Int J Parallel Prog (2015) 43:1028–1053
DOI 10.1007/s10766-014-0344-3

PageRank Computation Using a Multiple Implicitly Restarted Arnoldi Method for Modeling Epidemic Spread

Zifan Liu · Nahid Emad · Soufian Ben Amor · Michel Lamure

Received: 5 January 2014 / Accepted: 24 October 2014 / Published online: 13 November 2014

Abstract A parallel implementation based on implicitly restarted Arnoldi method (MIRAM) is proposed for calculating dominant eigenpair of stochastic matrices derived from very large real networks. Their high damping factor makes many existing algorithms less efficient, while MIRAM could be promising. Also, we apply this method in an epidemic application. We describe in this paper a stochastic model based on PageRank to simulate the epidemic spread, where a PageRank-like infection vector is calculated by MIRAM to help establish efficient vaccination strategy. MIRAM is implemented within the framework of Trilinos, targeting big data and sparse matrices representing scale-free networks, also known as power law networks. Hypergraph partitioning approach is employed to minimize the communication overhead. The algorithm is tested on a nation wide cluster of clusters Grid5000. Experiments on very large networks such as *twitter* and *yahoo* with over 1 billion nodes are conducted. With our parallel implementation, a speedup of 27× is met compared to the sequential solver.

Z. Liu (✉) · N. Emad
Maison de la Simulation, USR 3441, Building 565, 91191 Gif-sur-Yvette, France
e-mail: zifan.liu@prism.uvsq.fr

Z. Liu · N. Emad · S. B. Amor
PRiSM Laboratory, UMR 8144, University of Versailles, 45 avenue des Etats-Unis, 78035 Versailles, France
e-mail: nahid.emad@prism.uvsq.fr

S. B. Amor
e-mail: soufian.benamor@uvsq.fr

M. Lamure
Santé, Individu, Société, EAM 4129, University of Lyon 1, University of Lyon, Campus Laennec, 11 rue Guillaume Paradin, 69372 Lyon, France
e-mail: michel.lamure@univ-lyon1.fr

Keywords Epidemic · PageRank · Scale free networks · Power law · IRAM · Big data · Hypergraph partitioning

1 Introduction

Dynamic complex systems appear in many areas, such as physics, biology, and computer networks. In the domain of health research, quick response and effective control of widely spreading health crises stay a big challenge for public health officials as well as scientists. In order to simulate the epidemic spread, such as H1N1 outbreak in France, traditional models need hundreds of experiments and compute the expected outcome by averaging. In addition, these experiments should be adjusted on a daily basis during the initial outbreak.

To answer urgent requests during the beginning phase of outbreak, an eigenvalue model is proposed in [1]. In this model, a PageRank-like Infection vector is calculated, which could help health officials decide the relative importance of different agents or groups of agents in a population facing an epidemic. Concerning the computational aspect, the difficulty for computing PageRank arises from the size of network and the big damping factor. Due to similar characteristics, this problem is also encountered in other real applications. In the present paper, we study the computation of PageRank within this context.

PageRank citation ranking was initially introduced in [2] to bring order to the Web. A page has high rank if the sum of the ranks of its inlinks is high. In other words, rank is propagated through links. To use mathematical formalism, we look for a PageRank vector x, which is the dominant eigenvector of the *Google matrix*,

$$A = \alpha P + (1 - \alpha) v e^T, \quad 0 \le \alpha < 1 \tag{1}$$

where the matrix P is a column stochastic matrix, called *transition matrix*, representing the outlink structure of the Web, e is the vector $(1, \dots, 1)^T$, α is called the damping factor, and the vector v is the teleportation vector, which ensures the uniqueness of the PageRank vector. Noticing that the virus has a small probability $(1 - \alpha)$ to jump from any individual to any other individual in a social graph. This would happen, for example, when an infected person (virus carrier) meets and passes the disease to someone outside his normal contacts . This event happens rarely so that the damping factor α is very close to 1 in application on epidemics. A difficulty in PageRank model is caused by the existence of dangling nodes [3]. These nodes will result in one or more columns of zeros in transition matrix P. Several ideas have been proposed to deal with this problem. A good reference can be found in section 8.4 from the Langville and Meyer book on PageRank [4]. We will continue our discussion about this issue within epidemic application in Sect. 2.3.

Many algorithms have been proposed for computing PageRank [5]. In this paper, we focus on Arnoldi-type algorithms. The method proposed by Golub and Greif combines Arnoldi process and singular value decomposition to compute PageRank [6]. Wu and Wei use an extrapolation procedure to provide increasingly better initial guess to Arnoldi iteration [7]. Their idea is to periodically subtract off estimates of the nonprincipal eigenvectors. Authors of [8] demonstrated the fast convergence of Krylov subspace methods for the PageRank linear systems. A comparison of the eigenproblem viewpoint and the linear system viewpoint over the PageRank problem can be found in [9]. The idea to use GMRES method for PageRank is further explored in [10].

In real applications, computation of PageRank has three challenging aspects. First, the matrices involved are very large and rely on a parallel sparse matrix-vector product (MVP) kernel. Suppose z is a vector of p-norm 1, Az can be written as $\alpha Pz + (1 - \alpha)v(e^T z)$ where $e^T z$ is a scalar. So the MVP of A is expressed as MVP of a sparse matrix P plus a vector. Otherwise, any direct computation using A will be bottlenecked by memory requirement for large networks. In fact, the Google matrix A becomes a dense matrix due to the part $(1 - \alpha)ve^T$. For the above reason, algorithms based on MVP might be advantageous. Secondly, the damping factor α generally needs to take values approaching 1. For example, in the model of epidemic spread, the virus has the probability $1-\alpha$ to jump randomly from an infected individual to any other individual through some unusual contact. Intuitively, this event rarely happens, and for disease spread, α must be very close to 1. This is an argument in favor of using Arnoldi-type methods, as opposed to the Power method. In fact, it can be proved that the second largest eigenvalue of matrix A is very close to α [11]. For big α, the second largest eigenvalue will be close to the dominant eigenvalue (that equals to 1) of A, which will slow down the convergence of the Power method. Last but not least, the network is very large and of scale-free structure. Vectors used in the computation should be stored in parallel among p processors, because they could be larger than any single processor could handle. For example, take $n = 10^9$ for a network, the corresponding PageRank vector contains 10^9 entries, which could take as much as $16 * 10^9$ bytes $\approx$ 15 GB of memory in complex double precision. This issue of storage requirement is worsened when using Krylov subspace methods. For instance, we can consider the parameters $n = 10^{11}$ and $m = 10^3$, where n is the size of the problem and m is the projection subspace size. Then, each iteration requires 10 Peta bytes memory space to maintain the orthogonal basis.

In [12], we proposed a parallel algorithm for implicitly restarted Arnoldi method (IRAM). We illustrate here multiple implicitly restarted Arnoldi method (MIRAM), introduced in [13] and propose a parallel version of it for PageRank computation. Besides, in the present paper, we add details of our epidemic model and discuss the issue of dangling individuals. Furthermore, MIRAM is implemented and executed in parallel within the framework of Trilinos [14]. We use hypergraph-based partitioning [15] to implement the sparse MVP kernel in this paper.

The model of parallelization used is so general that it could be employed for modern (possibly future) parallel architecture. According to our numerical results, we inspect that: the strategies proposed could accelerate the convergence of single IRAM for matrices derived from real applications. The remainder of this paper is organized as follows. In Sect. 2, we will briefly discuss how to use PageRank in models of epidemic spread, explaining the motivation for the present work. In Sect. 3, we will justify the use of MIRAM as computation method for PageRank and discuss its parallel algorithm. In Sect. 4, we will present a parallel MIRAM implementation, targeting very large and scale-free real networks. Section 5 is devoted to numerical experiments. Finally, future work along with the conclusion are discussed in Sect. 6.

2 Modeling of Epidemic Spread

2.1 Related Work

Computational epidemiology arises recently as an interdisciplinary area setting its sight on developing and using computer models to understand and control the spatiotemporal diffusion of disease within populations [16]. Here we focus on networked epidemiology, which seeks to understand the interplay between individual behaviour and dynamical process on social networks. In other words, this approach investigates the influence of the network topology on epidemic spread.

Agent based stochastic simulations (ASS) put all kinds of details into the model at an individual level. The simulation proceeds by establishing a set of rules to "guess" all its random parameters. This is the most common approach to simulate epidemics in a large population. Network Dynamics and Simulation Science Laboratory has proposed a parallel simulation model "Simdemics" [17,18], designed to scale to the entire United States (300 million people). A similar work can be found in [19], where an individual based influenza simulation model "FluTe" has been proposed. According to the numerical results of these studies, ASS are useful to help establish different pharmaceutical interventions as well as social distancing measures. Furthermore, from a computational point of view, ASS may easily scale up to simulate millions of people in a very efficient way. Nevertheless, one inconvenience of ASS approach is that its result depends on averaging over repeated runs, which could take large amount of time to ensure the quality.

An interesting work [20] has proved the close relationship between epidemic threshold of a network and the largest eigenvalue of network's adjacency matrix, which can subsume many known thresholds for special case graphs (scale-free, homogeneous, etc.). Rather than using ASS, this work employs matrix analysis to study the epidemic spread. In [21], authors have presented some empirical results on the potential usefulness of PageRank for establishing effective vaccination strategies. In [22], authors have proved that by using PageRank vectors, any infection will die out quickly and this process is independent of the size of the whole network. Although the idea to use PageRank vectors in epidemic studies is not new, we have not found any previous studies on discussing the computational aspects of PageRank-like epidemic models. Another novelty of our work lies in the application of very large real networks for such models.

