Fast algorithm for relaxation processes in big-data systems

S. Hwang,[1] D.-S. Lee,[2,*] and B. Kahng[1,†]

[1]*Department of Physics and Astronomy, Seoul National University, Seoul 151-747, Korea*

[2]*Department of Physics and Department of Natural Medical Sciences, Inha University, Incheon 402-751, Korea*

(Received 19 March 2014; revised manuscript received 11 August 2014; published 6 October 2014)

Relaxation processes driven by a Laplacian matrix can be found in many real-world big-data systems, for example, in search engines on the World Wide Web and the dynamic load-balancing protocols in mesh networks. To numerically implement such processes, a fast-running algorithm for the calculation of the pseudoinverse of the Laplacian matrix is essential. Here we propose an algorithm which computes quickly and efficiently the pseudoinverse of Markov chain generator matrices satisfying the detailed-balance condition, a general class of matrices including the Laplacian. The algorithm utilizes the renormalization of the Gaussian integral. In addition to its applicability to a wide range of problems, the algorithm outperforms other algorithms in its ability to compute within a manageable computing time arbitrary elements of the pseudoinverse of a matrix of size millions by millions. Therefore our algorithm can be used very widely in analyzing the relaxation processes occurring on large-scale networked systems.

DOI: 10.1103/PhysRevE.90.043303 PACS number(s): 05.10.Cc, 05.40.−a, 89.75.Hc

I. INTRODUCTION

Fast analyses of big data sets [1] are increasingly requested in diverse interdisciplinary area in this information era. Given the limitations of available computing resources in space and time, designing and implementing scalable and efficient algorithms are essential for practical applications. One of the tasks most often encountered in such problems is the analysis of huge sparse matrices, for example, the Laplacian matrix $\boldsymbol{L}$ of a large-scale complex network. The matrix $\boldsymbol{L}$ plays important roles in a wide range of problems such as diffusion processes, random walks [2,3], search engines on web pages [4], synchronization phenomena [5], epidemics [6], and load balancing in parallel computing [7]. For instance, the spectrum of a Laplacian matrix determines the number of minimum spanning tree, minimal cuts [8,9], and Kirchhoff index [10].

The elements of the Laplacian matrix $\boldsymbol{L}$ of a given network are represented as $L_{ij} = \delta_{ij} - A_{ij}/k_j$ with the degree of node j given by $k_j = \sum_\ell A_{j\ell}$. The Laplacian matrix has a couple of remarkable features. It has positive eigenvalues and one nondegenerate zero eigenvalue. The zero eigenvalue appears since $\sum_i \boldsymbol{L}_{ij} = 0$, related to, e.g., the probability conservation in the context of random walks and diffusion. Also, the Laplacian matrix can be symmetrized as $\bar{\boldsymbol{L}} = \boldsymbol{S}\boldsymbol{L}\boldsymbol{S}^{-1}$, whose element is given as $\bar{L}_{ij} = \delta_{ij} - A_{ij}/\sqrt{k_i k_j}$ for $S_{ij} = k_i^{-1/2}\delta_{ij}$. This symmetrization can be performed not only for the Laplacian matrix but also for all the generators $\boldsymbol{V}$ of Markov chains satisfying the detailed-balance condition [11], the definition of which will be explained in detail later. In this paper, we propose an algorithm for computing the generalized inverse, so-called Moore-Penrose pseudoinverse of those generators, which is relevant to the first passage property and the correlation function of the Markov chains and therefore has been extensively studied in the physics context [12–16].

If one uses the standard eigendecomposition method based on the QR algorithm [17], it needs $O(N^2)$ memory space and takes $O(N^3)$ computing time to obtain the inverse of a $N \times N$ matrix. Therefore this algorithm cannot be actually applied to obtain the inverse of large-size matrices, and faster algorithms have been developed to solve specific problems handling large sparse matrices. For instance, the iterative methods such as the well-known Jacobi method or the Krylov subspace method [18] are very efficient for the linear problem $\boldsymbol{M}\,|x\rangle = |b\rangle$. In the Euclidean lattice, the Fourier acceleration method has been introduced to overcome slow convergence of the Jacobi method for random resistor networks embedded in the Euclidean space [19–21]. Also the graph theoretic methods such as the fast inverse using nested dissection (FIND) are known to be efficient for computing the inverse of large sparse positive-definite matrices [22–26], most of which are useful in a two-dimensional lattice. The pseudoinverse of singular matrices has been investigated [27–29] and can be obtained efficiently for each specific domain of strength such as for bipartite graphs [27], the linear problem of the Laplacian matrix [28], or the Laplacian-specific method [29].

Our algorithm can be used for a wide range of problems effectively; it enables one to obtain a set of $O(N)$ arbitrary elements of the pseudoinverse of a class of $N \times N$ matrices within the computing time much shorter than $O(N^3)$ in most cases. Note that the solution to a single linear problem cannot provide a set of arbitrary elements of the pseudoinverse in a single run. The class of the singular matrices we consider here are the generators of the Markov chains satisfying the detailed-balance condition. The algorithm exploits the fact that the Gaussian integral with a coupling matrix $\boldsymbol{H}$ under external fields turns into a Gaussian function of the external-field variables with the coupling matrix given by $\boldsymbol{H}^{-1}$. The coupling matrix $\boldsymbol{H}$ is constructed from a given generator matrix $\boldsymbol{V}$. Its Gaussian integral is evaluated by decimating the variables and renormalizing the coupling matrix with an appropriate treatment of the zero eigenvalue mode of V.

To verify the usefulness and performance of the proposed algorithm in physics problems, we apply the algorithm to compute the global mean first passage time (GMFPT) of

*deoksun.lee@inha.ac.kr

†bkahng@snu.ac.kr

a random walk on various networks, which requires the computation of all the diagonal elements of the pseudoinverse of the generator, the Laplacian matrix. We compare the computational cost of our algorithm with that of the QR algorithm for small system sizes and that of the random-walk simulation. The dependence of network topology on the computing time of our algorithm is discussed.

This paper is organized as follows. In Sec. II we introduce the basic formulas of the Gaussian integral, which play the central roles in designing our algorithm. Before presenting the main algorithm, the one computing the inverse of a positive-definite matrix is outlined in Sec. III. In Sec. IV we specify a target problem of our algorithm, of which the pseudoinverse can be obtained exactly by our algorithm. The applications of the pseudoinverse in the physics context are also presented. In Sec. V we describe each procedure of the algorithm in detail. In VI the running time of our algorithm to compute the GMFPT on various model networks is presented and compared with that of other methods. The algorithm is applied to large real-world networks, demonstrating its practical use in the same section. The summary and discussion are given in Sec. VII.

