SeMA (2018) 75:1–16
https://doi.org/10.1007/s40324-017-0108-2

Computing the smallest eigenpairs of the graph Laplacian

Luca Bergamaschi[1] · **Enrico Bozzo**[2]

Received: 28 September 2016 / Accepted: 19 January 2017 / Published online: 6 February 2017

Abstract The graph Laplacian, a typical representation of a network, is an important matrix that can tell us much about the network structure. In particular its eigenpairs (eigenvalues and eigenvectors) incubate precious topological information about the network at hand, including connectivity, partitioning, node distance and centrality. Real networks might be very large in number of nodes; luckily, most real networks are sparse, meaning that the number of edges (binary connections among nodes) are few with respect to the maximum number of possible edges. In this paper we experimentally compare three important algorithms for computation of a few among the smallest eigenpairs of large and sparse matrices: the Implicitly Restarted Lanczos Method, which is the current implementation in the most popular scientific computing environments (MATLAB/R), the Jacobi–Davidson method, and the Deflation Accelerated Conjugate Gradient method. We implemented the algorithms in a uniform programming setting and tested them over diverse real-world networks including biological, technological, information, and social networks.

Keywords Graph Laplacian · Eigenpair computation · Algorithms · Networks

Mathematics Subject Classification 65F15 · 65Y20 · 05C82

The work of L. Bergamaschi has been partially supported by the Spanish Grant MTM2010-18674 and by the Italian Grant CPDA155834/15.

✉ Luca Bergamaschi
luca.bergamaschi@unipd.it

Enrico Bozzo
enrico.bozzo@uniud.it

[1] Department of Civil Environmental and Architectural Engineering, University of Padua, via Marzolo 9, Padua, Italy

[2] Department of Mathematics Computer Science and Physics, University of Udine, via delle Scienze 206, Udine, Italy

1 Introduction

The bi-directional link between network science and matrix algebra is intriguing and promising [10,22]. Of course we can apply results and methods of matrix algebra to investigate the properties of networks. On the other hand real networks, with their universal architectures, represent a class of algebraic structures (matrices) for which results and methods of matrix algebra can be improved and specialized. Clearly, both network science and matrix algebra would benefit from this synergistic approach. Network science would gain additional insight in the structure of real networks, while matrix algebra would obtain more challenging applications.

Networks, in their basic form of graphs of nodes and edges, can be represented as matrices. The most common representation of a graph consists of the graph adjacency matrix, where the entries of the matrix that are not null represent the edges of the graph. Often, it is convenient to represent a graph with its Laplacian matrix, which places on the diagonal the degrees of the graph nodes (the number of connections of the nodes) and elsewhere information about the distribution of edges among nodes in the graph. The Laplacian matrix, and in particular its smallest eigenpairs (eigenpairs relative to the smallest eigenvalues), turn up in many different places in network science. Examples include random walks on networks, resistor networks, resistance distance on networks, current-flow closeness and betweenness centrality measures, graph partitioning, and network connectivity [9,16,23].

Real networks might be very large; however, they are typically also very sparse. Moreover, generally, not the entire matrix spectrum is necessary, but only a few eigenpairs, either the lowest of the largest, are enough. In Sect. 2 we will discuss in detail some applications that require the approximate computation of a number of the smallest eigenvalues of the Laplacian matrix. One of these is the computation of the betweenness centrality index [23] which quantifies the information that passes through a node in order to transit between others. This index can be approximated by computing some of the smallest eigenpairs of the Laplacian.

A number of iterative procedures, based on a generalization of the well-known power method, have been recently developed to compute a few eigenpairs of a large and sparse matrix. In this paper, we experimentally analyze three important iterative methods: (i) the Implicitly Restarted Lanczos Method [20], (ii) the Jacobi–Davidson method [25], and (iii) the Deflation Accelerated Conjugate Gradient method [5]. We implement these methods in a uniform programming environment and experimentally compare them on four Laplacian matrices of networks arising from realistic applications. The real networks include a biological network (a protein-protein interaction network of yeast), a technological network (a snapshot of the Internet), an information network (a fragment of the Web), and a social network (the whole collaboration network among Computer Science scholars).

The layout of the rest of the paper is the following. In Sect. 2 we describe some applications of the lowest eigenpairs of the Laplacian matrix of a graph. The compared algorithms are reviewed in Sect. 3. Section 4 is devoted to the discussion of the outcomes of the comparison among the algorithms when they run on real network data. We draw our conclusions in Sect. 5.

2 Why computing some eigenpairs of the Laplacian matrix of a graph

Let $\mathcal{G} = (N, E, w)$ be a simple (no multiple edges, no self-loops) undirected weighted graph with N the set of nodes, $|N| = n$, E the set of edges, $|E| = m$, and w a vector such that

$w_k > 0$ is the positive weight of edge k, for $k = 1, \ldots, m$. The weighted Laplacian of $\mathcal{G}$ is the $n \times n$ symmetric matrix

$$G = D - A,$$

where $A = (a_{i,j}), i, j = 1, \ldots, n$, is the weighted adjacency matrix of $\mathcal{G}$ and D is the diagonal matrix of the generalized degrees (the sum of the weights of the incident arcs) of the nodes. Hence, if $G = (g_{i,j}), i, j = 1, \ldots, n$, then $g_{i,j} = -a_{i,j}$ if $i \neq j$ while $g_{i,i} = \sum_{j=1}^{n} a_{i,j}$. In the following we focus on the spectral properties of the weighted Laplacian matrix and on their applicative importance.

If e denotes a vector of ones by definition $De = Ae$ so that $Ge = 0$. Thus e is an eigenvector of G associated to the eigenvalue $\lambda_1 = 0$. In addition if $x \in \mathbb{R}^n$ then

$$\begin{aligned} 0 \leq \frac{1}{2} \sum_{i,j=1}^{n} a_{i,j}(x_i - x_j)^2 &= \frac{1}{2} \left(2 \sum_{i=1}^{n} x_i^2 \sum_{j=1}^{n} a_{i,j} - 2 \sum_{\substack{i,j=1 \\ j \neq i}}^{n} x_i x_j a_{i,j} \right) \\ &= \frac{1}{2} \left(2 \sum_{i=1}^{n} x_i^2 g_{i,i} + 2 \sum_{\substack{i,j=1 \\ j \neq i}}^{n} x_i x_j g_{i,j} \right) \\ &= \sum_{i,j=1}^{n} x_i x_j g_{i,j} = x^T G x. \end{aligned} \tag{1}$$

From (1) it follows that G, besides symmetric, is positive semidefinite, and hence it has real and nonnegative eigenvalues that is useful to order $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$.