2.2 PageRank-Like Model

In order to efficiently establish the vaccination strategy, we propose to make use of Google's PageRank model [2] by analogy. The common concept between PageRank model and epidemic model is the random walk. To a social network, an individual is what a web page to Internet. In PageRank model, the surfer (or walker) starts from one page, and then randomly selects one of its outlinks. Each page has two states: visited or not. The PageRank (importance) of a specific page represents the probability of the surfer presenting at this page. In our epidemic model, the virus could be viewed as

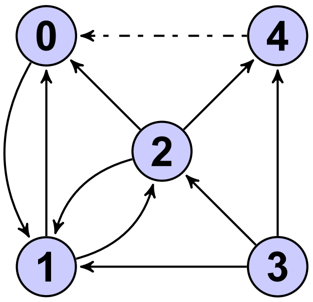

Fig. 1 Small social network of 5 individuals with individual 4 vaccinated

a walker and its propagation as a path that consists of a succession of random steps. Each individual has two states: infected or not. The "PageRank" (importance) of a specific individual represents the probability of virus reaching this person during the course of epidemic.

Thus, the formulation of epidemic spread can also be expressed as (1). Here, the matrix P is derived from social networks. A social network might be a directed network in the context of epidemic spread. For example, blood disease could only pass from donators to acceptors. As a result, the matrix P might be a non-symmetric matrix. According to (1), a virus has a small probability $(1-\alpha)$ to jump from any individual to any other one. This would happen, for example, when an infected person (virus carrier) meets someone outside his normal contacts. Considering the preferential attachment of scale-free networks [23], we could choose v to be proportional to individuals' degree and normalizes it by "1-norm". In short, there is a small probability that an individual establishes a new temporary link with someone who already has many links,

$$v = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}, v_i = \frac{d_i}{\sum_{i=1}^{i=n} d_i} \tag{2}$$

where d_j is the degree of individual j. For simplicity, entries of v is taken to be equal-probability in our experiments. Individuals with higher rank in PageRank-like infection vector are more likely to be infected. Therefore, vaccination strategies can be established accordingly.

For pharmaceutical interventions, our simulations start with cutting all of the out-links of vaccinated people. Then the propagation of virus proceeds by time step and stops if meeting these vaccinated individuals.

An example of 5 individuals is given in Fig. 1. The set of nodes (different individuals) is $V = \{0, 1, 2, 3, 4\}$ and the set of edges (different human contacts) is $E = \{0 \rightarrow 1, 1 \rightarrow 0, 1 \rightarrow 2, 2 \rightarrow 0, 2 \rightarrow 1, 2 \rightarrow 4, 3 \rightarrow 1, 3 \rightarrow 2, 3 \rightarrow 4, 4 \rightarrow 0\}$. It follows that $d = (1, 2, 3, 3, 1)$ and $\sum_{i=1}^{i=n} d_i = 10$. By taking $\alpha = 0.9$, we have:

$$
\begin{aligned}
A &= \alpha P + (1-\alpha) v e^T \\
&= 0.9 \times \begin{pmatrix} 0 & 1/2 & 1/3 & 0 & 1 \\ 1 & 0 & 1/3 & 1/3 & 0 \\ 0 & 1/2 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 0 \end{pmatrix} + 0.1 \times \begin{pmatrix} 1/10 \\ 2/10 \\ 3/10 \\ 3/10 \\ 1/10 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}^T \\
&= \begin{pmatrix} 0.01 & 0.46 & 0.31 & 0.01 & 0.91 \\ 0.92 & 0.02 & 0.32 & 0.32 & 0.02 \\ 0.03 & 0.48 & 0.03 & 0.33 & 0.03 \\ 0.03 & 0.03 & 0.03 & 0.03 & 0.03 \\ 0.01 & 0.01 & 0.31 & 0.31 & 0.01 \end{pmatrix}
\end{aligned}
\tag{3}
$$

If we vaccinate individual number 4 at the out break, the final transition matrix A becomes

$$
A = \begin{pmatrix} 0.01 & 0.46 & 0.31 & 0.01 & 0 \\ 0.92 & 0.02 & 0.32 & 0.32 & 0 \\ 0.03 & 0.48 & 0.03 & 0.33 & 0 \\ 0.03 & 0.03 & 0.03 & 0.03 & 0 \\ 0.01 & 0.01 & 0.31 & 0.31 & 0 \end{pmatrix} \tag{4}
$$

The column corresponding to this vaccinated person reduces to a $0-$column. And he will thus never be infected during the course of epidemic. Further insight is given by numerical experiments in Sect. 5.

In this model, we don't consider the important question of the time evolution of epidemic spread. This can be the subject of a future work.

2.3 Dangling Individuals

In our epidemic model, a decision must be made to deal with the "dangling individuals". There are several possibilities for their existence. For example, a person with innate immunity against certain disease, a person in quarantine after getting the disease, or someone who dies, etc. There is a difference between a dangling individual and a dangling web page. A dangling page contains no outlinks. However, in most cases, a dangling individual will still have some social connections. They are called dangling because they somehow cannot spread epidemic after getting it. In other words, the outlinks of the dangling individuals will be temporally disabled.

A dangling individual may have high PageRank as normal people. PageRank model computes the score for a person based on individuals that link to it, rather than based on features (such as dangling) of the person. Someone in contact with these dangling individuals contributes to their scores.

Research by the initial PageRank paper [2] indicates that the PageRank could be calculated by removing the links to dangling pages from the web network. However, theoretically this process might generate new dangling pages and iteratively remove all pages from the network. The work by Lee et al. lumps the dangling nodes together into one new state [24]. A rigorous justification for this approach can be found in [25].

The solution proposed in [26] adds artificial links to the dangling nodes. The idea is to force the transition matrix P to be stochastic.

We simply add an artificial loop with probability 1 to the dangling individuals. The disease will be "trapped" once reaching them. In this way, their corresponding diagonal elements in matrix P are filled with 1. This handling can be justified by similar arguments as shown in [26]. A virtual $(n+1)^{th}$ node is added to a n-sized social network. Let $\mathcal{C}$ denote the set of non-dangling nodes and $\mathcal{D}$ denote the set of dangling nodes. Suppose the size of $\mathcal{C}$ is $|\mathcal{C}| = m$, we have $|\mathcal{D}| = n - m$. Apart from the artificial loops added to dangling nodes, we add new edges $(i, n+1)$ for $i \in \mathcal{D}$ and $(n+1, i)$ for $i \in \mathcal{C}$. We construct a linear system as follows,

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \alpha C & 0 & e^{(1)}/m \\ \alpha D & \alpha I & 0 \\ (1-\alpha)(e^{(1)})^T & (1-\alpha)(e^{(2)})^T & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{5}$$

where, if d_j is the out degree of the node j, matrices $C(m \times m)$ and $D((n-m) \times m)$ are defined by:

$$c_{ij} = \begin{cases} d_j^{-1} & \text{if } i, j \in \mathcal{C} \\ 0 & \text{otherwise} \end{cases} \qquad d_{ji} = \begin{cases} d_j^{-1} & \text{if } i \in \mathcal{C}, j \in \mathcal{D} \\ 0 & \text{otherwise} \end{cases}$$

and $e^{(1)}, e^{(2)}$ are column vectors of 1's of conforming dimension.

Theorem 1 *The linear system (5) computes the PageRank for dangling nodes as well as non-dangling nodes in the network.*

Proof Rewrite the equation (5) as

$$x = \alpha C x + \frac{e^{(1)}}{m} z \tag{6}$$

$$y = \alpha D x + \alpha y \tag{7}$$

$$z = (1-\alpha)(e^{(1)})^T x + (1-\alpha)(e^{(2)})^T y \tag{8}$$

It follows,

$$z = [(1-\alpha)(e^{(1)})^T + \alpha (e^{(2)})^T D] x \tag{9}$$

We rewrite the equations (6) and (9) as

$$\begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} \alpha C & \frac{e^{(1)}}{m} \\ (1-\alpha)(e^{(1)})^T + \alpha (e^{(2)})^T D & 0 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix} \tag{10}$$

The matrix in the system (10) is a stochastic matrix, so that the vector x corresponds to the PageRank of non-dangling nodes $\mathcal{C}$. The PageRank for dangling nodes can then be computed by

$$y = \frac{\alpha}{1-\alpha} D x \tag{11}$$

□

Table 1 Notation used to describe the problem as well as the computing configuration

Symbol	Description
n	The matrix size
m	The size of Krylov subspace
k	The number of wanted eigenvalues
r	The number of shifts used in IRAM algorithm, $m = k + r$
nnz	The number of nonzero elements in A
A	$n \times n$ stochastic matrix
H_m	$m \times m$ projected upper Hessenberg matrix
W_m	$n \times m$ matrix, orthogonal basis in the Krylov subspace
$w[j]$	The j-th element of a vector w
f_m	Residual vector of length n
$\{\mu_i^{(m)}\}_{i=1}^m$	Eigenvalues of H_m (Ritz values of A)
$\{y_i^{(m)}\}_{i=1}^m$	Eigenvectors of H_m
$\{x_i^{(m)}\}_{i=1}^m$	Ritz vectors of A ($x_i^{(m)} = V_m y_i^{(m)}$)
$\|A\|_F$	The Frobenius norm of the matrix A
p	The number of processors available

Noticing that, by adding a virtual node, the initial PageRank problem (1) can be written as

$$\begin{pmatrix} x \\ z \end{pmatrix} = \begin{pmatrix} \alpha P & v \\ (1-\alpha)e^T & 0 \end{pmatrix} \begin{pmatrix} x \\ z \end{pmatrix}$$

which takes a similar form as (10).