II. BASIC FORMULATION

Here we present the formulas which will be used in our algorithm. For an $N \times N$ nonsingular real symmetric matrix $\boldsymbol{H}$ and an arbitrary column vector $|J\rangle$ of size N, we consider the Gaussian integral given by

$$Z \equiv \int_{-\infty}^{\infty} \prod_{j=1}^{N} d\phi_j \exp\left[\frac{i}{2}\langle\phi|\,\boldsymbol{H}\,|\phi\rangle + i\,\langle J|\phi\rangle\right] = \sqrt{\frac{(2\pi i)^N}{\det \boldsymbol{H}}}\, e^{-\frac{i}{2}\langle J|\boldsymbol{H}^{-1}|J\rangle}, \tag{1}$$

where $|\phi\rangle = (\phi_1,\phi_2,\ldots,\phi_N)^\dagger$ and the factor i is introduced for the convergence of the integral. Once the Gaussian integral Z is evaluated, the inverse matrix $\boldsymbol{H}^{-1}$ can be obtained by

$$H^{-1}_{j\ell} = -i\,\frac{\partial^2}{\partial J_j \partial J_\ell} \log Z\bigg|_{|J\rangle=|0\rangle}, \tag{2}$$

where $|0\rangle$ is a null vector. If we introduce a $2N$-dimensional vector $|\psi\rangle$ by gluing $|J\rangle$ and $|\phi\rangle$ as

$$\psi_j = \begin{cases} J_j & \text{for} \quad 1 \leqslant j \leqslant N \\ \phi_{j-N} & \text{for} \quad N+1 \leqslant j \leqslant 2N \end{cases}, \tag{3}$$

and a $2N \times 2N$ real symmetric matrix $\tilde{\boldsymbol{H}}$

$$\tilde{H}_{j\ell} = \begin{cases} \delta_{j\ell} & \text{for} \quad 1 \leqslant j,\ell \leqslant N \\ H_{j-N,\ell-N} & \text{for} \quad N+1 \leqslant j,\ell \leqslant 2N, \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

we can represent Eq. (1) in a simple form as

$$Z = \int_{-\infty}^{\infty} \prod_{\ell=N+1}^{2N} \mathrm{d}\psi_\ell \exp\left[\frac{i}{2}\langle\psi|\,\tilde{\boldsymbol{H}}\,|\psi\rangle\right]. \tag{5}$$

The evaluation of the Gaussian integral in Eq. (5) can be done by integrating out ψ variables one by one and renormalizing the elements of $\tilde{\boldsymbol{H}}$ accordingly. The matrix $\tilde{\boldsymbol{H}}$ thus reduces its dimension by one at every stage. Some of the zero elements in $\tilde{\boldsymbol{H}}$ can be nonzero after such renormalization, which should be taken care of as detailed in the next section. For an extended coupling matrix $\tilde{\boldsymbol{H}}$, we consider a graph G with the adjacency matrix $\boldsymbol{A}$ with its elements given by

$$A_{j\ell} = \begin{cases} 1 & \text{if} \quad \tilde{H}_{j\ell} \neq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{6}$$

G and $\boldsymbol{A}$ then evolve as $\tilde{\boldsymbol{H}}$ is renormalized successively.

III. OUTLINE OF THE ALGORITHM FOR THE INVERSE OF A POSITIVE-DEFINITE MATRIX

In this section we outline the algorithm for computing the inverse of a positive definite matrix $\boldsymbol{H}$ by evaluating the Gaussian integral in Eq. (5), which will be generalized to singular matrices in Sec. V. For an $N \times N$ positive definite matrix $\boldsymbol{H}$, following Eq. (4), we construct the extended matrix $\tilde{\boldsymbol{H}}$ of $2N \times 2N$. The corresponding graph G of $2N$ vertices has the adjacency matrix $\boldsymbol{A}$ as in Eq. (6).

Suppose that we integrate out ψ_{2N} in 5 to transform $\tilde{\boldsymbol{H}}$ into $\tilde{\boldsymbol{H}}^{(1)}$ of size $(2N-1)\times(2N-1)$. Since $\boldsymbol{H}$ is positive definite, $\tilde{H}_{2N\,2N}$ is positive. Collecting the terms involving ψ_{2N}, we find that

$$\int_{-\infty}^{\infty} d\psi_{2N} \exp\left(\frac{i}{2}\tilde{H}_{2N\,2N}\psi_{2N}^2 + i\,B_{2N}\psi_{2N}\right) = \sqrt{\frac{2\pi i}{\tilde{H}_{2N\,2N}}} \exp\left(-\frac{i}{2}\frac{B_{2N}^2}{\tilde{H}_{2N\,2N}}\right), \tag{7}$$

where $B_{2N} \equiv \sum_j A_{j\,2N}\tilde{H}_{j\,2N}\psi_j$. Noting that $B_{2N}^2 = \sum_{j,\ell} A_{j\,2N}A_{\ell\,2N}\tilde{H}_{j\,2N}\tilde{H}_{\ell\,2N}\psi_j\psi_\ell$, one can identify the renormalized Hamiltonian $\tilde{\boldsymbol{H}}^{(1)}$ in the Gaussian integral as

$$Z = \sqrt{\frac{2\pi i}{\tilde{H}_{11}}}\, Z^{(1)}, \qquad Z^{(1)} = \int_{-\infty}^{\infty} \prod_{j=N+1}^{2N-1} \mathrm{d}\psi_j \exp\left[\frac{i}{2}\langle\psi|\,\tilde{\boldsymbol{H}}^{(1)}\,|\psi\rangle\right], \tag{8}$$

where $\tilde{H}^{(1)}_{j\ell} = \tilde{H}_{j\ell}$ for all $1 \leqslant j,\ell \leqslant (2N-1)$ unless both j and ℓ are the neighbor nodes of the decimated node $2N$ in G, the graph representation of $\tilde{\boldsymbol{H}}$. If $A_{j\,2N}A_{\ell\,2N} > 0$, the corresponding matrix element $\tilde{H}_{j\ell}$ is changed to $\tilde{H}^{(1)}_{j\ell} = \tilde{H}_{j\ell} - \tilde{H}_{j\,2N}\tilde{H}_{\ell\,2N}/\tilde{H}_{2N\,2N}$. In the graph representation, the corresponding graph G is transformed to $G^{(1)}$ by eliminating the node $2N$ and adding links to every pair of the nodes that were adjacent to the node $2N$ but disconnected in G (see Fig. 1). Accordingly, the adjacency matrix evolves from $\boldsymbol{A}$ to $\boldsymbol{A}^{(1)}$.