A basic result states that the multiplicity of $\lambda_1 = 0$ as an eigenvalue of G coincides with the number of the connected components of $\mathcal{G}$. Hence $\lambda_2 > 0$ if and only if $\mathcal{G}$ is connected. Fiedler [12] was one of the pioneers of the study of the relations between the connectivity properties of $\mathcal{G}$ and the spectral properties of G, and for this reason λ_2 is called algebraic connectivity or Fiedler value.

Since G is symmetric it admits the spectral decomposition

$$G = V \Lambda V^T$$

where Λ is the diagonal matrix such that $\Lambda(i, i) = \lambda_i, i = 1, \ldots, n$, and V is orthogonal, i.e. $VV^T = I = V^T V$, and its columns are the eigenvectors of G. In particular, the first column of V is equal to $e/\sqrt{n}$ since it is the normalized eigenvector of G associated with the eigenvalue $\lambda_1 = 0$. Hence, by using the well known MATLAB colon notation, $V(:, 1) = e/\sqrt{n}$. In addition $V(:, 2)$ will be called Fiedler vector, being the eigenvector associated with the Fiedler value.

Certain applicative problems require the minimization of $x^T Gx$ under the condition that the entries of the vector x belong to some discrete set.

An important example is discussed in [23] and concerns graph partitioning. Let us partition N in two subsets N_1 and N_2. If we set $x_i = \frac{1}{2}(-1)^j$ if $x_i \in N_j$ then $\frac{1}{2} \sum_{i,j=1}^{n} a_{i,j}(x_i - x_j)^2$ is the sum of the weighs of the arcs between N_1 and N_2, and is called the cut size. The graph partitioning problem requires to find N_1 and N_2 of prescribed dimensions n_1 and $n_2 = n - n_1$, in such a way the cut size is minimized. Actually, all the known methods for finding the minimum are very demanding, since they reduce to an enumeration of all the $\binom{n}{n_1} = \dfrac{n!}{n_1!(n - n_1)!}$ possible solutions. However, it is possible to approximate the minimum

by relaxing the constraints on the entries of x, allowing them to assume real values in such a way that $x^T x = n/4$ and $e^T x = (n_1 - n_2)/2$. From (1) it follows that

$$\min_{\substack{s.t. \\ x^T x = n/4 \\ x^T e = (n_1 - n_2)/2}} \frac{1}{2} \sum_{i,j=1}^{n} a_{i,j}(x_i - x_j)^2 = \min_{\substack{s.t. \\ x^T x = n/4 \\ x^T e = (n_1 - n_2)/2}} x^T G x.$$

If we set $y = V^T x$ we obtain $e^T x = e^T V y = y_1 \sqrt{n}$ so that

$$\min_{\substack{s.t. \\ x^T x = n/4 \\ x^T e = (n_1 - n_2)/2}} x^T G x = \min_{\substack{s.t. \\ y^T y = n/4 \\ y_1 = (n_1 - n_2)/(2\sqrt{n})}} y^T \Lambda y.$$

Presented in this spectral form the problem becomes simple. It is easy to find that the minimum is $\frac{n_1 n_2}{n} \lambda_2$ and is obtained when $y_1 = (n_1 - n_2)/(2\sqrt{n})$, $y_2 = \sqrt{(n_1 n_2)/n}$ and $y_i = 0$ for $i > 2$. Hence, the minimum of the original problem is obtained for

$$x = \frac{n_1 - n_2}{2n} e + \sqrt{\frac{n_1 n_2}{n}} V(:, 2),$$

showing the central role played by the Fiedler vector in the problem.

A second example of the same nature is discussed in [17]. In the case where all the weights are equal to one, the minimization of $x^T G x$, with the constraint that the entries of x belong to the $n!$ possible permutations of the integers from 1 to n, allows to find an ordering of the nodes of $\mathcal{G}$ that concentrates the entries of A near the main diagonal. For this reason it is known as profile reducing ordering. By relaxing the problem we find as an approximate solution the ordering induced by the entries of the Fiedler vector. This kind of applications do not require an accurate computation of the entries of the vector.

A different but equally important application concerns the problem of the computation of betweenness centrality [23]. This centrality index quantifies the quantity of information that passes through a node in order to transit between others. Actually, for the computation of betweenness centralities, a linear system in G for every couple of nodes of the network has to be solved. This is actually equivalent to the computation of G^+, the Moore–Penrose generalized inverse of G [3,16]. It turns out that

$$G^+ = \sum_{i=2}^{n} \frac{1}{\lambda_i} V(:, i) V(:, i)^T.$$

The use of approximations of G^+ obtained by partial sums

$$T^{(k)} = \sum_{i=2}^{k} \frac{1}{\lambda_i} V(:, i) V(:, i)^T,$$

has been proposed in [9]. Clearly this implies the computation a certain number of the smallest eigenpairs of the Laplacian. Moreover, if the eigenvalues λ_i, for $i = k+1, \ldots, n$ are close to each other it is possible to approximate them by means of a suitable constant σ (for example $\sigma = (\lambda_k + \lambda_n)/2$, or simply $\sigma = \lambda_k$). In [9] it has been shown that the use of

$$\begin{aligned} S^{(k)} &= T^{(k)} + \sum_{j=k+1}^{n} \frac{1}{\sigma} V(:, j) V(:, j)^T \\ &= \frac{1}{\sigma} I - \frac{1}{\sigma} V(:, 1) V(:, 1)^T + \sum_{j=2}^{k} \left(\frac{1}{\lambda_j} - \frac{1}{\sigma} \right) V(:, j) V(:, j)^T, \end{aligned}$$

in the place of $T^{(k)}$ leads to improved approximations of the centralities. It is important to note that in order to use $S^{(k)}$ no additional eigenpairs with respect to $T^{(k)}$ are requested.