3 Computation Method

An efficient solution to a very large sparse eigenvalue problem strongly depends on the proper choice of iterative methods. Our first objective is to choose the best method to calculate the dominant eigenvector in this context.

A lot of researches found that the damping factor α strongly affects the convergence of iterative methods [8,11]. So another special attention has been paid to investigate how the convergence of the proposed algorithm is influenced by this parameter. To facilitate our discussion, we list some of the notations used throughout this paper in Table 1.

3.1 Implicitly Restarted Arnoldi Method

The Arnoldi procedure approximates k eigenpairs of $A(n \times n)$ by those of a matrix of order m (the size of the subspace), where $k \leq m \ll n$. This method is promising for

PageRank computation. The matrices in question are derived from real networks and are of very large size n. The basic Arnoldi algorithm increases m until the dominant eigenvalues of A are found. For storage, in addition to A, the method keeps m vectors of length n and an $m \times m$ Hessenberg matrix, which gives $nm + O(m^2/2)$. For computation complexity, matrix-vector product costs about $2m * nnz + O(1)$ operations. The modified Gram-Schmidt procedure costs $m^2 n + O(1)$ operations. Since the size n of a real network may attain millions or even billions of nodes, increasing m causes both storage and computational overhead.

One way to overcome this difficulty is by restarting techniques. A variant, called IRAM [27–29], combines the implicitly shifted QR mechanism with an Arnoldi factorization and can be viewed as a truncated form of the implicitly shifted QR-iteration. As stated before, the most consuming part in Arnoldi procedure is the MVP due to the very large size of the Google matrix A. IRAM reduces the number of MVP needed from m to $r = m - k$. Here, $m - k$ is the number of shifts used in QR iterations [30]. The sequential algorithm of IRAM is described in [28]. In PageRank computation, to find the dominant eigenpair, we could choose as shifts the r eigenvalues with smallest moduli from the spectrum of H_m.

Concerning the stopping criteria, define the vector $x = W_m y$ to be a Ritz vector associated with Ritz value μ, where W_m is the matrix whose columns $w_1, w_2, \cdots, w_m$ constitute an orthogonal basis of the Krylov subspace $\mathbb{K}_m$. We have

$$\| AW_m y - W_m H_m y \| = \| Ax - \mu x \| = \| f_m \| \, \|e_m^T y\| \tag{12}$$

By using the backward error associated with IRAM [31], convergence test is: $\| f_m \| \, \|e_m^T y\| < \| A \|_F \, \epsilon$ where e_m is m^{th} vector of the canonical basis of $\mathbb{C}^m$ and ϵ is the tolerance.

3.2 Multiple Implicitly Restarted Arnoldi Method

For PageRank computation in real applications, IRAM should not be used naively. Due to the very large problem scale, subspace size m must be small to maintain the orthogonal basis W_m in memory. It is known that the eigen-information of interest may not appear when m is too small [28]. In addition, high damping factor results in clustered eigenvalues around the dominant one [11], which will slow down the convergence even further.

In IRAM, only the initial vector is used to improve the quality of the subspace during restarting cycles. The authors of [13] investigate the influence of the size of subspace. The idea is to make use of Arnoldi method to compute the Ritz elements of a large matrix A in a set of l nested Krylov subspaces. If the accuracy of the Ritz elements calculated is not satisfactory in any of these subspaces, the algorithm will select the one that contains the "best" current Ritz elements. Then a QR shifted algorithm will be applied to the $m_{best} \times m_{best}$ matrix which represents A in this $m_{best}-$size projection subspace. The leading $k \times k$ submatrix issued from QR algorithm concentrates the information corresponding to the desired eigenvalues. Arnoldi projections are then completed on nested Krylov subspaces starting with this submatrix. This method can

```
Require: $(A, W_{m_i}, H_{m_i}, f_{m_i})$ with $AW_{m_i} = W_{m_i}H_{m_i} + f_{m_i}e_{m_i}^T$ the $m_i$-steps Arnoldi factorization ($i = 1, 2, \dots, \ell$)
 1: for $it = 1, 2, \dots$ until convergence do
 2:    • Compute $\sigma(H_{m_i})$ and their associated eigenvectors (for $i = 1, \dots, \ell$)
       • Compute residual norms. If convergence in one of subspaces then stop.
 3:    Select the best result in these subspaces and the associated best subspace size $m_{best}$. Set $m = m_{best}, H_m = H_{m_{best}}$ and $W_m = W_{m_{best}}, f_m = f_{m_{best}}$.
 4:    Select set of $r = m - k$ shifts $(\mu_1^{(m)}, \dots, \mu_r^{(m)})$ based upon $\sigma(H_m)$ and set $q^T \leftarrow e_m^T$.
 5:    for $j = 1, \dots, r$ do
 6:       Factor $[Q_j, R_j] = qr(H_m - \mu_j^{(m)} I)$;
 7:       $H_m \leftarrow Q_j^H H_m Q_j$; $W_m \leftarrow W_m Q_j$,
 8:       $q \leftarrow q^H Q_j$
 9:    end for
10:    Set $f_k \leftarrow w_{k+1}H_m(k+1, k) + f_m q(k)$, $W_k \leftarrow W_m(1:n, 1:k)$, $H_k \leftarrow H_m(1:k, 1:k)$
11:    Beginning with the $k$-step Arnoldi factorization $AW_k = W_kH_k + f_ke_k^T$, apply $r_i = m_i - k$ additional steps of the Arnoldi process to obtain $\ell$ new $m_i$-step Arnoldi factorization $AW_{m_i} = W_{m_i}H_{m_i} + f_{m_i}e_{m_i}^T$ (for $i = 1, \dots, \ell$).
12: end for
```

Fig. 2 MIRAM algorithm

be considered as an IRAM with the largest subspace size, which uses eigen-information of some of its nested subspaces in order to update its restarting vector. In this paper, we focus on the parallelization of the method and present a parallel "multiple IRAM" algorithm (MIRAM).

The MIRAM procedure is described in Fig. 2. Let v be an initial vector and $M = (m_1, \cdots, m_\ell)$ be a set of ℓ subspace-sizes with $m_1 < \cdots < m_\ell$. We built ℓ Arnoldi projections on the subspaces $\mathbb{K}_{m_i,v}$ (for $i = 1, \dots, \ell$) where $\mathbb{K}_{m_1,v} \subset \mathbb{K}_{m_2,v} \subset \dots \subset \mathbb{K}_{m_\ell,v}$. We select then the subspace size m_{best} corresponding to the Arnoldi factorization, which offers the Ritz estimations for k desired eigenpairs. Similar to IRAM algorithm, $AW_{m_{best}} = W_{m_{best}}H_{m_{best}} + f_{m_{best}}e_{m_{best}}^T$ are then applied onto this Arnoldi factorization. As a result, only this factorization among the ℓ ones will be compressed with the eigen-information of interest. This is achieved using QR steps to apply $r_{best} = m_{best} - k$ shifts implicitly. The results after the shift process and equating the first k columns on both sides are

$$AW_{m_{best}}^+ = W_{m_{best}}^+ H_{m_{best}}^+ + f_{m_{best}}e_{m_{best}}^T Q \tag{13}$$

where $W_{m_{best}}^+ = W_{m_{best}}Q$, $H_{m_{best}}^+ = Q^T H_{m_{best}} Q$, and $Q = Q_1Q_2\cdots Q_{r_{best}}$ with Q_j the orthogonal matrix in QR process associated with the shift $\mu_j^{(m_{best})}$ and

$$AW_k^+ = W_k^+ H_k^+ + f_k^+ e_k^T, \tag{14}$$

with $f_k^+ = W_{m_{best}}^+ e_{k+1}\widehat{\beta}_k + f_{m_{best}}\sigma_k$ where $\widehat{\beta}_k = H_m^+(k+1, k)$ and $\sigma_k = Q(m, k)$. Beginning with this resulting k-step Arnoldi factorization, we apply $r_i = m_i - k$ additional steps of Arnoldi factorizations to obtain ℓ new projections onto the updated subspaces (for $i = 1, \cdots, \ell$). This allows again the projection onto ℓ nested subspaces with initial guess determined by the compressed k-step Arnoldi factorization,