We repeat this procedure, decimation followed by renormalization, N times to integrate out all $\psi_j = \phi_{j-N}$ variables for $N+1 \leqslant j \leqslant 2N$. Consequently the extended matrix $\boldsymbol{H}$

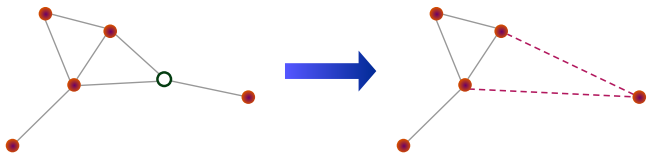

FIG. 1. (Color online) Example of eliminating a node in a graph. When a node (open circle) is eliminated, new links (dashed lines) are added to the pairs of its neighbor nodes if disconnected. Consequently, the neighbor nodes form a clique, a completely connected subgraph.

evolves as

$$\tilde{\boldsymbol{H}}^{(0)} = \tilde{\boldsymbol{H}} \to \tilde{\boldsymbol{H}}^{(1)} \to \cdots \to \tilde{\boldsymbol{H}}^{(N-1)} \to \tilde{\boldsymbol{H}}^{(N)} = \boldsymbol{H}^{-1} \quad (9)$$

while reducing its dimension from $2N$ to N. The $N \times N$ matrix $\tilde{\boldsymbol{H}}^{(N)}$ obtained at the last stage represents the coupling between J and is equal to $\boldsymbol{H}^{-1}$ in Eq. (1). The adjacency matrix $\boldsymbol{A}$ and the graph G also evolve as $\boldsymbol{A} = \boldsymbol{A}^{(0)} \to \boldsymbol{A}^{(1)} \to \cdots \to \boldsymbol{A}^{(N)}$ and $G = G^{(0)} \to G^{(1)} \to \cdots \to G^{(N)}$, respectively.

The order of decimating nodes affects significantly the running time of the algorithm and will be discussed in detail later. Once it is determined, one can rearrange the node indices of $\tilde{\boldsymbol{H}}$ such that nodes are eliminated from $j = 2N$ to $j = N+1$. That is, if we introduce v_n, the index of the node that is eliminated in $G^{(n)}$ to obtain $G^{(n+1)}$; it is given by $v_n = 2N - n$ for $n = 0,1,\ldots,N-1$. The matrix $\tilde{\boldsymbol{H}}^{(n+1)}$ is obtained by removing the last row and column of $\tilde{\boldsymbol{H}}^{(n)}$, corresponding to the node $v_n = 2N - n$, and updating the elements $\tilde{H}_{j\ell}$ for both j and ℓ adjacent to v_n as

$$\tilde{H}^{(n+1)}_{j\ell} = \tilde{H}^{(n)}_{j\ell} - A^{(n)}_{jv_n} A^{(n)}_{\ell v_n} \frac{\tilde{H}^{(n)}_{jv_n} \tilde{H}^{(n)}_{\ell v_n}}{\tilde{H}^{(n)}_{v_n v_n}} \quad (10)$$

for $1 \leqslant j,\ell \leqslant v_{n+1} = v_n - 1$. We remark that $\tilde{H}^{(n)}_{v_n v_n} \neq 0$, and therefore one can apply Eq. (10) for $n = 0,1,2,\ldots,N-1$. This can be understood as follows: If $\tilde{H}^{(n)}_{v_n v_n} = 0$ for $0 \leqslant n < N$, the $(n+1) \times (n+1)$ block matrix $\boldsymbol{B}$ representing the coupling among $v_0, v_1, \ldots, v_n$ should have its determinant equal to zero, since $\det \boldsymbol{B} \propto \prod_{\ell=0}^{n} \tilde{H}^{(\ell)}_{v_\ell v_\ell}$ as shown in 1, and the latter is zero under the assumption that $\tilde{H}^{(n)}_{v_n v_n} = 0$. This contradicts the condition that $\boldsymbol{H}$ is positive definite since all submatrices of a positive-definite matrix are also positive definite.

The adjacency matrix $\boldsymbol{A}^{(n+1)}$ is obtained by removing the last row and column of $\boldsymbol{A}^{(n)}$ and updating the element as

$$\boldsymbol{A}^{(n+1)}_{j\ell} = \boldsymbol{A}^{(n)}_{j\ell} + \boldsymbol{A}^{(n)}_{jv_n} \boldsymbol{A}^{(n)}_{\ell v_n} \left(1 - \boldsymbol{A}^{(n)}_{j\ell}\right). \quad (11)$$

Note that connecting each pair of the neighbor nodes of the decimated node may increase the mean degree $\langle k \rangle = 2L/N$, the ratio of the number of links (L) to the number of nodes (N), of the evolving graph.

$\boldsymbol{H}^{(N)}$ is uniquely determined regardless of the order of decimating nodes. However, the ordering v_n is important for reducing the computational cost. For instance, as shown in Fig. 1 and Eq. (11), if a node with degree k is removed, its k links are removed but its neighbors get interconnected, resulting in the maximum possible increase of links by $k(k-1)/2 - k$: If a hub node is eliminated, one should update many elements $\tilde{H}_{j\ell}$ according to Eq. (10) in the following stages of evolution, which increases the computing time.

The appearance of new links as in Fig. 1 and Eq. (11) are called *fill-ins* in the context of graph theory, and there have been many efforts to find the optimal ordering that suppresses those fill-ins. Eliminating nodes in a graph, the so-called *graph elimination game*, is encountered in the Cholesky factorization, which is generally used to solve the linear problem $\boldsymbol{M}\,|x\rangle = |b\rangle$ for a positive definite matrix $\boldsymbol{M}$. While the ideal ordering which minimizes the fill-ins is hard to find, heuristic methods have been proposed, such as the minimum-degree ordering, the reverse Cuthill-McKee ordering, and the nested-dissection ordering [23,30].