3 Three algorithms for eigenvalue computations of large matrices

Starting from the subspace iteration, which is a generalization of the well-known power method, a number of iterative procedures have been developed to compute a few eigenpairs of a large and sparse matrix A. In the following, we describe three important methods:

- The Implicitly Restarted Lanczos Method (IRLM).
- The Jacobi–Davidson method (JD).
- The Deflation Accelerated Conjugate Gradient method (DACG).

All the methods are iterative, in the sense that they compute one or more eigenpairs by constructing a sequence of vectors which approximate the exact solution. They are all based on the following tasks which must be efficiently carried on:

1. Computation of the product of the matrix A by a vector. This matrix vector product (MVP) has a cost proportional to the number of nonzero entries of A.
2. Computation of a matrix M, known as preconditioner, that approximates A^{-1} in such a way that the eigenvalues of MA are well clustered around 1 and in addition the computations of Mv and Av, being v a generic vector, require comparable CPU time.

It is important to stress that IRLM and JD are characterized by an inner-outer iteration, where at every outer iteration a linear system has to be solved. However, while IRLM requires to solve these linear systems to a high accuracy, which is strictly related to the accuracy requested for the eigenpairs, for JD inexact solution of inner linear system is sufficient to achieve overall convergence. On the other hand, DACG does not require any linear system solution.

3.1 Description of implicitly restarted Lanczos method

The best known method, the Implicitly Restarted Arnoldi Method (IRAM), is implemented within the ARPACK package [20] and is also available in the most popular scientific computing packages (MATLAB/R). For symmetric positive definite matrices, IRAM simplifies to IRLM, Implicitly Restarted Lanczos Method which reduces the computational cost, by taking advantage of the symmetry of the problem.

The idea of the Lanczos method is to project the coefficient matrix A onto a subspace generated by an arbitrary initial vector $\boldsymbol{v}_1$ and the matrix A itself, known as Krylov subspace. In particular, a Krylov subspace of dimension m is generated by the following set of independent vectors:

$$\boldsymbol{v}_1, A\boldsymbol{v}_1, \ldots A^{m-1}\boldsymbol{v}_1.$$

Actually it is convenient to work with an orthogonal counterpart of this basis and to organize its vectors as columns of a matrix V_m. Then, a symmetric and tridiagonal matrix

$$T_m = \begin{pmatrix} \alpha_1 & \beta_2 & & & \\ \beta_2 & \alpha_2 & \beta_3 & & \\ & \ddots & \ddots & \ddots & \\ & & \beta_{m-1} & \alpha_{m-1} & \beta_m \\ & & & \beta_m & \alpha_m \end{pmatrix}$$

Algorithm 1 Implicitly Restarted Lanczos Method

Computation of T_m for A.
$\boldsymbol{v}_1 :=$ unitary norm initial vector.
$\boldsymbol{v}_0 := 0$
$\beta_1 := 0$
for $j = 1, 2, \ldots, m$
  $w_j := A\boldsymbol{v}_j$
  $\alpha_j := w_j^T \boldsymbol{v}_j$
  $w_j := w_j - \alpha_j \boldsymbol{v}_j - \beta_j \boldsymbol{v}_{j-1}$
  $\beta_{j+1} := \|w_j\|$
  $\boldsymbol{v}_{j+1} := w_j/\beta_{j+1}$
end for

Computation of T_m for A^{-1}.
$\boldsymbol{v}_1 :=$ unitary norm initial vector.
$\boldsymbol{v}_0 := 0$
$\beta_1 := 0$
for $j = 1, 2, \ldots, m$
  Solve $Aw_j = \boldsymbol{v}_j$ using the PCG method
  $\alpha_j := w_j^T \boldsymbol{v}_j$
  $w_j := w_j - \alpha_j \boldsymbol{v}_j - \beta_j \boldsymbol{v}_{j-1}$
  $\beta_{j+1} := \|w_j\|$
  $\boldsymbol{v}_{j+1} := w_j/\beta_{j+1}$
end for

can be computed as

$$T_m = V_m^T A V_m.$$

It is well known that the largest eigenvalues of T_m, $\lambda_n^{(m)}, \lambda_{n-1}^{(m)}, \ldots$ converge, as the size of the Krylov subspace m increases, to the largest eigenvalues of A: $\lambda_m, \lambda_{m-1}, \ldots$, while the corresponding eigenvectors of A can be computed from the homologous eigenvectors of T_m by $\boldsymbol{u}_i = V_m \boldsymbol{u}_i^{(m)}$.

Such a convergence in many cases is very fast: denoted with N_{eig} the number of sought eigenpairs, we experimentally found that $2N_{eig} \div 3N_{eig}$ matrix-vector products (MVP) are usually enough to compute a small number N_{eig} of the rightmost eigenpairs to a satisfactory accuracy. The ratio between the MVP number and N_{eig} is also known to decrease when N_{eig} is increasing. This eigenvalue solver exits whenever the following test is satisfied:

$$\sum_{k=1}^{p} \frac{1}{p} \frac{\|A\boldsymbol{u}_k - \lambda_k \boldsymbol{u}_k\|}{\lambda_k} \leq \delta,$$

with δ a fixed tolerance.

Convergence to the smallest eigenvalues is much slower. Hence, to compute the leftmost part of the spectrum, it is more usual to apply the Lanczos process to the inverse of the coefficient matrix A^{-1}. Since A is expected to be large and sparse, its explicit inversion is not convenient from both CPU time and storage point of view. Algorithm 1, left code, must then be changed since now w_j is computed as the solution of the linear system $Aw_j = \boldsymbol{v}_j$, as reported in Algorithm 1, right code.

We also adopted the implicit-restart strategy as described in [20], a technique to combine the implicitly shifted QR scheme with a k-step Lanczos factorization and obtain a truncated form of the implicitly shifted QR iteration. The numerical difficulties and storage problems normally associated with the Lanczos process are avoided. The algorithm is capable of computing k eigenvalues using storage for only a moderate multiple of k vectors. The computed eigenvectors form a basis for the desired k-dimensional eigenspace and are numerically orthogonal to working precision.

Implicit restart provides a means to extract interesting information from large Krylov subspaces while avoiding the storage and numerical difficulties associated with the standard approach. It does this by continually compressing the interesting information into a fixed-size k-dimensional subspace. This is accomplished through the implicitly shifted QR mechanism. A Lanczos factorization of length $m = k + p$, $AV_m = V_m T_m + r_m e_m$, is compressed to a factorization of length k that retains the eigeninformation of interest.