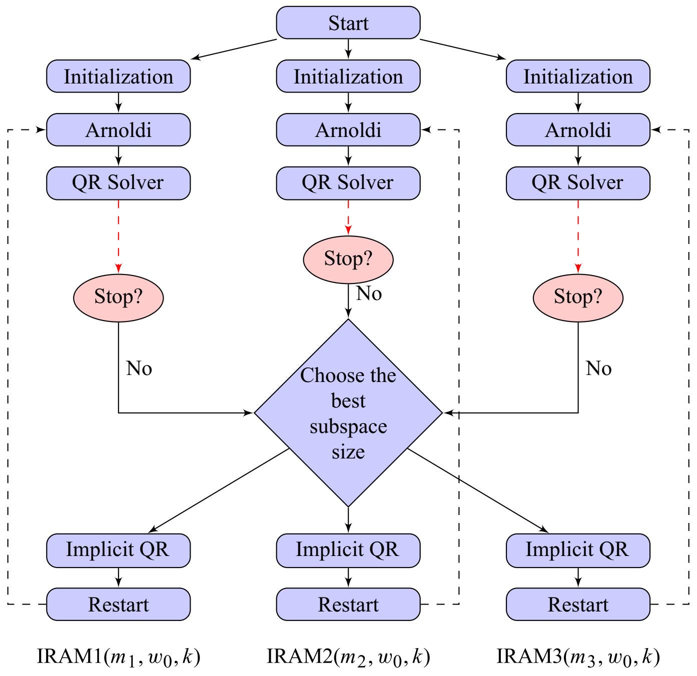

Fig. 3 The overview of MIRAM

issued from the QR shifted applied to $m_{best}-$step Arnoldi factorization. The multiple technique suggests the proliferation of subspace sizes and dynamically chooses one of them. Among different restarting steps, MIRAM could take advantage of the appearance of the eigen-information of interest, thanks to the larger subspace-sizes. In addition, the loss of orthogonality is slower compared to $IRAM(m_\ell)$ since some smaller subspace sizes are employed during restarting cycles of MIRAM. The overview of the algorithm is shown in Fig. 3.

It is important to notice that the communication of the eigen-information of interest of each IRAM process to other IRAM processes can be avoided. The idea is to run a single Arnoldi process proceeding across all processors and save the information whenever m reaches l different values. In other words, the steps 1 to 4 in Fig. 2 are duplicated across all processors. Furthermore, since f_m and W_m are distributed, the implicit QR iterations in steps 5 to 9 could be done locally on different processors as well.

Concerning the choice of parameter k, Stathopoulos et al. proposed in [32] a technique, called thick restart, where k_0 eigenpairs are needed, k ($k > k_0$) pairs are retained after each restart, and $r = m - k$ additional vectors are built. Some results of using thick restarting approach for the choice of parameter k is given in Sect. 5.

Concerning the time and space complexities of MIRAM versus that of IRAM. We assume that $m \ll n$ and let nrc be the number of restarting cycles excluding the initialization. The cost of IRAM in terms of matrix-vector products for nrc restarting cycles is $m + r \times (nrc - 1)$. Indeed, in the first cycle, the number of matrix-vector products is m and for each of the restarting cycles, the number of matrix-vector products is $r = m - k$. Noted that the cost of orthogonalization in a restarting cycle is $O(2 \times r \times n^2)$. When A is sparse and r is large, this cost of orthogonalization may dominant the computation. The space complexity of IRAM is $n^2 + O(m \times n)$.

Recall that m_ℓ is the maximum of $m_1, \ldots, m_\ell$ subspace sizes. The cost of MIRAM in terms of matrix-vector products is $m_\ell + r_\ell \times (nrc - 1)$. Still the cost of orthogonalization in Arnoldi process is $2 \times r_\ell \times n^2$. As a result, this cost of orthogonalization may dominant the computation when A is sparse and r_ℓ is large. The space complexity of MIRAM is $n^2 + O(n \times m_\ell)$.

We denote by CI and CM the time complexities of *one restarting cycle* of IRAM(m_ℓ) and MIRAM($m_1, \cdots, m_\ell$) respectively. Ignoring terms not including n and the cost of stopping criterion, these complexities can be given by $CI = \alpha + 2 \times n \times m_\ell^2$ and $CM = \alpha + 2 \times n \times \left[k \times (m_1 + \ldots + m_\ell) + m_{best}^2 - k \times m_{best}\right]$, where α is a common part in both algorithms. In the worst case for MIRAM, where $m_{best} = m_\ell$, $CM - CI = 2 \times n \times k \times (m_1 + \ldots + m_{\ell-1})$, which is positive. In the best case for MIRAM, where $m_{best} = m_1$, $C_M - C_I = 2 \times n \times \left[k \times (m_2 + \ldots + m_\ell) + m_1^2 - m_\ell^2\right]$, which could be positive or negative. Depending on the values of k and m_i, one restarting cycle of MIRAM could be less expensive than that of IRAM. To conclude, If MIRAM($m_1, \cdots, m_\ell$) uses m_{best} most of time, it will cause more computations than IRAM(m_ℓ). This is confirmed by our experiments in Sect. 5.

3.3 Scalable Sparse MVP for Scale-Free Networks Using Hypergraph Partitioning

The name "scale-free networks" comes from a project to map the World Wide Web in 1998, which has revealed a surprising fact that a few highly connected pages are essentially holding the World Wide Web together. Counting how many Web pages have a certain number of links showed that the degree distribution followed a power-law. Following researches observed many real networks that display similar phenomenon, among which are social networks.

When mining information from a network, eigenpairs of the various matrices that represent the network are used. Sparse Matrix vector product is the bottleneck of many existing eigensolvers for scale-free networks. This is especially true for any Krylov subspace method. There are a couple of approaches to improve the sparse MVP performance.

One way consists in balancing the workload: first, each processor should have at most $\lceil n/p \rceil$ columns; second, each processor should have roughly equal number of nonzero elements. We could use a simple heuristic method. Suppose there are p processors. We begin by sorting the columns according to their number of nonzero elements. Then from dense to sparse we attribute the column j to processor $i (i = 1, \ldots, p)$. After that, the rest sorted columns should be attributed one by one to the processor with the least number of nonzero elements each time. Another constraint is when a processor has $\lceil n/p \rceil$ columns, it should not be considered for attribution any more.

However, there are a couple of issues associated with this approach. First of all, the columns in each processor are usually not contiguous after redistribution, which will

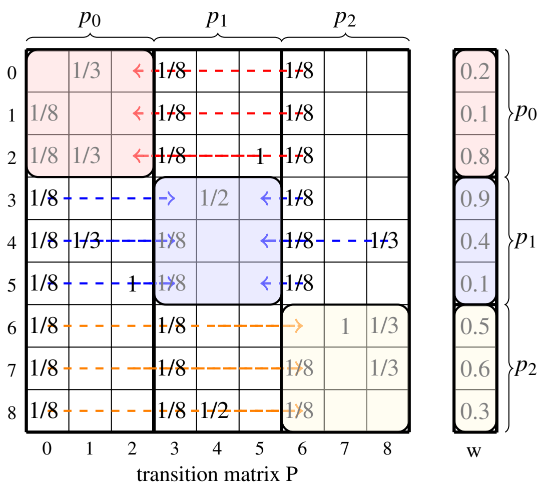

Fig. 4 An 1-D column-wise partitioning on 3 processors and its matrix vector multiplication

generate complex communication pattern while doing sparse MVP. A possible remedy is by reordering the nodes to make the columns in the same processor contiguous. The procedure above is equivalent to symmetrically permuting rows and columns of A. In other words, we construct a new matrix $B = T^T AT$, where T is the product of successive permutation matrices: $T = (T_1 \times T_2 \times \ldots)$. Then

$$Bu = \mu x \Rightarrow T^T ATx = \mu x \Rightarrow A(Tx) = \mu(Tx), \tag{15}$$

so that A and B have the same eigenvalues, and if x is an eigenvector of B, then $x' = Tx$ is an eigenvector of A. In consequence, the computation by MIRAM could be applied on the distributed matrix B instead.

Secondly, this workload-centric approach may result in extensive communication volume. Parallel sparse MVP for scale-free networks does not scale well due to the high communication overhead caused by hubs (the most connected nodes). While sparse, the nonzero structure of their adjacency matrices are quite different from that of a PDE matrix. Rather, the existence of hubs necessitates an all-to-all communication either before or after the reduction operation in MVP, which makes the parallel communication requirements more similar to those of a dense matrix.

To clarify this observation, we use a simple example given in Fig. 4 with 3 processors. The columns 0, 3 and 6 of P correspond to three hubs in the network since they contain the most nonzero elements. From left to right, the columns 0–2 are distributed to processor 0; the columns 3–5 are distributed to processor 1, and the columns 6–8 are distributed to processor 2. The vector w is also partitioned into three segments (marked by three colors) and distributed among these processors.

Before the reduction of MVP, the processor that owns the column j needs only the corresponding element $w[j]$ in the vector w, which is also local to this processor. After the reduction operation to get its “partial sums” (in row-wise), each processor sends its partial sums to the processor that owns the vector segment for the corresponding rows. For example, after local reduction, the processor p_1 will get 9 partial sums, numbered from 0 to 8. The partial sums 0–2 will be sent to the processor 0 since it owns the red segment (the rows 0–2) of the vector w. Due to the existence of hubs, each processor will have 9 partial sums in the example. As a result, using 1D column-wise partitioning, every processor might be required to send messages to all other processors. This results in an all-to-all communication after the local reduction. Similarly, if we use 1D row-wise partitioning, an all-to-all communication before the reduction will be needed because of the existence of hubs.