In decimating $\phi_j (= \psi_{j-N})$ variables in Eq. (5), every pair of nodes that are adjacent to the decimated node should update their corresponding matrix element. If we classify the pairs of nodes $(j\ell)$ into three groups according to the types of their associated variables as $(\phi_j\phi_\ell)$, $(J_j J_\ell)$, and $(\phi_j J_\ell)$, the computing time is expected to increase if many fill-ins appear for pairs of type (ϕ, J) or (ϕ,ϕ). Therefore, we here choose the minimum-degree ordering which minimizes the fill-ins for (ϕ_j,ϕ_ℓ). To find the order of decimating nodes and rearrange the node indices so that nodes are eliminated from the one with $j = 2N$ to $N+1$ in $\tilde{\boldsymbol{H}}$ after the rearrangement, we perform the node elimination in G representing $\boldsymbol{H}$ as follows:

(1) Construct graph $G^{(0)} = G$ representing $\boldsymbol{H}$.
(2) $n \leftarrow 0$.
(3) Choose one of the nodes having the minimum degree in $G^{(n)}$ and record its index in $w(n)$.
(4) Assign a link to every disconnected pair of neighboring nodes of the node $w(n)$ and eliminate the node $w(n)$ and its links, which yields $G^{(n+1)}$.
(5) If $n < N$, $n \leftarrow n+1$ and go to step 3. Otherwise, for each node of index $i = 1,2,\ldots,N$ of $\boldsymbol{H}$, assign a new index $N - w(i)$.

IV. TARGET PROBLEM

Our idea is that one can use the method in Sec. III to obtain a set of the arbitrary elements of the pseudoinverse of an $N \times N$ matrix $\boldsymbol{V}$ satisfying the following conditions:

(1) $\boldsymbol{V}$ is a semi-positive-definite symmetric matrix with the zero eigenvalue $\lambda_1 = 0$ of multiplicity 1, and
(2) $\boldsymbol{V}$ has the eigenvector $|\boldsymbol{e}^{(1)}\rangle$ corresponding to the zero eigenvalue, which does not have any zero component, i.e., $e^{(1)}_i \neq 0$ for $1 \leqslant i \leqslant N$.

We call such a matrix a semipositive definite symmetric (SPDS) matrix for simplicity. For a SPDS matrix $\boldsymbol{V}$, one can define its pseudoinverse matrix $\boldsymbol{V}^+$ by dropping the zero-eigenvalue mode as

$$\boldsymbol{V}^+ = \sum_{n=2}^{N} \frac{|\boldsymbol{e}^{(n)}\rangle\langle\boldsymbol{e}^{(n)}|}{\lambda_n}, \quad (12)$$

where λ_n are the eigenvalues of $\boldsymbol{V}$ with $\lambda_1 = 0$ and $|\boldsymbol{e}^{(n)}\rangle$ are the corresponding eigenvectors.

The generator $\boldsymbol{V}$ of a Markov chain satisfying the detailed-balance condition is an example [11]. $\boldsymbol{V}$ has the zero eigenvalue of multiplicity 1: Its left eigenvector is $\langle\boldsymbol{e}^{(1)}| = (1,1,\ldots,1)$, representing the conservation of the probability,

and the component $e_j^{(1)}$ of the right eigenvector $|\boldsymbol{e}^{(1)}\rangle = (e_1^{(1)}, e_2^{(0)}, \ldots)^\dagger$ represents the stationary-state probability of the state j. The detailed-balance condition requires that the transition from a state i to another j happens with equal probability to that of the transition from j to i. If $\boldsymbol{V}$ satisfies the detailed-balance condition, all the components $e_j^{(1)}$ should be nonzero. If $\boldsymbol{V}$ is not symmetric, a symmetric matrix $\bar{\boldsymbol{V}}$ can be obtained by the similarity transformation

$$\bar{\boldsymbol{V}} = \boldsymbol{S}\boldsymbol{V}\boldsymbol{S}^{-1} \tag{13}$$

with $S_{j\ell} = \delta_{j\ell}/\sqrt{e_j^{(1)}}$. The matrix $\bar{\boldsymbol{V}}$ is then a SPDS matrix. To obtain the stationary-state probability, there have been many efficient algorithms suggested so far; e.g., see Ref. [31].

As a concrete example of SPDS matrices, the Laplacian $\boldsymbol{L}$ with $L_{j\ell} = \delta_{j\ell} - A_{j\ell}/k_\ell$ generates the time evolution of the occupation probability $P_j(t)$ of a random walker as $P_j(t+1) = P_j(t) - \sum_\ell L_{j\ell} P_\ell(t)$. One can symmetrize $\boldsymbol{L}$ by the transformation $\bar{\boldsymbol{L}} = \boldsymbol{S}\boldsymbol{L}\boldsymbol{S}^{-1}$ with $S_{j\ell} = \delta_{j\ell} k_j^{-1/2}$ to obtain $\bar{L}_{j\ell} = \delta_{j\ell} - A_{j\ell}/\sqrt{k_j k_\ell}$. The pseudoinverse $\boldsymbol{L}^+$ or $\bar{\boldsymbol{L}}^+$ contains important information on random walk dynamics. For instance, the mean-first passage time (MFPT) $T_{\rm is}$ from a node s to i is represented as [32,33]

$$T_{\rm is} = \begin{cases} \frac{2L}{k_i}(L_{ii}^+ - L_{\rm is}^+) = \frac{2L}{k_i}\left(\bar{L}_{ii}^+ - \sqrt{\frac{k_i}{k_s}}\bar{L}_{\rm is}^+\right) & \text{for } i \neq s \\ \frac{2L}{k_i} & \text{for } i = s \end{cases}. \tag{14}$$

The GMFPT T_i of node i denotes the MFPT to the target node i averaged over all possible starting nodes in the stationary state [34] and is represented by the diagonal element of the pseudoinverse of the Laplacian as

$$T_i = \sum_s \frac{k_s}{2L} T_{\rm is} = \frac{2L}{k_i} L_{ii}^+ + 1 = \frac{2L}{k_i}\bar{L}_{ii}^+ + 1. \tag{15}$$