3.1.1 Solution of the linear system

The linear system solution needed at every Lanczos step can be solved either by a direct method (Cholesky factorization) or by an iterative method such as the Preconditioned Conjugate Gradient (PCG) method. The former approach is unviable if the system matrix size is large (say $n > 10^4 \div 10^5$) due to the excessively dense triangular factor provided by the direct factorization. In such a case the PCG method should be used with the aid of a preconditioner, which speeds ups convergence. We choose the best known multi purpose preconditioner: the incomplete Cholesky factorization with no fill-in. Another advantage of the iterative solution is that the iterative procedure to solve the inner linear system is usually stopped when the following test is satisfied:

$$\frac{\|\boldsymbol{v}_j - A w_j\|}{\|\boldsymbol{v}_j\|} \leq \delta_{PCG}$$

where the tolerance δ_{PCG} can be chosen proportional to the accuracy required for the eigenvectors.

Note Since in our applications matrix A is singular, we computed the incomplete Cholesky factorization for $A + \varepsilon I$ with $\varepsilon = 10^{-8}$ and worked with matrix $A + \boldsymbol{u}\boldsymbol{u}^T$, being $\boldsymbol{u}$ the unit eigenvector corresponding to $\lambda = 0$. Clearly we did not compute explicitly the dense matrix $\widetilde{A} = A + \boldsymbol{u}\boldsymbol{u}^T$; we only needed to provide a routine to multiply such matrix times a vector:

$$\boldsymbol{z} = \widetilde{A}\boldsymbol{w} \qquad \longrightarrow \qquad \alpha = \boldsymbol{u}^T\boldsymbol{w}; \quad \boldsymbol{z} = A\boldsymbol{w} + \alpha\boldsymbol{u}.$$

3.2 Description of the Jacobi–Davidson method

To compute the smallest eigenvalue this method considers the minimization of the Rayleigh Quotient

$$q(x) = \frac{x^T A x}{x^T x}$$

which can be accomplished by setting its gradient to 0, namely

$$Ax - q(x)x = 0. \tag{2}$$

Equation (2) is a nonlinear system of equations which can be solved by means of the classical Newton's method in which the Jacobian of (2) (or the Hessian of the Rayleigh Quotient) is replaced by a simplified formula: $J(u) \approx A - q(u)I$, which is shown to maintain the convergence properties of the Newton's method. The kth iterate of this Newton's method hence reads

$$(A - q(u_k)I)s_k = -(Au_k - q(u_k)u_k) \tag{3}$$

$$u_{k+1} = u_k + s_k. \tag{4}$$

In practice solution of the system (3) is known to produce stagnation in the Newton process. In [25] the authors proposed to use a projected Jacobian namely

$$(I - u_k u_k^T)(A - q(u_k)I)(I - u_k u_k^T)s_k = -(Au_k - q(u_k)u_k) \tag{5}$$

$$u_{k+1} = u_k + s_k \tag{6}$$

ensuring that the search direction s_k be orthogonal to u_k to avoid stagnation.

Even this corrected Newton iteration may be slow, especially if a good starting point is not available. In [25] it is proposed to perform a Rayleigh–Ritz step at every Newton iteration. In

Algorithm 2 Jacobi–Davidson method

Choose unitary starting vector $\boldsymbol{v}$. Initialize empty matrices H, V and W, $k = 0$.
WHILE $k < k_{\max}$

1. $k := k + 1$.
2. Orthogonalize $\boldsymbol{v}$ against V via modified Gram-Schmidt.
3. Normalize v. Compute $\boldsymbol{w} = A\boldsymbol{v}$.
4. $H := \begin{pmatrix} H & V^T\boldsymbol{w} \\ v^T W & v^T w \end{pmatrix}$, $\quad V := [V|v] \quad W := [W|w]$.
5. Compute the smallest eigenpair $(\theta, \boldsymbol{y})$ of H (with $\|\boldsymbol{y}\| = 1$).
6. Compute the vector $\boldsymbol{u} := V\boldsymbol{y}$ and the associated residual vector $\boldsymbol{r} := A\boldsymbol{u} - \theta\boldsymbol{u}$.
7. IF $\|\boldsymbol{r}\| < \varepsilon\ (\theta\|\boldsymbol{u}\|)$ STOP
8. Solve the linear system

$$(I - \boldsymbol{u}\boldsymbol{u}^T)(A - \theta I)(I - \boldsymbol{u}\boldsymbol{u}^T)\boldsymbol{v} = -(A\boldsymbol{u} - \theta\boldsymbol{u}),$$

using the PCG method.

END WHILE

detail, the Newton iterates are collected as columns of a matrix V; then a very small matrix $H = V^T AV$ is computed. The leftmost eigenvector y is easily computed and a new vector u is obtained as $u = Vy$. The main consequence of this procedure is the acceleration of the Newton's method toward the desired eigenvector.

To compute $\lambda_2, \lambda_3, \ldots$, the previous scheme can be used provided that the Jacobian matrix is projected onto a subspace orthogonal to the previously computed eigenvectors. In detail, if λ_j is to be computed, the Newton step reads:

$$(I - QQ^T)(A - q(u_k)I)(I - QQ^T)s_k = -(Au_k - q(u_k)u_k) \tag{7}$$

$$u_{k+1} = u_k + s_k \tag{8}$$

where $Q = [\boldsymbol{v}_1\ \boldsymbol{v}_2\ \ldots\ \boldsymbol{v}_{j-1}\ u_k]$. In order to maintain the dimension of matrix H sufficiently small, two additional parameters are usually introduced. If the size of matrix H is larger than $m_{\max}$ then only the last $m_{\min}$ columns of matrix V are kept.

Even more than in the Lanczos process, the solution of linear system (5) must be found using an iterative method. This system is usually solved to a very low accuracy so that in practice few iterations $(20 \div 30)$ are sufficient to provide a good search direction s_k. Moreover, it has been proved in [24] that linear system (5) can be solved by the PCG method, despite of the fact that the system matrix is not symmetric positive definite. The resulting algorithm is very fast in computing the smallest eigenvalue provided that a good preconditioner is available for the matrix A in order to solve efficiently the system (5).

