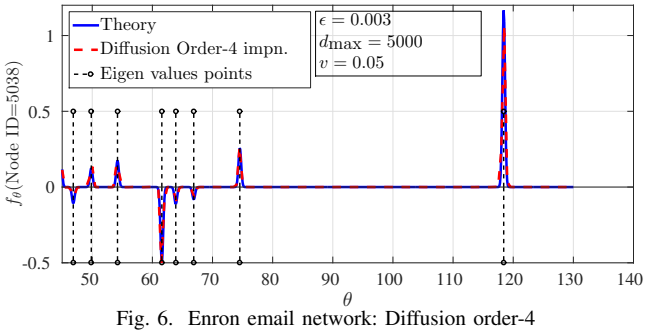

Fig. 6. Enron email network: Diffusion order-4

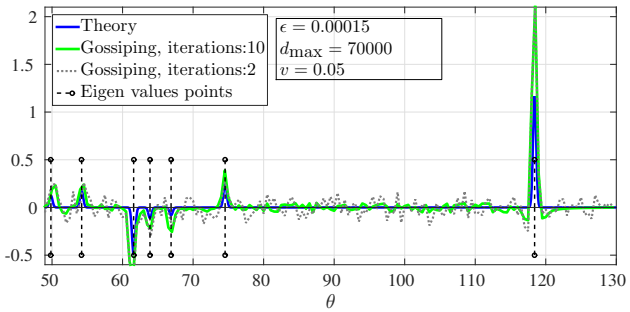

Fig. 7. Enron email network: Gossiping

DBLP network

We provide one more example, DBLP computer science network, which is ten times larger than Enron email network. It is made out of co-authorship network where the nodes are the authors and the edges between two authors are formed if they have written at least one paper together. The number of nodes is $317,080$ and number of edges is $1,049,866$. We consider the node with ID 6737, which has degree in top 2% of the network.

In order to show the effectiveness of the scalability argument provided in Section VI-C for higher order approximations, the diffusion algorithm with order-4 is shown in Figure 8. The algorithm is matching very well with the theory. The $d_{\max}$ for order-4 diffusion in Enron-email and DBLP networks for a good match with theory, is around 5000. This is in tune with our finding in Section VI-C that $d_{\max}$ mainly depends on $|\lambda_1 - \lambda_n|$. In case of Enron-email $|\lambda_1 - \lambda_n| = 159.72$ and in DBLP network $|\lambda_1 - \lambda_n| = 132.42$. They are at least in the same order.

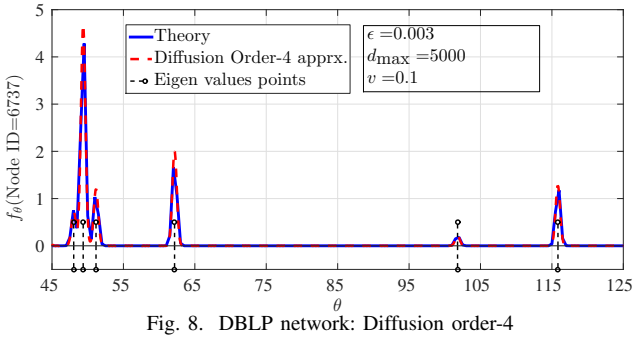

Fig. 8. DBLP network: Diffusion order-4

VIII. Asymmetric matrices

Here we check the applicability of the proposed algorithms to asymmetric matrices. In this case some of the eigenvalues will be complex with the exception of the main eigenvalue which will be still real. The right and left eigenvectors will be different, $\mathbf{A} = \sum_j \lambda_j \mathbf{u}_j \mathbf{v}_j^\intercal$, and consequently,

$$\begin{aligned}\int_{-\infty}^{\infty} & e^{it\mathbf{A}} e^{-vt^2/2 - it\theta} dt \\ &= \sum_j \mathbf{u}_j \mathbf{v}_j^\intercal \exp\left(-\frac{(\Re(\lambda_j) - \theta)^2}{2v}\right) \\ &\quad \times \exp\left(\frac{(\Im(\lambda_j)^2}{2v}\right) \cos\left(\Im(\lambda_j(\Re(\lambda_j) - \theta))\right).\end{aligned}$$

This implies that the peaks on the eigenvalues have value $\exp\left(\frac{(\Im(\lambda_j)^2}{2v}\right)$ which may be extremely large and too discrepant between eigenvalues. Nevertheless the complex diffusion and quantum random walks could be applied if the eigenvalues with non zero imaginary part are clearly separated from the pure real eigenvalues. This is the case when using the non-backtracking matrix based spectral clustering of [7], where it is proved to be more efficient than the classic spectral clustering provided the graph satisfies certain properties.

IX. Conclusion

We have proposed some new approaches for distributed spectral decomposition of graph matrices. The fourier analysis of complex exponential of the graph matrices, with Gaussian smoothing, turns out to have a simple interpretation of the spectrum in terms of peaks at eigenvalue points. This exposition leads us to develop two efficient distributed algorithms based on "complex power iterations": complex diffusion if the nodes can collect data from all the neighbors and complex gossiping when data is taken from only one random neighbor. Later we detailed the connection of complex exponential of graph matrices to quantum random walk techniques. We derived the order of convergence of algorithms and it is found that the algorithms are scalable in proportion to the maximum degree of the graph. Numerical simulations on real-world networks of varying order of magnitude in size, showed the effectiveness and scalability of our various algorithms.

References

[1] C. Tsourakakis, "Fast counting of triangles in large real networks without counting: Algorithms and laws," in *ICDM*, Dec 2008, pp. 608–617.

[2] D. Kempe and F. McSherry, "A decentralized algorithm for spectral analysis," in *STOC*, 2004, pp. 561–568.

[3] D. Easley and J. Kleinberg, *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

[4] B. A. Prakash, A. Sridharan, M. Seshadri, S. Machiraju, and C. Faloutsos, "Eigenspokes: Surprising patterns and scalable community chipping in large graphs," in *PAKDD*, 2010, pp. 435–448.

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

[6] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 8, pp. 888–905, 2000.

[7] C. Bordenave, M. Lelarge, and L. Massoulié, "Non-backtracking spectrum of random graphs: community detection and non-regular ramanujan graphs," in *FOCS*, Oct 2015, pp. 1347–1357.

[8] T. Sahai, A. Speranzon, and A. Banaszuk, "Hearing the clusters of a graph: A distributed algorithm," *Automatica*, vol. 48, pp. 15–24, 2012.

[9] M. Franceschelli, A. Gasparri, A. Giua, and C. Seatzu, "Decentralized estimation of laplacian eigenvalues in multi-agent systems," *Automatica*, vol. 49, no. 4, pp. 1031–1036, 2013.

[10] V. S. Borkar, R. Makhijani, and R. Sundaresan, "Asynchronous gossip for averaging and spectral ranking," *IEEE J. Sel. Areas Commun.*, vol. 8, no. 4, pp. 703–716, 2014.

[11] M. Tenenbaum and H. Pollard, *Ordinary Differential Equations*. Dover Publications, Oct 1985.

[12] S. E. Venegas-Andraca, "Quantum walks for computer scientists," *Synthesis Lectures on Quantum Computing*, vol. 1, no. 1, pp. 1–119, 2008.

[13] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM journal on computing*, vol. 26, no. 5, pp. 1484–1509, 1997.

[14] L. Lovász, "Eigenvalues of graphs," 2007, unpublished. [Online]. Available: http://www.cs.elte.hu/~lovasz/eigenvals-x.pdf

[15] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," http://snap.stanford.edu/data, Jun. 2014.