Another Laplacian $\hat{\boldsymbol{L}}$ with its element given by $\hat{L}_{j\ell} = k_j\delta_{j\ell} - A_{j\ell}$ is the time-evolution operator of the Edwards-Wilkinson model describing the fluctuating interfaces under tension and noise as $\dot{h}_j = -\sum_\ell \hat{L}_{j\ell} h_\ell + \xi_j(t)$ with h_j the height at site j and $\xi_j(t)$ the noise [35]. The height-height correlation is represented in terms of the pseudoinverse of $\hat{\boldsymbol{L}}$ as

$$\langle (h_j - \bar{h})(h_\ell - \bar{h})\rangle = \hat{L}_{j\ell}^+ \tag{16}$$

with the mean height $\bar{h} = N^{-1}\sum_j h_j$ [7,36,37]. The roughness is defined as $w = \sqrt{N^{-1}\sum_j \langle (h_j - \bar{h})\rangle^2}$ and is evaluated by

$$w = \sqrt{\frac{1}{N}\sum_j \hat{L}_{jj}^+}. \tag{17}$$

As shown above, the MFPT and the GMFPT of random walk and the height-height correlation and the roughness of fluctuating interfaces are commonly represented in terms of $O(N)$ number of elements of the pseudoinverse of an $N \times N$ SPDS matrix. The algorithm in the next section is appropriate for computing a set of such *arbitrary* elements of the pseudoinverse of a SPDS matrix. Other algorithms are optimal for N small [33], for the linear problems [18–21,28], for the positive-definite matrices [22–26], or for limited cases [27,29]. The linear problem $\boldsymbol{M}|x\rangle = |b\rangle$ is encountered in numerous applications and can give, for instance, the kth column of $\boldsymbol{M}^+$ by setting $b_j = \delta_{jk}$. However, the solution to such a single linear problem cannot give the sum of the diagonal elements $\sum_j M_{jj}^+$ as required in the GMFPT or the roughness. In contrast, our algorithm obtains a set of $O(N)$ *arbitrary* elements of the pseudoinverse of a large SPDS matrix at a time. In general, the inverse of a sparse matrix is not guaranteed to be sparse. Therefore given the limitation of space and time of computation, it is not always available to obtain *all* the elements of the inverse matrix of a large sparse matrix.

V. ALGORITHM FOR THE ARBITRARY ELEMENTS OF THE PSEUDOINVERSE OF A SPDS MATRIX

Here we present the algorithm for computing the arbitrary elements of the pseudoinverse of a SPDS matrix $\boldsymbol{V}$. Since $\boldsymbol{V}$ is not invertible, we introduce $\boldsymbol{H}(\mu) \equiv \mu\boldsymbol{I} + \boldsymbol{V}$ where μ is a positive real constant and $\boldsymbol{I}$ is the identity matrix of the same dimension as $\boldsymbol{V}$. Then $\boldsymbol{H}(\mu)$ is positive definite, and therefore we can apply Eqs. (1) and (2) to obtain

$$\begin{aligned} \boldsymbol{H}_{j\ell}^{-1}(\mu) &= -i\frac{\partial^2}{\partial J_j \partial J_\ell}\log Z(\mu)\Big|_{\vec{J}=0} \\ &= \sum_{n=1}^N \frac{e_j^{(n)} e_\ell^{(n)}}{\mu + \lambda_n} \\ &= \frac{e_j^{(1)} e_\ell^{(1)}}{\mu} + \sum_{n=2}^N \frac{e_j^{(n)} e_\ell^{(n)}}{\lambda_n} + O(\mu^1), \end{aligned} \tag{18}$$

where $\lambda_n (n = 1,2,\ldots,N)$ are the eigenvalues of $\boldsymbol{V}$ and $\mathbf{e}^{(n)}$ are the corresponding eigenvectors $\mathbf{e}^{(n)} = (e_1^{(n)}, e_2^{(n)}, \ldots, e_N^{(n)})^\dagger$. Therefore, the pseudoinverse $\boldsymbol{V}^+$ of $\boldsymbol{V}$ can be obtained by using $\boldsymbol{H}^{-1}$ of Eq. (18) as

$$V_{j\ell}^+ = \frac{\partial}{\partial\mu}\mu H_{j\ell}^{-1}(\mu)\Big|_{\mu=0}. \tag{19}$$

Equation (19) implies that one can obtain $\boldsymbol{V}^+$ once $\boldsymbol{H}^{-1}(\mu)$ is known as expanded in Eq. (18), which becomes available by the few first terms, up to $O(\mu^2)$, in the expansion of the extended matrix $\tilde{\boldsymbol{H}}$

$$\begin{aligned} \tilde{H}_{j\ell}(\mu) &= \left(1 + \mu\frac{\mathrm{d}}{\mathrm{d}\mu} + \frac{\mu^2}{2!}\frac{\mathrm{d}^2}{\mathrm{d}\mu^2}\right)\tilde{H}_{j\ell}\Big|_{\mu=0} + O(\mu^3) \\ &= \tilde{H}_{0,j\ell} + \tilde{H}_{1,j\ell}\mu + \tilde{H}_{2,j\ell}\mu^2 + O(\mu^3). \end{aligned} \tag{20}$$

Our idea is to apply the algorithm in Sec. III to trace the evolution of the three coefficient matrices $\tilde{\boldsymbol{H}}_0, \tilde{\boldsymbol{H}}_1$, and $\tilde{\boldsymbol{H}}_2$ in Eq. (20). Using 10 and Eq. (20), we eliminate a node with index $v_n = 2N - n$ and renormalize the coefficients at each