In the second approach, the problem of load distribution and balancing in parallel MVP is formulated as a graph partitioning one. The idea is to find the subsets of nodes in the origin graph such that the number of edges between any two partitions are minimized. The nodes correspond to different rows/columns in the matrix A, and the edges between two partitions represent the communication requirements between two processors in parallel MVP. There are some popular graph partitioning packages, such as Chaco, Metis and Scotch [33–35]. They also offer MPI-based libraries for parallel graph partitioning. These packages are based on row-wise partitioning where each processor holds a block of rows of the matrix. From a matrix theoretical view, they simply try to minimize the total number of off-block-diagonal nonzeros without considering the relative spatial locations of such nonzeros. In other words, the graph models treat all off-block-diagonal nonzeros in an identical manner by assuming that each of them will incur a distinct communication of a single word [15]. However, before the reduction, the off-block-diagonal nonzeros in the same column engender only one message to get the corresponding vector component. After the reduction, the off-block-diagonal nonzeros in the same row reduce to one partial sum and incur only one message as well.

Recently, hypergraph-based partitioning [15] has drawn much attention from the PageRank community. We will continue our discussion firstly with a retrospect of some basic definitions of the hypergraph theory.

Definition 1 A hypergraph $\mathbb{H} = (V, N)$ is defined as a set of vertices V and a set of nets (hyperedges) N among these vertices. Every net $n_j \in N$ is a subset of vertices, i.e., $n_j \subseteq V$.

Definition 2 A k-way partition Π ($k > 1$) of the set V is defined as $\Pi = \{V_1, \ldots, V_k\}$, where V_i are subset of V s.t. $V_i \bigcap V_j = \emptyset$ for all $1 \leq i < j \leq k$.

Definition 3 The $k - 1$ metric is defined as

$$f(\mathbb{H}) = \sum_{i=1}^{n^*} (\pi_i - 1)\omega_i \tag{16}$$

where n^* is the number of nets, π_i is the number of subsets that the net n_i spans (i.e. has a vertex in) and ω_i is the number of constituent vertices of the net n_i.

The hypergraph partitioning problem consists in finding a k-way partition $\Pi = \{V_1, \ldots, V_k\}$ such that the $k - 1$ metric is optimized, and the number of vertices in each subset V_i is balanced. For 1D sparse matrix decomposition scheme, a matrix A is represented as a hypergraph $\mathbb{H}_{\mathbb{R}} = (V_R, N_C)$. Vertex and net sets V_R and N_C

correspond to the rows and columns of matrix A, respectively. The distribution of the rows of matrix A to p processors for parallel sparse MVP corresponds to a p-way partition of the above hypergraph. For 2D sparse matrix decomposition scheme, the objective is to distribute matrix nonzeros to processors instead. Here, each nonzero is represented by a vertex. Every column/row is modelled by a net. Its constituent vertices are the nonzeros of the column/row. In consequence, minimizing communication before and after the reduction of MVP could be accurately modelled by a hypergraph partitioning problem. Using these two schemes, Bradley et al. has observed a reduction of communication by $3\times$ compared to conventional graph partitioners [36]. In our implementation, we use "Zoltan" package [37] as our hypergraph partitioning tool. The cost is that it takes longer to run than graph algorithms.

4 Parallel Implementation

Today, the building block of the high-end computing system consists of multiple multi-core chips sharing memory in a single node. We use a hybrid programming model with message passing and shared memory (MPI and OpenMP). This model assumes that the system has a number of nodes with local memories and communicate with each other by means of memory transfer. In the meantime, each node is composed of a number of processors sharing a local memory. There is thus a hierarchical two-level parallelization in our implementation. The first one applies the 1D row-wise hypergraph partitioning for minimizing the communication in sparse MVP. Each MPI process works on one group of rows and exchanges data before the reduction operation. Parallelism in the first level is limited to the number of computing nodes available in the system. For the second-level parallelism, MVP kernel uses OpenMP parallel regions for local multiplication and reduction within a node. Our code is developed based on the Trilinos framework [14], where about fifty C++ packages are included.

From the developer's view, parallel MIRAM consists of three components. The first is a network loader to store the entire network in memory on a distributed memory parallel computer, the second is the Zoltan package that preprocesses the parallel matrix for load balancing and communication minimization and the third is the eigensolver described in Fig. 2.

4.1 Network Loader

The networks are initially stored as edge set in a coordinate format file. We parse the file and derive the corresponding transition matrix P and store it in matlab coordinate format. An example of 5 nodes is given in Fig. 5. The two columns on the left of the Table 2 are the endpoints of the edges, while the three columns on the right are the triplet (row_index, col_index, value).

After the conversion, we use *MatlabFileToCrsMatrix* function (in Trilinos' EpetraExt package) to load the matrix. Epetra provides construction routines as well as services function for data objects in distributed memory parallel machines. A class called *Epetra_Map* describes the mapping of every vector and matrix over MPI ranks. Vectors have a single 1D map while sparse matrices may have 1D or 2D maps. 1D

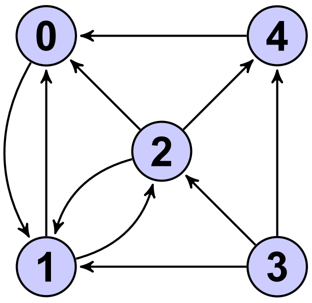

Fig. 5 Small social network of 5 individuals

Table 2 Example of network coordinate format and matlab coordinate format for Fig.5

Network coordinate format		Matlab coordinate format		
0	1	2	1	1
1	0	1	2	0.5
1	2	3	2	0.5
2	0	1	3	0.333
2	1	2	3	0.333
2	4	5	3	0.333
3	1	2	4	0.333
3	2	3	4	0.333
3	4	5	4	0.333
4	0	1	5	1

row-wise/column-wise distribution of sparse matrices is specified by row/column map. The 2D distribution can be specified by giving both row map and column map to the constructor of matrices. In our implementation, we store the transition matrix P as an *Epetra_CrsMatrix* using row map.

4.2 Hypergraph Partitioner

The main focus is to improve the scalability of sparse matrix vector multiplication over scale-free networks. To do so, we use the *Isorropia* package, interface to the Zoltan toolkit. It performs the partitioning mainly through three steps:

1. Create a *Isorropia::Partitioner* instance.
2. Create a *Isorropia::Redistributor* object.
3. Use the *Isorropia::Redistributor* to redistribute one or more objects to the new partitioning.

Weights can be defined by *Isorropia::CostDescriber* class for graphs and hypergraphs. *Isorropia* currently supports partitioning/redistributing of several Epetra objects, including *Epetra_CrsGraph* and *Epetra_CrsMatrix*, etc. *Isorropia* has a number of parameters that control the partitioning methods [38]. These parameters are placed in a *Teuchos::ParameterList* object, which is passed as an argument to the following *Isorropia*'s function:

Epetra_CrsMatrix Isorropia::Epetra::createBalancedCopy(const Epetra_CrsMatrix & input_matrix, const Teuchos::ParameterList & paramlist).*
We implement the hypergraph partitioner by calling
paramlist.set("PARTITIONING METHOD", "HYPERGRAPH").

4.3 Parallel MIRAM

MIRAM consists of four main tasks. First, the projection phase manipulates the n-sized data sets for sparse MVP. The second phase including implicitly shifted QR iterations acts on m-sized data sets. The third phase constructing the r additional steps of Arnoldi factorization manipulates on n-sized data sets as well. At last, the convergence test deals with n-sized data sets to calculate $\| f_m \|$. Because phase one and three constitute the most expensive part of the algorithm, we propose to distribute them among processors and to run phases two and four redundantly on all processors.

To conduct sparse MVP, Epetra uses two additional maps to specify the distribution of the input (*domain map*) and the output vectors (*range map*). Both the domain and range maps are one-to-one: that is, each global index in the map is uniquely owned by only one process. There are four steps for sparse MVP implemented in Epetra:

1. Import: Send w_i to the processes that own a nonzero a_{ij} for some i.
2. Local reduction: $y_i := y_i + a_{ij} * w_j$.
3. Export: Send partial y values to the owner processes.
4. Reduction: Add up partial y contributions received.

The communication steps 1 and 3 are point-to-point in Epetra and are implemented as the *Epetra_Import* and *Epetra_Export* classes respectively. In our MIRAM code, we use the following function:
int Epetra_CrsMatrix::Multiply(bool TransA, const Epetra_Vector & x, Epetra_Vector & y)
of *Epetra_CrsMatrix* class to perform MVP on the matrix P. The Importer and the Exporter classes will be automatically constructed based on its maps.

5 Experiments

In all of our experiments, the initial vector is $e = (1, 1, \ldots, 1)^T$.

Grid5000 platform We run our experiments on a nation wide cluster of clusters Grid5000. Grid5000 is a scientific instrument for the study of large scale parallel and distributed systems. It provides a highly reconfigurable, controllable and monitorable experimental platform to its users [39]. We conduct our experiments mainly on four clusters (some hardware details are given in Table 3). All clusters run a Debian Wheezy with a 3.2 Linux kernel.