3.2.1 Comments on the algorithm

The sketch of the Jacobi–Davidson algorithm is reported in Algorithm 2. Step 5 implements the Rayleigh–Ritz projection. It is a crucial step for the convergence of the algorithm, but requires small CPU time since it consists in the eigensolution of the usually very small matrix H.

Step 8 is the most relevant one from the viewpoint of computational cost. A good projected preconditioner should be devised in order to guarantee fast convergence of the PCG method. We used here as the preconditioner $M = (I - \boldsymbol{u}\boldsymbol{u}^T)P(I - \boldsymbol{u}\boldsymbol{u}^T)$, with P the same incomplete Cholesky factorization employed by IRLM.

Algorithm 3 Deflation Accelerated Conjugate Gradient method

Choose tolerance ε, set $U = 0$.

DO $j = 1, p$

1. Choose $\boldsymbol{x}_0$ such that $U^T \boldsymbol{x}_0 = 0$; set $k = 0$, $\beta_0 = 0$;
2. Find the minimum of the Rayleigh Quotient $q(\boldsymbol{x}) = \dfrac{\boldsymbol{x}^T A \boldsymbol{x}}{\boldsymbol{x}^T \boldsymbol{x}}$ for every $\boldsymbol{x}$ such that $U^T \boldsymbol{x}_0 = 0$ by a nonlinear preconditioned conjugate gradient procedure.
3. Stop whenever the following test is satisfied:

$$\frac{\|A\boldsymbol{x}_k - q_k \boldsymbol{x}_k\|}{q(\boldsymbol{x}_k)} \leq \varepsilon$$

4. Set $\lambda_j = q_k, \quad \boldsymbol{u}_j = \boldsymbol{x}_k/\sqrt{\eta}, \quad U = [U, \boldsymbol{u}_j]$.

END DO

For the details of this method we refer to the paper [25], as well as to successive works by [13,24,26] who analyze both theoretically and experimentally a number of variants of this well known method.

3.3 Description of the deflation accelerated conjugate gradient method

Instead of minimizing $q(\boldsymbol{x})$ by Newton's method the nonlinear Conjugate Gradient method can be employed. Differently from the two methods just described, this one does not need any linear system solution. Like the JD method, Deflation Accelerated Conjugate Gradient (DACG) computes the eigenvalues sequentially, starting from the smallest one [5,7]. The leftmost eigenpairs are computed sequentially, by minimizing the Rayleigh Quotient over a subspace orthogonal to the previously computed eigenvectors. Although not as popular as IRLM and JD, this method, which applies only to symmetric positive definite matrices, has been proven very efficient in the solution of eigenproblems arising from discretization of Partial Differential Equations (PDEs) in [8] DACG also proved very suited to parallel implementation as documented in [6] where an efficient parallel matrix vector product has been employed.

Convergence of DACG is strictly related to the relative separation between consecutive eigenvalues, namely

$$\xi_j = \frac{\lambda_j}{\lambda_{j+1} - \lambda_j}. \tag{9}$$

When two eigenvalues are relatively very close, DACG convergence may be very slow. Also DACG takes advantage of preconditioning which, as in the two previous approaches, can be chosen to be the Incomplete Cholesky factorization.

3.3.1 Comments on the algorithm

The DACG procedure is described in Algorithm 3. The PCG minimization of the Rayleigh Quotient (Step 2) is carried out by performing a number of iterations. The main computational burden of a single iteration is represented by:

1. One matrix-vector product.
2. One application of the preconditioner.

3. Orthogonalization of the search direction against the previously computed eigenpairs (columns of matrix U). The cost of this step is increasing with the number of eigenpairs begin sought.

As common in the iterative methods, the number of iterations can not be known in advance. However, it is known to be proportional to the reciprocal of the relative separation ξ between consecutive eigenvalues (Eq. (9)).

4 Numerical results and comparisons

In this section we experimentally compare the three previously described solvers in the computation of some of the leftmost eigenpairs of a number of Laplacian matrices of graphs G arising from the following realistic applications covering all four main categories of real networks, namely biological networks, technological networks, information networks, and social networks:

1. Matrix `protein` represents the Laplacian of the protein-protein interaction network of yeast [18]. In a protein-protein interaction network the vertices are proteins and two vertices are connected by an undirected edge if the corresponding protein interact.
2. Matrix `internet` is the Laplacian of a symmetrized snapshot of the structure of the Internet at the level of autonomous systems, reconstructed from BGP tables posted by the University of Oregon Route Views Project. This snapshot was created by Mark Newman and is not previously published.
3. Matrix `www` is the Laplacian of the Web network within nd.edu domain [1]. This network is directed but arc direction has been ignored in order to obtain a symmetric Laplacian.
4. Matrix `dblp` is the Laplacian of a graph describing collaboration network of computer scientists. Nodes are authors and edges are collaborations in published papers, the edge weight is the number of publications shared by the authors [14].

In Table 1 we report the number of matrix rows (n), the number of matrix nonzero entries (nnz), the average nonzeros per row (anzr), which account for the sparsity of the matrix, and the ratio λ_{51}/λ_2 (gap), which indicates how the 50 smallest nonzero eigenvalues are separated. Note that the number of nonzeros is computed as $nnz = n + 2m$ where m is the number of arcs in the graph.

We also report the distribution of the first 50 normalized eigenvalues for the four test problems in Fig. 1. As mentioned before, a pronounced relative separation between consecutive eigenpairs may suggest fast convergence of the iterative procedures, and particularly so for the DACG method. We notice from Fig. 1 that for three problems out of four, with the exception of matrix `www`, the eigenvalues are clustered, thus suggesting a slow convergence of the iterative solvers. This fact is also accounted for by the ratios λ_{51}/λ_2 provided by Table 1. The smallest this ratio, the slowest the convergence to the desired eigenvalues.