stage $0 \leqslant n < N-1$ as

$$
\begin{aligned}
\tilde{H}^{(n+1)}_{0,j\ell} &= \tilde{H}^{(n)}_{0,j\ell} - A^{(n)}_{jv_n} A^{(n)}_{\ell v_n} \frac{\tilde{H}^{(n)}_{0,jv_n} \tilde{H}^{(n)}_{0,\ell v_n}}{\tilde{H}^{(n)}_{0,v_n v_n}}, \\
\tilde{H}^{(n+1)}_{1,ij} &= \tilde{H}^{(n)}_{1,ij} - A^{(n)}_{jv_n} A^{(n)}_{\ell v_n} \frac{\tilde{H}^{(n)}_{1,jv_n} \tilde{H}^{(n)}_{0,\ell v_n} \tilde{H}^{(n)}_{0,v_n v_n} + \tilde{H}^{(n)}_{0,jv_n} \tilde{H}^{(n)}_{1,\ell v_n} \tilde{H}^{(n)}_{0,v_n v_n} - \tilde{H}^{(n)}_{0,jv_n} \tilde{H}^{(n)}_{0,\ell v_n} \tilde{H}^{(n)}_{1,v_n v_n}}{\left[\tilde{H}^{(n)}_{0,v_n v_n}\right]^2}, \\
\tilde{H}^{(n+1)}_{2,ij} &= \tilde{H}^{(n)}_{2,ij} - A^{(n)}_{jv_n} A^{(n)}_{\ell v_n} \frac{\tilde{H}^{(n)}_{2,jv_n} \tilde{H}^{(n)}_{0,\ell v_n} \left[\tilde{H}^{(n)}_{0,v_n v_n}\right]^2 + \tilde{H}^{(n)}_{1,jv_n} \tilde{H}^{(n)}_{1,\ell v_n} \left[\tilde{H}^{(n)}_{0,v_n v_n}\right]^2}{\left[\tilde{H}^{(n)}_{0,v_n v_n}\right]^3} \\
&\quad - A^{(n)}_{jv_n} A^{(n)}_{\ell v_n} \frac{\tilde{H}^{(n)}_{0,jv_n} \tilde{H}^{(n)}_{2,\ell v_n} \left[\tilde{H}^{(n)}_{0,v_n v_n}\right]^2 - \tilde{H}^{(n)}_{1,jv_n} \tilde{H}^{(n)}_{0,\ell v_n} \tilde{H}^{(n)}_{1,v_n v_n} \tilde{H}^{(n)}_{0,v_n v_n} - \tilde{H}^{(n)}_{0,jv_n} \tilde{H}^{(n)}_{1,\ell v_n} \tilde{H}^{(n)}_{1,v_n v_n} \tilde{H}^{(n)}_{0,v_n v_n}}{\left[\tilde{H}^{(n)}_{0,v_n v_n}\right]^3} \\
&\quad - A^{(n)}_{jv_n} A^{(n)}_{\ell v_n} \frac{-\tilde{H}^{(n)}_{0,jv_n} \tilde{H}^{(n)}_{0,\ell v_n} \tilde{H}^{(n)}_{2,v_n v_n} \tilde{H}^{(n)}_{0,v_n v_n} + \tilde{H}^{(n)}_{0,jv_n} \tilde{H}^{(n)}_{0,\ell v_n} \left[\tilde{H}^{(n)}_{1,v_n v_n}\right]^2}{\left[\tilde{H}^{(n)}_{0,v_n v_n}\right]^3},
\end{aligned}
\tag{21}
$$

where the indices j and ℓ run from 1 to $v_n - 1 = 2N - n - 1$.

Given the relation $\boldsymbol{H} = \mu \boldsymbol{I} + \boldsymbol{V}$ and $\boldsymbol{V}$ is a SPDS matrix, one eigenvalue of $\boldsymbol{H}$ can be zero if $\mu = 0$. During the renormalization of $\tilde{\boldsymbol{H}}_0, \tilde{\boldsymbol{H}}_1$, and $\tilde{\boldsymbol{H}}_2$ as in Eq. (21), $\tilde{H}^{(n)}_{0,v_n v_n}$ are nonzero for $n = 0,1,2,\ldots,N-2$. It is only in $\tilde{\boldsymbol{H}}^{(N-1)}$ that a singular element $\tilde{H}^{(N-1)}_{0,v_{N-1}v_{N-1}} = 0$ appears. This can be understood similarly to in III. Suppose that there exists n such that $\tilde{H}^{(n)}_{0,v_n v_n} = O(\mu^1)$ for $0 \leqslant n < N-1$. Then the determinant of the submatrix $\boldsymbol{B}$ representing the coupling among $v_0, v_1, \cdots, v_n$ should be $O(\mu^1)$, as $\det \boldsymbol{B} \propto \prod_{\ell=0}^{n} \tilde{H}^{(\ell)}_{v_\ell v_\ell}$ and $\tilde{\boldsymbol{H}}^{(n)}_{v_n v_n} = O(\mu^1)$ even if $\tilde{\boldsymbol{H}}^{(\ell)}_{v_\ell v_\ell} = O(1)$ for all $0 \leqslant \ell < n$. This means that the vector space spanned by $v_0, v_1, \ldots, v_n$ contains the eigenvector of $\tilde{\boldsymbol{H}}$ associated with the zero eigenvalue for $\mu = 0$, which contradicts the condition that the eigenvector of $\boldsymbol{V}$ and in turn that of $\tilde{\boldsymbol{H}}$ associated to the zero eigenvalue has no zero component. Therefore, $\tilde{H}^{(n)}_{0,v_n v_n} = 0$ should appear only for $n = N-1$.

At the last step of decimation, when the last node $v_{N-1} = N+1$ should be eliminated, its matrix element $\tilde{H}^{(N-1)}_{v_{N-1}v_{N-1}}$ becomes zero for $\mu = 0$, that is, $\tilde{H}^{(N-1)}_{v_{N-1}v_{N-1}}(\mu) = \tilde{H}^{(N-1)}_{1,v_{N-1}v_{N-1}}\mu + \tilde{H}^{(N-1)}_{2,v_{N-1}v_{N-1}}\mu^2 + O(\mu^3)$. Using this expansion in Eq. (10), one can find that $\tilde{H}^{(N)}_{j\ell}$ is expanded as

$$
H^{-1}_{j\ell}(\mu) = \tilde{H}^{(N)}_{j\ell}(\mu) = \tilde{H}^{(N)}_{-1,j\ell} \frac{1}{\mu} + \tilde{H}^{(N)}_{0,j\ell} + O(\mu) \tag{22}
$$

with the coefficient matrices $\tilde{H}^{(N)}_{-1,j\ell}$ and $\tilde{H}^{(N)}_{0,j\ell}$ evaluated in terms of $\tilde{\boldsymbol{H}}^{(N-1)}$ as