Test data In social science, lack of realistic data, scientists tend to use synthesized or network-based social graphs to study various social problems including epidemiology. Many studies show that web graphs display similar underlying structure as social

Table 3 Hardware details of Clusters

Cluster	CPU	Network	Memory (GB)
Taurus 16 nodes	6 Cores/CPU	10 Gigabit ethernet	32
2 CPUs/node	Intel 2.3GHz		
Graphene 144 nodes	4 Cores/CPU	Infiniband-20G	16
1 CPUs/node	Intel 2.5GHz		
Griffon 120 nodes	4 Cores/CPU	Infiniband-20G	16
2 CPUs/node	Intel 2.5GHz		
Granduc 22 nodes	4 Cores/CPU	Gigabit ethernet	16
2 CPUs/node	Intel 2GHz		

Table 4 Statistics of datasets

Name	n	nnz	Storage
ba	7,010	13,985	117 KB
com-Youtube	1,134,890	2,988,374	38.7 MB
soc-LiveJournal1	4,847,571	68,993,773	1.1 GB
twitter	41,652,230	1,469,914,131	25 GB
yahoo	1,413,511,394	8,050,112,173	78 GB

graphs such as power law distribution of degrees and small-world phenomenon. In the following section, we present our results on seven networks. Their statistics are presented in Table 4. n is the number of nodes, nnz is the number of links. The number of links in the table is bigger than that in initial datasets because we add links for dangling nodes. *ba* is collected at the Oregon router views [40]. *com-Youtube* and *soc-LiveJournal1* are obtained from Stanford Large Network Dataset Collection [41]. *twitter* is collected from 467 million Twitter posts from 20 million users covering a 7-month period from June 1, 2009 to December 31, 2009 [42]. This dataset is more realistic to represent a social network. *yahoo* contains URLs and hyperlinks for over 1.4 billion public web pages indexed by the Yahoo! AltaVista search engine in 2002.

5.1 Thick Restart for the Choice of Parameter k

In the first place, we check the strategy proposed by Stathopoulos et al. in [32] for the choice of parameter k. The damping factor α is fixed to 0.85.

In the test on $twitter$ network, we set the m to be 4 and change the value of k to 1, 2 and 3. We run our experiments on cluster "Taurus" using 16 nodes with 2 MPI processes per node (without OpenMP multithreading). The results are presented in Fig. 6. While ($k = 1$) uses the fewest restarting cycles, ($k = 2$) allows the fastest convergence in terms of execution time. In consequence, keeping a buffer of 1 extra vector accelerates the convergence rate for the dominant eigenpair.

Similar experiments are conducted for *yahoo* network using 144 nodes of "Graphene" cluster with one MPI process per core. The results are presented in Fig. 7.

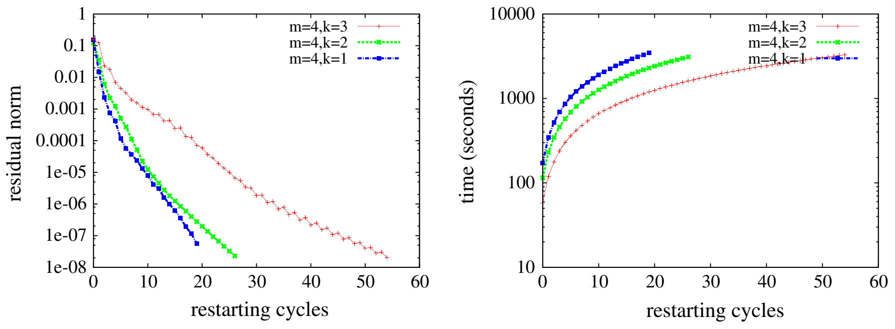

Fig. 6 Convergence experiments for different number of shifts on *twitter* network, where $\alpha = 0.85$ and $tol = 1E - 7$. **a** Number of restarting cycles. **b** Execution time

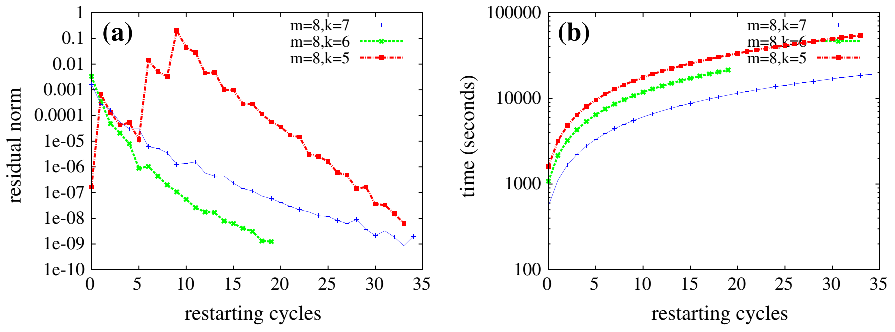

Fig. 7 Convergence experiments for different number of shifts on *yahoo* network, where $\alpha = 0.85$ and $tol = 1E - 8$. **a** Number of restarting cycles. **b** Execution time

Parameter configurations ($k = 7$) and ($k = 6$) have almost the same convergence rate, while ($k = 5$) converges much slower.

To sum up, retaining more eigenvectors in IRAM ($k > 1$) is generally beneficial to the convergence of dominant eigenpair.

5.2 Strong Scalability Tests

In this experiment, we run each MPI process on one 4-core CPU with 4 OpenMP threads. So each core has only one OpenMP thread running on it.

Firstly, we test the scalability of sparse MVP on *com-Youtube* network. Figure 8 shows the computation time as a function of number of processors. The first curve in the top-down order corresponds to an equal-partitioned scheme with $\lceil n/p \rceil$ rows per processor. The curve below shows the strong scalability result of hypergraph partitioning on matrix A. Equal-partitioned scheme leads to slower computation due to more significant communications overhead. The result shows that the hypergraph partitioning strategy is useful to handle matrices of this particular structure. And our implementation has obtained up to $11\times$ acceleration with many cores.

In the second place, we conduct scalability tests for our parallel MIRAM implementation. The experiments are conducted for *com-Youtube* and *soc-LiveJournal1*

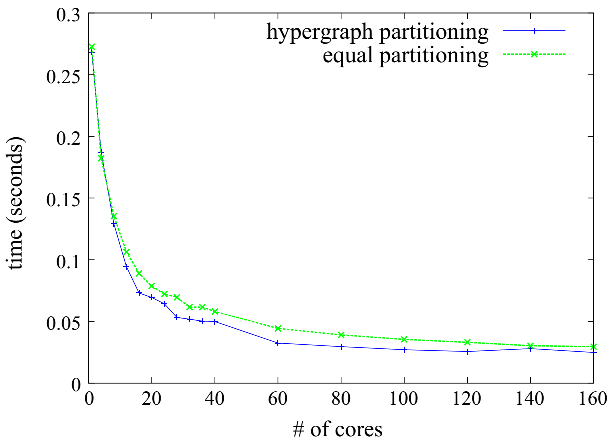

Fig. 8 Scalability experiment of sparse MVP for *com-Youtube* network, where $\alpha = 0.85$, on "Griffon cluster"

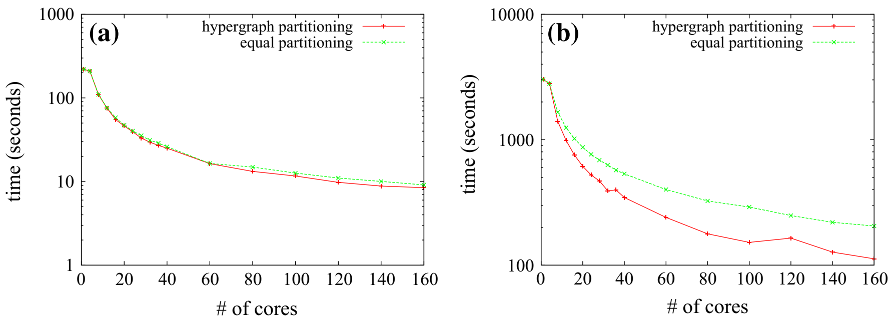

Fig. 9 Scalability experiment of MIRAM, where $\alpha = 0.85$, $k = 2$ and $tol = 1E - 12$. **a** MIRAM(4,8) for *com-Youtube*, on "Griffon cluster". **b** MIRAM(4,8,16) for *soc-LiveJournal1*, on "Granduc cluster"

matrices. Still, we see in Fig. 9 that the hypergraph-based implementation outperforms the equal-partitioned version. With 160 processors, we have obtained an acceleration up to 27×.

Parallel efficiency has tendency to decrease as the number of nodes increase. This is because the communication overhead is important in grid systems. As shown in Fig. 7b, with 144 grid nodes, we could expect an execution time around 8 hours for a very large network such as *yahoo*, comparable to a country/continental wide realistic scenario.

5.3 MIRAM Versus IRAM

Concerning the use of the parallelism of the system, we use 30 nodes from "Griffon" cluster. We run one MPI process on each node with 8 OpenMP threads (one OpenMP thread per core). Totally, 240 cores are used for each test.