Table 1 Main characteristics of the sample matrices: size (n), number of nonzero entries (nnz), average nonzeros per row (anzr) and ratio between the largest and smallest computed eigenvalues (gap)

Matrix	n	nnz	anzr	gap
`protein`	1453	5344	3.7	7.28
`internet`	22,963	119,835	5.2	4.39
`www`	325,729	2,505,945	7.8	23.25
`dblp`	928,498	8,628,378	9.3	2.11

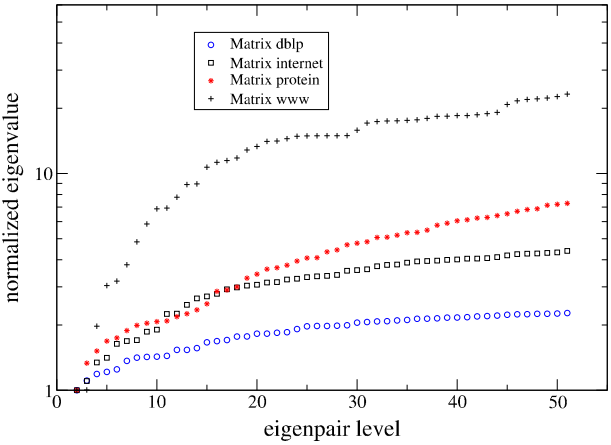

Fig. 1 Semilog plot of the distribution of the 50 smallest normalized eigenvalues (λ_j/λ_2) of the four test matrices

In the JD implementation two parameters are crucial for its efficiency namely $m_{\min}$ and $m_{\max}$, the smallest and the largest dimension of the subspace where the Rayleigh Ritz projection takes place. After some attempts, we found that $m_{\min} = 5$ and $m_{\max} = 10$ were on the average the optimal values of such parameters. As for the solution of the Newton linear systems we choose as the maximum number of inner iterations $ITMAX = 20$ and we use as the accuracy for the inner linear solver $\delta_{PCG} = 10^{-2}$. Regarding IRLM parameters, we set $\delta_{PCG} = 10^{-2} \times \delta$, since the iterative solution of the inner linear system must be run to a higher accuracy than that required for the eigenpairs. The dimension of the Krylov subspace `ncv` for the restarted Lanczos iteration has been chosen as `ncv` = 15, 30, 60, 120, for $N_{eig} = 1, 5, 20, 50$, respectively. The previously described parameters regard memory storage and efficiency for both JD and IRLM. JD is usually less demanding than IRLM in terms of memory storage. If N_{eig} eigenvectors are to be computed, the Jacobi–Davidson method requires saving of at most $N_{eig} + m_{\max} = N_{eig} + 10$ dense vectors. For both solvers three more vectors are needed by the PCG implementation. Also the fact that the inner linear system has to be solved much more accurately by IRLM is accounted for by the choice of parameter δ_{PCG}. The DACG code requires the storage of only $N_{eig} + 5$ vectors, begin therefore the less demanding algorithm in terms of memory occupancy.

The three solvers, IRLM, DACG and JD have been preconditioned by $K^{-1} = \left(LL^T\right)^{-1}$ being L the lower triangular factor of the Cholesky factorization of A with no fill-in. We reported the results in computing the smallest strictly positive $N_{eig} = 1, 5, 20, 50$ eigenvalues with tolerances $\delta = 10^{-3}, 10^{-6}$ for the relative residual using as the exit test:

$$\frac{\|A\boldsymbol{u}_k - \theta_k \boldsymbol{u}_k\|}{\theta_k} < \delta$$

where $\theta_k = \boldsymbol{u}_k{}^T A \boldsymbol{u}_k$ is the approximation of the eigenvalue being $\boldsymbol{u}_k$ normalized.

The results regarding the four test problems are summarized in Tables 2, 3, 4 and 5, respectively, where we report CPU times and number of matrix vector products (MVP) for the three codes. The number of linear system solutions of IRLM and JD are also provided (outer iterations). Notice that DACG does not need to solve any linear system. The Fortran

Table 2 Number of linear system solutions (outer its), number of matrix vector products (MVP), and CPU times for DACG, JD, and IRLM on matrix `protein` for the computation of the 50 smallest eigenvalues with two different accuracies (δ)

δ	DACG		JD			IRLM		
	MVP	CPU	Outer its	MVP	CPU	Outer its	MVP	CPU
10^{-3}	3623	0.46	191	2501	**0.44**	155	2403	0.45
10^{-6}	6513	0.82	293	4084	**0.68**	175	4381	0.72

Table 3 Number of linear system solutions (outer its), number of matrix vector products (MVP), and CPU times for DACG, JD, and IRLM on matrix `internet` for the computation of the smallest 1, 5, 20 and 50 eigenvalues with two different accuracies (δ)

N_{eig}	δ	DACG		JD			IRLM		
		MVP	CPU	Outer its	MVP	CPU	Outer its	MVP	CPU
1	10^{-3}	54	**0.48**	13	162	0.66	21	939	2.23
5	10^{-3}	324	**0.97**	33	337	1.04	29	1176	2.86
20	10^{-3}	1394	3.12	82	1030	**2.89**	82	2231	7.41
50	10^{-3}	4090	10.66	197	2383	**7.71**	202	5258	19.29
1	10^{-6}	93	**0.50**	15	159	0.67	27	1927	4.40
5	10^{-6}	589	1.42	32	454	**1.26**	40	2904	6.48
20	10^{-6}	2701	5.41	116	1585	**3.98**	96	6724	15.72
50	10^{-6}	7591	18.97	285	4266	**12.93**	211	14,578	36.31

Table 4 Number of linear system solutions (outer its), number of matrix vector products (MVP), and CPU times for DACG, JD, and IRLM on matrix `www` for the computation of the smallest 1, 5, 20 and 50 eigenvalues with two different accuracies (δ)

N_{eig}	δ	DACG		JD			IRLM		
		MVP	CPU	Outer its	MVP	CPU	Outer its	MVP	CPU
1	10^{-3}	1262	**50**	34	1631	90	37	18,999	655
5	10^{-3}	5835	246	63	2958	**169**	49	24,023	870
20	10^{-3}	23,733	1211	103	9851	**604**	81	36,483	1231
50	10^{-3}	64,197	4086	504	22,993	**1757**	120	53,742	2289
1	10^{-6}	2121	**96**	44	2143	143	40	2502	860
5	10^{-6}	9058	428	109	5359	**358**	52	31,244	1147
20	10^{-6}	38,544	2329	373	18,317	**1411**	97	56,578	2217
50	10^{-6}	143,102	10,935	986	44,469	**3545**	150	89,675	3853

implementations of the three solvers have been run on an IBM Power6 at 4.7 GHz and with up to 64 Gb of RAM. The CPU times are expressed in seconds.