$$
\begin{aligned}
\tilde{H}^{(N)}_{-1,j\ell} &= -A^{(N-1)}_{jv_{N-1}} A^{(N-1)}_{\ell v_{N-1}} \frac{\tilde{H}^{(N-1)}_{0,jv_{N-1}} \tilde{H}^{(N-1)}_{0,\ell v_{N-1}}}{\tilde{H}^{(N-1)}_{1,v_{N-1}v_{N-1}}}, \\
\tilde{H}^{(N)}_{0,j\ell} &= \tilde{H}^{(N-1)}_{0,j\ell} - A^{(N-1)}_{jv_{N-1}} A^{(N-1)}_{\ell v_{N-1}} \left\{ \frac{\tilde{H}^{(N-1)}_{1,jv_{N-1}} \tilde{H}^{(N-1)}_{0,\ell v_{N-1}} + \tilde{H}^{(N-1)}_{0,jv_{N-1}} \tilde{H}^{(N-1)}_{1,\ell v_{N-1}}}{\tilde{H}^{(N-1)}_{1,v_{N-1}v_{N-1}}} - \frac{\tilde{H}^{(N-1)}_{0,jv_{N-1}} \tilde{H}^{(N-1)}_{0,\ell v_{N-1}} \tilde{H}^{(N-1)}_{2,v_{N-1}v_{N-1}}}{\left[\tilde{H}^{(N-1)}_{1,v_{N-1}v_{N-1}}\right]^2} \right\}.
\end{aligned}
\tag{23}
$$

Then, from Eq. (19), the elements of the pseudoinverse of $\boldsymbol{V}$ are evaluated as

$$
V^{+}_{j\ell} = \tilde{H}^{(N)}_{0,j\ell}. \tag{24}
$$

The above procedures for computing the exact pseudoinverse $\boldsymbol{V}^+$ of an $N \times N$ SPDS matrix $\mathbf{V}$ are summarized in the following:

(1) Construct a $N \times N$ matrix $\boldsymbol{H}^{(0)} = \mu \boldsymbol{I} + \boldsymbol{V}$ and its extended matrix $\tilde{\boldsymbol{H}}$ of $2N \times 2N$ using Eq. (4).

(2) Construct a graph $G^{(0)}$ representing $\tilde{\boldsymbol{H}}$ and make its adjacency matrix $\boldsymbol{A}^{(0)}$.

(3) $n \leftarrow 0$.

(4) Remove the last row and column of $\tilde{\boldsymbol{H}}^{(n)}$ and update the elements related to the neighbor nodes of the node v_n using Eq. (21) if $n < N-1$ or Eq. (23) for $n = N-1$. This yields $\tilde{\boldsymbol{H}}^{(n+1)}$.

(5) Assign a link between every disconnected pair of the neighbor nodes of v_n and eliminate the node v_n and its links in $G^{(n)}$. This yields $G^{(n+1)}$. Remove the last row and column of $\boldsymbol{A}^{(n)}$ and update the elements related to the neighbor nodes of v_n by using Eq. (11). This yields $\boldsymbol{A}^{(n+1)}$.

(6) If $n = N-1$, stop the process, else $n \leftarrow n+1$ and go to step 4.

Here we emphasize that before applying this algorithm, the indices of the matrix $\boldsymbol{H}$ should be rearranged such that the ordered list of decimated nodes in $\tilde{\boldsymbol{H}}$ is given by $v_n = 2N - n$. The source code implementing the proposed algorithm is available in Ref. [38].

The space and time complexities of the algorithm are as follows. If the number of neighbors of the decimated nodes is $O(1)$, each step in the algorithm takes $O(1)$ time and the whole algorithm will take $O(N)$ time as steps 4 and 5 are repeated N times. This implies that the number of nonzero elements of $\boldsymbol{H}$ is $O(N)$ throughout the computation and $O(N)$ space of memory is sufficient. On the other hand, if the number of neighbors of the decimated node is of order N and thereby $O(N^2)$ elements should be updated at steps 4 and 5, the computation time scales as $\sim N^3$. Also, $\boldsymbol{H}$ becomes dense and $O(N^2)$ memory is needed. Therefore the computational cost of our algorithm depends critically on the network topology such that the space and the time complexity are $O(N)$ and $O(N)$ in the best case and $O(N^2)$ and $O(N^3)$ in the weakest case, respectively. It depends also on the order of decimating nodes while we do not explore this issue systematically here. In the next section, we investigate the performance of our algorithm in more detail, focusing on time complexity.

VI. PERFORMANCE OF OUR ALGORITHM IN COMPUTING THE GMFPT

The major use of our algorithm lies in its ability to compute a set of arbitrary elements of the exact pseudoinverse of a large SPDS matrix, which are important in the physics context at least as described in Sec. IV. Furthermore, its computing time is much shorter than $O(N^3)$ for most matrices, as we will show in this section, which enables us to apply the algorithm to large matrices constructed from big data.

To address the performance of the proposed algorithm specifically, we investigate the computing time $\mathcal{T}$ taken to obtain *all* the diagonal elements of the symmetric Laplacian matrix $\bar{\boldsymbol{L}}$ of diverse networks including artificial and real ones. As shown in Eq. (15), the set of all the diagonal components $\{\bar{L}^+_{jj}|j = 1,2,\ldots,N\}$ indicates the GMFPTs to all nodes, T_j, in a network having the symmetric Laplacian matrix $\bar{\boldsymbol{L}}$.

A. GMFPT from the simulation of random walk

For comparison, let us consider estimating the GMFPT T_j by performing the simulation of random walk on a given sparse network of N nodes and $L = O(N)$ links. The average of the MFPT for m random walkers starting at arbitrary locations and arriving at node j gives the *ensemble average* $\langle T_j\rangle$. The advantage of the random walk simulation is that the required memory is only $O(N)$, much smaller than $O(N^2)$ in the worst case of our algorithm. Concerning the time complexity, it takes $\mathcal{T} \simeq \langle T_j\rangle Nm$ to obtain all the GMFPT's $\{\langle T_j\rangle|j = 1,2,\ldots,N\}$ from the simulation. The deviation of $\langle T_j\rangle$ from the exact value T_j scales as $T_j - \langle T_j\rangle \sim \langle T_j\rangle/\sqrt{m}$ [34], and thus the higher accuracy we require, the larger number of ensembles (random walkers) we need to run. Note that the algorithm proposed in this work provides the exact values T_j, and their higher moments can be also obtained exactly [16]. We simply require that the relative error $\frac{T_j-\langle T_j\rangle}{\langle T_j\rangle}$ should be statistically less than $\frac{1}{\sqrt{\langle T_j\rangle}}$, which leads to the requirement that the number of ensemble should be larger than $\langle T_j\rangle$, i.e., $m \geqslant \langle T_j\rangle$. The total simulation time is then given by