In Fig. 10a, MIRAM(2,3,4,5,6) and MIRAM(2,6) use fewer restarting cycles than IRAM(6). The result shows that the convergence of MIRAM can be better than that

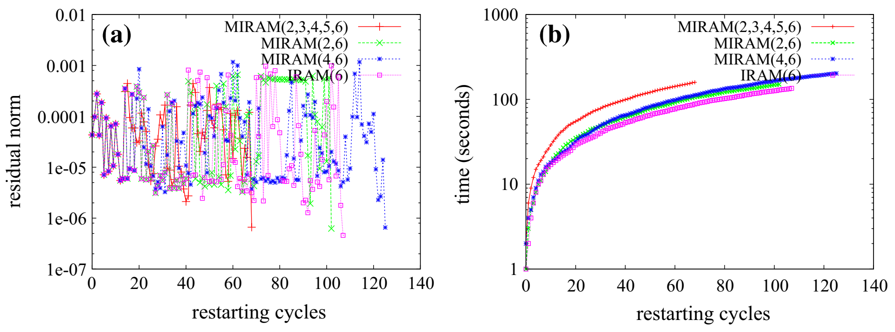

Fig. 10 MIRAM versus IRAM for *com-Youtube*, where $\alpha = 0.99$, $k = 1$ and $tol = 1E-6$. **a** Number of restarting cycles. **b** Execution time

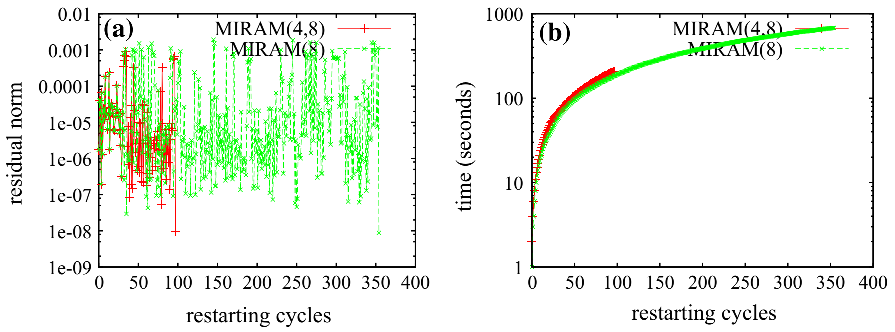

Fig. 11 MIRAM versus IRAM for *com-Youtube*, where $\alpha = 0.85$, $k = 1$ and $tol = 1E-8$. **a** Number of restarting cycles. **b** Execution time

of IRAM. Nevertheless, MIRAM(4,6) using the most restarting cycles indicates that an unfortunate parameter setting for MIRAM could result in slower convergence. Moreover, it is not the number of subspace spaces who counts. In fact, MIRAM(2,6) uses the fewest restarting cycles in this test.

Fig. 10b shows that IRAM(6) has the fastest convergence in terms of execution time. As analysed in section 3.2, one restarting cycle of MIRAM (when m_ℓ is chosen) is more expensive than that of IRAM. Indeed, from Fig. 12a, we see that m_ℓ is used most of the time for all three MIRAMs. That is the reason why MIRAM spends less restarting cycles but uses more execution time.

The good news is that MIRAM can significantly reduce the number of restarting cycles, which could compensate for its additional computation cost. This is demonstrated by the result shown in Figs. 11 and 12(b).

Due to the limitation on subspace size for large scale applications, IRAM may not be efficient for computing the dominant eigenvector for such large sparse non-Hermitian matrices. Making use of several nested Krylov subspaces could help to improve the convergence as shown in our experiments. Furthermore, the number of MVP in MIRAM is decided by the largest subspace size because other subspaces are nested within this one. As a result, MIRAM($m_1, \cdots, m_\ell$) accelerates the convergence of IRAM(m_ℓ) with the same number of MVP in each iteration.

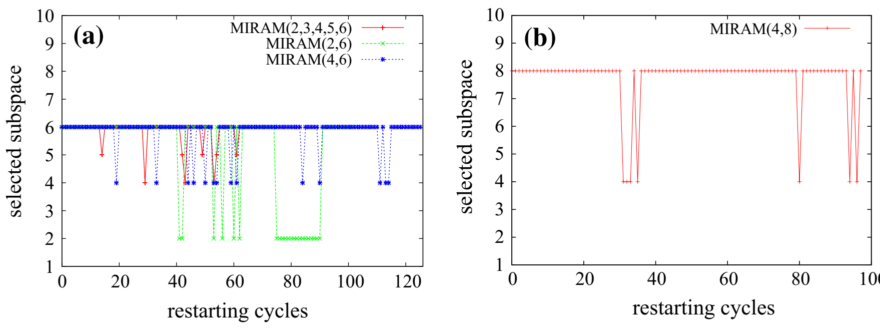

Fig. 12 Evolution of m_{best} in MIRAM along restarting cycles for *com-Youtube*, where $k = 1$. **a** $\alpha = 0.99$ and $tol = 1E - 6$. **b** $\alpha = 0.85$ and $tol = 1E - 8$

Table 5 MIRAM versus Power method in terms of execution time (seconds) and number of MVP, where $tol = 1.55E - 14$

	Power method		MIRAM(4,8),k=2	
α	MVP	Ex.Time	MVP	Ex.Time
0.85	239	27.29	56	22
0.90	401	48.53	62	21
0.95	725	83.56	62	27
0.99	3525	363.94	62	23

5.4 Experiments on Damping Factor α

We use the same computing system configuration as in the previous experiment. In this test, we study the influence of damping factor on convergence rate of Power method and MIRAM. For *com-Youtube* network, this dependency is quantified in Table 5. We found that bigger damping factor α needs more MVPs to reach the accuracy for both methods. However, MIRAM has a much better performance than Power method for bigger α. As explained in [11], bigger α engenders a closer-to-1 second largest eigenvalue. This fact also favors Arnoldi-type methods, as opposed to Power method. Noticed that a Power iteration is extremely cheaper computationally than an IRAM iteration. As shown in the results for ($\alpha = 0.85$), 239 Power iterations only use 27.29s while MIRAM costs 22s in 56 MVP iterations.

5.5 Vaccination Strategies Based on PageRank

In this experiment, we use a small network *ba* to simulate vaccination effect on epidemic spread. We consider people receiving vaccination as permanently immune to viruses. For larger network, parallelization will be needed due to the memory and computation requirements, but the implementation of such parallel simulator is not the objective of the test.

We assume a universal infection rate ν, a jumping rate $1 - \alpha$ (damping factor) and a curing rate δ for every individual. Before each simulation, we randomly choose a set of infected individuals. The propagation of virus proceeds by time step. During

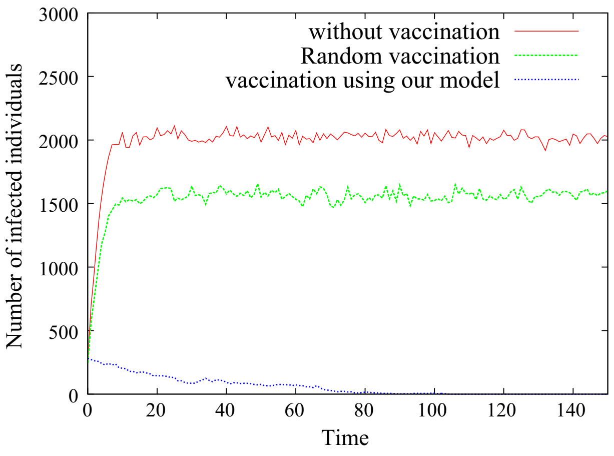

Fig. 13 Time series of infection in an 7010-node power-law social network *ba*, with $\alpha = 0.85$, $\nu = 0.2$ and $\delta = 0.24$

each time step, an infected individual infects each of its neighbours with probability ν. And this infected individual also passes the disease to another randomly chosen non-neighbour by probability $1 - \alpha$. Additionally, every infected individual is cured with probability δ. The result is the average over 10 runs and it is presented in Fig. 13.

Here, we compare three cases. First of all, without distribution of vaccination, we give the worst case for time evolution of infection. Secondly, with random distribution of vaccination, we begin the simulation by distributing vaccination to a randomly chosen group of individuals. Then, we simulate time evolution of infection. Thirdly, with distribution of vaccination using the PageRank-like vector, we calculate the infection vector for the underlying social network, and then distribute vaccination to individuals with big vector ranking.

The figure verifies the absence of epidemic threshold in scale-free networks [43]. Without interventions, the epidemic will always enter an endemic state. The second curve, in top-down order from the figure, shows that random distribution of vaccination could not prevent the virus from entering an endemic state. However, distributing vaccination to individuals with big ranking in the PageRank-like vector makes the epidemic die out quickly. This simple experiment confirms the important implication of infection vector for the control of epidemic spread.

6 Conclusion

Modeling of epidemic spread benefits a lot from network research to understand infection evolution in a population. PageRank-like model could shed light on understanding the impact of social network structure on propagation of virus and could help identifying individuals most likely to spread the disease. Besides, parallelism makes the model computationally adavantageous over traditional approaches.

It is known that PageRank can be computed using numerical methods based on sparse MVP and we propose to use a parallel “multiple IRAM” algorithm (MIRAM).

From the Experiment 5.4, we see that MIRAM is promising especially for big damping factors. The parallel MIRAM implementation takes into account the scale-free structure of underlying networks and is scalable to handle memory and computation issues arising from very large networks such as *twitter* and *yahoo* network. From our tests, we have obtained a speedup of 27$\times$ compared to sequential solver. Additionally, it is found in Experiment 5.1 that thick restart could help accelerate the convergence of the method even under constraints caused by storage.

MIRAM (with nested or non nested subspaces) has a great potential for large coarse grain parallelism among its Arnoldi factorizations. Different from the description in Sect. 4.3, the restarting vector can be made different among processors. In this case, the whole orthogonal basis of the chosen subspace should also be sent to processors. Consequently, the computation in different subspaces of MIRAM will be asynchronous. This coarse grain parallelism is fault tolerant since any loss of an IRAM process during MIRAM execution does not interfere with its termination. All these properties show that MIRAM is well suitable for large scale distributed computational environments. The analysis of intra and inter Arnoldi factorizations parallelism in the asynchronous version of MIRAM can be the subject of a future work. Moreover, we intend to expand our epidemic model by including various indicators of epidemic spread, such as characteristics of individuals as well as that of viruses, spreading timestamps, etc.