Regarding the small-size `protein` we only report the results of the computation of 50 eigenpairs since computing $N_{eig} = 1, 5$ and 20 eigenvalues is done by every solver in an

Table 5 Number of linear system solutions (outer its), number of matrix vector products (MVP), and CPU times for DACG, JD, and IRLM on matrix `dblp` for the computation of the smallest 1, 5, 20 and 50 eigenvalues with two different accuracies (δ)

N_{eig}	δ	DACG		JD			IRLM		
		MVP	CPU	Outer its	MVP	CPU	Outer its	MVP	CPU
1	10^{-3}	178	**79**	10	148	82	17	1359	421
5	10^{-3}	797	286	31	450	**206**	41	2979	937
20	10^{-3}	3808	1324	107	1675	**745**	103	8001	2447
50	10^{-3}	11,239	5231	315	5384	**2463**	225	11,111	4033
1	10^{-6}	1000	319	16	244	**118**	24	3373	1021
5	10^{-6}	2142	690	48	803	**340**	43	5975	1813
20	10^{-6}	7810	2709	171	2981	**1257**	110	15,628	4728
50	10^{-6}	24,978	10,571	543	9654	**4665**	270	34,445	11,396

almost negligible CPU time on our computer. Table 2 shows a very similar behavior of the three solvers both in terms of MVPs and CPU time.

Analyzing the results in Tables 3, 4, and 5 we can make the following observations:

1. The IRLM is almost always slower than the remaining two. This occurs since it requires many accurate inner linear system solutions. Actually IRLM implicitly computes the largest eigenpairs of A^{-1}. The latter matrix is not explicitly formed since it would have an excessive number of nonzeros. Only the action of A^{-1} upon a vector is computed as a linear system solution. Solving this system to a low accuracy would mean compute eigenpairs of a matrix different from A thus introducing unacceptable errors. However, it can be observed that the distance between IRLM and the other two solvers decreases as the number of sought eigenpairs increases.
2. The JD algorithm displays the best performance in terms of number of MVP and CPU time. In particular on the largest problem, it neatly outperforms both DACG and IRLM. The JD method inherits the nice convergence properties of the Newton's method enhanced by the Rayleigh–Ritz acceleration. Moreover, it allows very inaccurate (and hence very cheap) solution of the inner linear system.
3. The DACG algorithm provides comparable performances with JD for a very small number of eigenpairs (up to 5) and particularly when the eigenvalues are needed to a low accuracy. When the number of eigenvalues is large, the reorthogonalization cost prevails and makes this algorithm not competitive. For the sample tests presented in this paper, the DACG method is also penalized by the clustering of eigenvalues which results in a very small relative separation between consecutive eigenvalues. This argument applies also to matrix `www` where, apart of the first 10 eigenvalues which are relatively well separated, the remaining ones are as clustered as those of the other test problems, see Fig. 1.
4. When few eigenpairs are to be computed (and hence the reorthogonalization cost is not prevailing) JD does not seem particularly sensitive to eigenvalue accuracy. This is not surprising as it is based on a Newton iteration. This process is known to converge very rapidly in a neighborhood of the solution. For this reason, the transition between $\delta = 10^{-3}$ and 10^{-6} tolerance is very fast.
5. Despite of the favorable distribution of the leftmost part of its eigenspectrum (see Fig. 1), the number of iterations to eigensolve matrix `www` is high for all the three solvers. For this

test problem, the incomplete Cholesky factorization with no fill-in preconditioner does not provide a satisfactory acceleration. The choice of a suitable preconditioner is crucial for the convergence of all iterative methods. Devising a more "dense" preconditioner would improve the performance of all the methods described and particularly so for IRLM and JD that explicitly require a linear system solution.

4.1 Related work

We selected to use for the three methods the established implementations without making optimization to any of them. However, it is worth mentioning that much work is being devoted particularly to the Arnoldi method (the non symmetric counterpart of the Lanczos Method) in order to reduce its computational cost and memory storage. We refer e.g. to the recent work [15].

Other methods are efficiently employed for computing a number of eigenpairs of sparse matrices. Among these, we mention the Rayleigh Quotient iteration whose inexact variant has been recently analyzed in [27]. A method which has some common features with DACG is LOBPCG (Locally Optimal Block Preconditioned Conjugate Gradient Method) which has been proposed in [19], and is currently available under the `hypre` package [11]. We did not report any results with the LOBPCG package mainly since this code has been previously compared with the DACG code in [6] displaying comparable performances on a large set of test problems.

Another important class of iterative eigensolvers, which have not been reviewed in this manuscript, are the block variants of e.g. DACG and Jacobi–Davidson, see [2]. The block counterparts of the algorithms described in this paper are expected to be useful, particularly due to the high clustering of the smallest eigenvalues in our graph-based matrices. We leave a further comparison with such codes to a future work.

4.2 Preconditioners

We selected to use the incomplete Cholesky factorization with no fill-in as the preconditioner for all the iterative eigensolvers. It is worth noticing that selection of more appropriate preconditioners could change the experimental findings of this work. Some recent papers related to the acceleration of the Arnoldi method (we quote e.g. [15,21]) show that the performance of this method may be greatly improved by employing low-rank variation of a given initial preconditioner. At our knowledge, neither for DACG nor for JD similar preconditioned strategies could be carried on.

5 Conclusion

We experimentally compare three important iterative algorithms for computation of eigenpairs of large and sparse matrices: the Implicitly Restarted Lanczos Method, the Jacobi–Davidson method, and the Deflation Accelerated Conjugate Gradient method. We uniformly implemented the algorithms and ran them in order to compute some of the smallest eigenpairs of the Laplacian matrix of real-world networks of different sizes.

The iterative approach followed in this work seems to be particularly suited for the Laplacian matrices presented since it fully exploits the sparsity of the matrices involved. Each of our realistic test cases, indeed, has a very high degree of sparsity as accounted for by the very small number of nonzeros per row.