Acknowledgments We would like to thank Fabrcio Benevenuto from Federal University of Ouro Preto for the *twitter* network, Kim Capps from Yahoo! Labs for his help to get access to Alta Vista web network.

References

1. Liu, Z., Emad, N., Amor, S.B., Lamure, M.: Towards modeling of epidemic spread: eigenvalue computation. Preprint for publication. URL:http://hal.archives-ouvertes.fr/hal-01069010
2. Page, L., Brin, S., Motwani, R., Winograd, T.: The Pagerank citation ranking: bringing order to the Web. Technical Report 1999–66, Stanford InfoLab (1999)
3. Bryan, K., Leise, T.: The $25,000,000,000 eigenvector: The linear Algebra behind Google. SIAM Rev. **48**(3), 569–581 (2006). doi:10.1137/050623280. ISSN:0036-1445
4. Langville, A.N., Meyer, C.D.: Google's PageRank and Beyond: the Science of Search Engine Rankings. Princeton University Press, Princeton, NJ, USA. ISBN:0691122024 (2006)
5. Berkhin, P.: A survey on pagerank computing. Internet Math. **2**, 73–120 (2005)
6. Golub, G.H., Greif, C.: An Arnoldi-type algorithm for computing PageRank. BIT Numer. Math. **46**(4), 759–771 (2006)
7. Wu, G., Wei, Y.: An Arnoldi-extrapolation algorithm for computing PageRank. J. Comput. App. Math. **234**(11), 3196–3212 (2010) (Numerical linear algebra, internet and large scale applications). ISSN:0377-0427. doi:10.1016/j.cam.2010.02.009. URL:http://www.sciencedirect.com/science/article/pii/S0377042710000804
8. Gleich, D., Zhukov, L., Berkhin, P.: Fast parallel PageRank: a linear system approach. Technical Report L-2004-038, Yahoo! Research Labs (2004)
9. Wu, G., Wei, Y.: Arnoldi versus GMRES for computing PageRank: a theoretical contribution to Google's PageRank problem. ACM Trans. Inf. Syst. **28**(3), 11:1–11:28 (2010). ISSN:1046–8188. doi:10.1145/1777432.1777434
10. Wu, G., Wang, Y.-C., Jin, X.-Q.: A preconditioned and shifted GMRES algorithm for the PageRank problem with multiple damping factors. SIAM J. Sci. Comput. **34**(5) (2012)
11. Haveliwala, T.H., Kamvar, S.D., Kamvar, A.D.: The second eigenvalue of the Google matrix. Technical Report 2003-20, Stanford InfoLab (2003)

12. Liu, Z., Emad, N., Amor, S.B., Lamure, M.: A parallel IRAM algorithm to compute PageRank for modeling epidemic spread. Symp. Comput. Architect. High Perform. Comput. **0**, 120–127 (2013). doi:10.1109/SBAC-PAD.2013.2
13. Fazeli, S.A.S., Emad, N., Liu, Z.: A key to choose subspace size in implicitly restarted Arnoldi method. J. Numer. Algorithm (2014). http://hal.archives-ouvertes.fr/hal-01070577
14. Heroux, M., Bartlett, R., Hoekstra, V.H.R., Hu, J., Kolda, T., Lehoucq, R., Long, K., Pawlowski, R., Phipps, E., Salinger, A., Thornquist, H., Tuminaro, R., Willenbring, J., Williams, A.: An overview of Trilinos. Technical Report SAND2003-2927, Sandia National Laboratories (2003)
15. Catalyurek, U., Aykanat, C.: Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication. IEEE Trans. Parallel Distrib. Syst., **10**(7), 673–693 (1999). doi:10.1109/71.780863. ISSN 1045-9219
16. Marathe, M., Vullikanti, A.K.S.: Computational epidemiology. Commun. ACM **56**(7), 88–96 (2013). ISSN:0001-0782. doi:10.1145/2483852.2483871
17. Bisset, K., Chen, J., Feng, X., Anil Kumar, V.S., Marathe, M.: EpiFast: A fast algorithm for large scale realistic epidemic simulations on distributed memory systems. In: Proceedings of 23rd ACM International Conference on Supercomputing (ICS'09), pp. 430–439 (2009)
18. Bisset, K.: Urgent computing for interaction based socio-technical simulations. Invited presentation to Argonne National Laboratory, April
19. Chao, D.L., Halloran, M.E., Obenchain, V.J., Longini, I.M., Flu Jr, T.E.: A publicly available stochastic influenza epidemic simulation model. PLoS Comput. Biol. **6**(1), e1000656, 01 (2010). doi:10.1371/journal.pcbi.1000656
20. Wang, Y., Chakrabarti, D., Wang, C., Faloutsos, C.: Epidemic spreading in real networks: an eigenvalue viewpoint. In: SRDS, pp. 25–34 (2003)
21. Miller, J.C., Hyman, J.M.: Effective vaccination strategies for realistic social networks. Phys. A **386**(2), 780–785 (2007)
22. Fan, R.K.: Chung, Paul Horn, and Alexander Tsiatas. Distributing Antidote Using PageRank Vectors. Internet Math. **6**(2), 237–254 (2009)
23. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science **286**(5439), 509–512 (1999)
24. Lee, C.P., Golub, G.H., Zenios, S.A.: A fast two-stage algorithm for computing PageRank and its extensions. Technical report, Stanford University. URL:http://www-sccm.stanford.edu/pub/sccm/sccm03-15_2.pdf (2004)
25. Ipsen, I.C.F., Selee, T.M.: PageRank computation, with special attention to dangling nodes. SIAM J. Matrix Anal. Appl., **29**(4), 1281–1296 (2007). doi:10.1137/060664331. ISSN:0895-4798
26. Eiron, N., McCurley, K.S., Tomlin, J.A.: Ranking the web frontier. In: Proceedings of the 13th International Conference on World Wide Web, WWW '04, pp. 309–318, New York, NY, USA. ACM (2004). ISBN:1-58113-844-X. doi:10.1145/988672.988714
27. Sorensen, D.C.: Implicit application of polynomial filters in a k-step Arnoldi method. SIAM J. Matrix Anal. Appl. **13**(1), 357–385 (1992). ISSN:0895–4798. doi:10.1137/0613025
28. Sorensen, D.C.: Implicitly restarted Arnoldi/Lanczos methods for large scale eigenvalue calculations. Technical report (1996)
29. Sorensen, D.C.: Numerical methods for large eigenvalue problems. Acta Numer. **11**, 519–584 (2002). doi:10.1017/S0962492902000089
30. Watkins, D.S.: The QR algorithm revisited. SIAM Rev. **50**(1), 133–145 (2008). ISSN:0036-1445. doi:10.1137/060659454
31. Bennani, M., Braconnier, T.: Stopping Criteria for Eigensolvers. Technical Report TR/PA/94/22, CERFACS, Toulouse, France (1994)
32. Stathopoulos, A., Saad, Y.: Dynamic thick restarting of the Davidson, and the implicitly restarted Arnoldi methods. SIAM J. Sci. Comput. **19**, 227–245 (1996)
33. Hendrickson, B., Leland, R.: The chaco user's guide: Version 2.0. Technical Report SAND94-2692, Sandia National Lab (1994)
34. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1998). ISSN:1064–8275. doi:10.1137/S1064827595287997
35. Pellegrini, F.: Scotch and libScotch 5.1 user's guide. URL http://hal.archives-ouvertes.fr/hal-00410327. 127 pages User's manual (2008)
36. Bradley, J.T., de Jager, D., Knottenbelt, W.J., Trifunovic, A.: Hypergraph partitioning for faster parallel PageRank computation. In: EPEW'05, Proceedings of the 2nd European Performance Evaluation

Workshop, volume 3670 of Lecture Notes in Computer Science, pp. 155–171, September 2005 (2005). URL http://pubs.doc.ic.ac.uk/hypergraph-fast-pagerank/
37. Boman, E.G., Çatalyürek, Ü.V., Chevalier, C., Devine, K.D.: The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: partitioning, ordering and coloring. Sci. Progr. **20**(2), 129–150 (2012)
38. Isorropia: Partitioning, Coloring, and Ordering. http://trilinos.org/docs/r11.8/packages/isorropia/doc/html/index.html. Trilinos Release 11.8
39. Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., Touche, I.: Grid'5000: A large scale and highly reconfigurable experimental grid testbed. Int. J. High Perform. Comput. Appl. **20**(4), 481–494 (2006). ISSN:1094-3420. doi:10.1177/1094342006070078
40. BA Data Sets: http://topology.eecs.umich.edu/data.html
41. SNAP Data Sets.: http://snap.stanford.edu/data/index.html
42. Kwak, Haewoon., Lee, Changhyun., Park, Hosung., Moon, Sue.: What is Twitter, a social network or a news media? In: WWW '10: Proceedings of the 19th international conference on World wide web, pp. 591–600, New York, NY, USA. ACM (2010). ISBN:978-1-60558-799-8. doi:10.1145/1772690.1772751
43. Romualdo, P.-S., Alessandro, V.: Epidemic spreading in scale-free networks. Phys. Rev. Lett. **86**, 3200–3203 (2001). doi:10.1103/PhysRevLett.86.3200