Contrary to what observed for matrices arising from discretization of Partial Differential Equations [8], where especially the smallest eigenvalues are well separated, here the high clustering of lowest eigenvalues is disadvantageous for the Deflation Accelerated Conjugate Gradient algorithm. As for the Implicitly Restarted Lanczos Method, the need to solve the inner linear systems to a high accuracy makes this method less attractive for large eigenproblems. The Jacobi–Davidson procedure is less sensitive to the clustering thanks to the Rayleigh–Ritz projection; moreover, for this method, inexact and hence efficient solution of the inner linear systems is sufficient to achieve overall convergence. All in all, the Jacobi–Davidson algorithm is performing the best on our test problems.

We finally remark that on the basis of the previously cited recent works, such as [15,27], the proposed implementations of inexact variants of Implicitly Restarted Lanczos Method could make this solver competitive respect to Jacobi–Davidson, if a relatively large number of eigenpairs is desired.

All the proposed algorithms are well-suited to parallelization on supercomputers. The most important kernel is represented by the matrix-vector product which can be efficiently implemented in parallel environments. Also application of preconditioner, which is one of the most time-consuming task, in its turn can be devised as a product of sparse matrices as e.g. in the "approximate inverse preconditioner" approach (see the review article by [4]). The DACG method has been successfully parallelized as documented in [6], however all the iterative solvers described here, being based on the same linear algebra kernels, could be implemented in parallel with the same satisfactory results.

In our implementation we used a general purpose preconditioner, obtained by means of incomplete Cholesky factorization with no fill in. Certainly, the three methods would greatly benefit from the use of a more specific preconditioner. This point will be a topic of future research.

References

1. Albert, R., Jeong, H., Barabási, A.-L.: Diameter of the world-wide web. Nature **401**, 130–131 (1999)
2. Arbenz, P., Hetmaniuk, U., Lehoucq, R., Tuminaro, R.: A comparison of eigensolvers for large-scale 3D modal analysis using AMG-preconditioned iterative methods. Int. J. Numer. Methods Eng. **64**, 204–236 (2005)
3. Ben-Israel, A., Greville, T.: Generalized inverses. In: CMS Books in Mathematics/Ouvrages de Mathématiques de la SMC, vol. 15. Springer, New York (2003)
4. Benzi, M.: Preconditioning techniques for large linear systems: a survey. J. Comput. Phys. **182**, 418–477 (2002)
5. Bergamaschi, L., Gambolati, G., Pini, G.: Asymptotic convergence of conjugate gradient methods for the partial symmetric eigenproblem. Numer. Linear Algebra Appl. **4**, 69–84 (1997)
6. Bergamaschi, L., Martínez, A., Pini, G.: Parallel Rayleigh quotient optimization with FSAI-based preconditioning. J. Appl. Math., Article ID 872901, 14 pp. (2012)
7. Bergamaschi, L., Pini, G., Sartoretto, F.: Approximate inverse preconditioning in the parallel solution of sparse eigenproblems. Numer. Linear Algebra Appl. **7**, 99–116 (2000)
8. Bergamaschi, L., Putti, M.: Numerical comparison of iterative eigensolvers for large sparse symmetric matrices. Comput. Methods Appl. Mech. Eng. **191**, 5233–5247 (2002)
9. Bozzo, E., Franceschet, M.: Approximations of the generalized inverse of the graph Laplacian matrix. Internet Math. **8**, 1–26 (2012)
10. Brouwer, A.E., Haemers, W.: Spectra of Graphs. Springer, Berlin (2012)
11. Falgout, R.D., Jones, J.E., Yang, U.M.: Pursuing scalability for hypre's conceptual interfaces. ACM. Trans. Math. Softw. **31**(3), 326–350 (2005)
12. Fiedler, M.: Algebraic connectivity of graphs. Czechoslov. Math. J. **23**, 298–305 (1973)
13. Fokkema, D.R., Sleijpen, G.L.G., van der Vorst, H.A.: Jacobi–Davidson style QR and QZ algorithms for the reduction of matrix pencils. SIAM J. Sci. Comput. **20**, 94–125 (1998) **(electronic)**

14. Franceschet, M.: Collaboration in computer science: a network science approach. J. Am. Soc. Inf. Sci. Technol. **62**, 1992–2012 (2011)
15. Freitag, M.A., Spence, A.: Shift-invert Arnoldi's method with preconditioned iterative solves. SIAM J. Matrix Anal. Appl. **31**, 942–969 (2009)
16. Ghosh, A., Boyd, S., Saberi, A.: Minimizing effective resistance of a graph. SIAM Rev. **50**, 37–66 (2008)
17. Hu, Y., Scott, J.: HSLssMC73: A Fast Multilevel Fiedler and Profile Reduction Code. Rutherford Appleton Laboratory, Oxfordshire (2003)
18. Jeong, H., Mason, S., Barabási, A.-L., Oltvai, Z.: Lethality and centrality in protein networks. Nature **411**, 41–42 (2001)
19. Knyazev, A.: Toward the optimal preconditioned eigensolver: locally optimal block preconditioned conjugate gradient method. SIAM J. Sci. Comput. **23**, 517–541 (2001)
20. Lehoucq, R.B., Sorensen, D.C., Yang, C.: ARPACK Users Guide. Solution of Large Scale Eigenvalue Problem with Implicit Restarted Arnoldi Methods. SIAM, Philadelphia (1998)
21. Martínez, A.: Tuned preconditioners for iterative SPD eigensolvers. Numer. Linear Algebra Appl. **23**, 427–443 (2016)
22. Mieghem, P.V.: Graph Spectra for Complex Networks. Cambridge University Press, New York (2011)
23. Newman, M.: Networks: An Introduction. Oxford University Press Inc, New York (2010)
24. Notay, Y.: Combination of Jacobi-Davidson and conjugate gradients for the partial symmetric eigenproblem. Numer. Linear Algebra Appl. **9**, 21–44 (2002)
25. Sleijpen, G.L.G., van der Vorst, H.A.: A Jacobi-Davidson method for linear eigenvalue problems. SIAM J. Matrix Anal. **17**, 401–425 (1996)
26. Stathopoulos, A.: A case for a biorthogonal Jacobi–Davidson method: restarting and correction equation. SIAM J. Matrix Anal. Appl. **24**, 238–259 (2002) **(electronic)**
27. Xue, F., Elman, H.C.: Convergence analysis of iterative solvers in inexact Rayleigh quotient iteration. SIAM J. Matrix Anal. Appl. **31**, 877–899 (2009)