$$\mathcal{T} \sim \max_j\{\langle T_j\rangle^2\}N. \tag{25}$$

It is known that

$$\max_j\{\langle T_j\rangle\} \sim \begin{cases} N^{2/d_s} & \text{for} \quad d_s < 2 \\ N & \text{for} \quad d_s > 2 \end{cases} \tag{26}$$

with d_s the spectral dimension of the underlying network [34,39,40]. Therefore the whole simulation time needed to obtain such accurate ensemble averages $\langle T_j\rangle$ for all $j = 1,2,\ldots,N$ as the relative error being less than $1/\sqrt{\langle T_j\rangle}$ scales as

$$\mathcal{T} \sim N^z, \tag{27}$$

$$z = \begin{cases} \frac{4}{d_s+1} & \text{for} \quad d_s < 2 \\ 3 & \text{for} \quad d_s > 2 \end{cases}. \tag{28}$$

It is remarkable that the simulation time decreases with the spectral dimension; A very long simulation is needed for estimating T_j in networks of low dimensionality. For instance, $\mathcal{T} \sim N^5$ for $d_s = 1$ and $N^3 < \mathcal{T} < N^5$ for $1 < d_s < 2$. Such long simulations are not available practically for large N. Our algorithm gives the exact values of $\{T_j\}$ within $O(N^3)$ time even in the worst case. Moreover, in contrast to the simulation time in Eq. (28), the computing time $\mathcal{T}$ of the algorithm turns out to be short for networks of low dimensionality.

B. Computing time for GMFPT in model networks

The performance of our algorithm varies with the network topology. In Fig. 2 we present the computing time $\mathcal{T}$ of the GMFPTs as a function of the number of nodes N for various model networks such as the Sierpinski gasket ($d_f = \ln 3/\ln 2$ and $d_s = 2\ln 3/\ln 5$) [41], two-dimensional (2D) percolation clusters at the critical point($d_f = 91/48$ and $d_s = 1.32$) [42]), the Barabási-Albert (BA) model networks with a power-law degree distributions [43] ($d_f \to \infty, d_s = 4/3$ for $\langle k\rangle = 2$ and $d_f \to \infty, d_s \to \infty$ for $\langle k\rangle > 2$) [37,44,45] and the (1,2)-flower networks ($d_f = \infty, d_s = 2\ln 3/\ln 2$) [46–48], where $d_f(d_s)$ is the fractal (spectral) dimension. If we measure the scaling exponent z introduced in Eq. (27) also for the computing time $\mathcal{T}$ of our algorithm in each network, the exponent z turns out to be different as shown in Fig. 2.

We can classify those studied networks according to their fractal dimensions or the spectral dimensions. The Sierpinski gasket and the 2D critical percolation cluster have finite fractal dimensions, and other networks are not fractal, having infinite fractal dimensions. The spectral dimension is infinite in the BA model networks with $\langle k\rangle = 4$; however, it is finite between 1 and 2, for other networks.

The Sierpinski gasket and the 2D critical percolation cluster have their node degree bounded. Given such finite node degrees, the computing time of steps 4 and 5 at each iteration in the algorithm in Sec. V is expected to be $O(1)$ unless many fill-ins are generated during renormalization. The scaling exponent z in Eq. (27) is indeed $z = 1.7$ and $z = 1.3$ for the Sierpinski gasket and the 2D percolation cluster, respectively, in Fig. 2(a). Both are far smaller than $z = 3$ of the worst

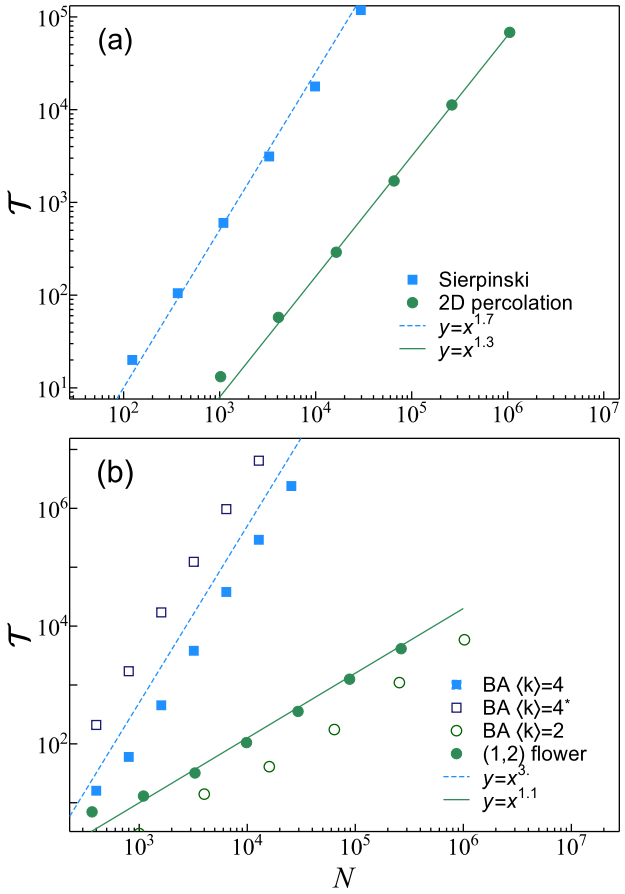

FIG. 2. (Color online) Scaling of the computing time $\mathcal{T}$ for the generalized MFPT in model networks of N nodes. The model networks are (a) the Sierpinski gasket and 2D critical percolation cluster and (b) the BA model network with the mean degree $\langle k\rangle = 2$ and $\langle k\rangle = 4$ and (1,2)-flower networks. For comparison, we also draw the result for the BA model (□) with the same mean degree $\langle k\rangle = 4$ but using the conventional eigendecomposition. We have not shown the scaling of computing times obtained by using the conventional method for other cases but the BA model with $\langle k\rangle = 4$ because they all behave as $\mathcal{T} \sim O(N^3)$.

case. This suggests that it affects the time complexity of our algorithm whether the degree is bounded or not. The Sierpinski gasket is constructed recursively and shows the self-similarity of fractal structures. The minimum-degree node is the oldest one in the Sierpinski gasket, which generates fill-ins. The structure of the 2D percolation cluster is not deterministic but random due to the removal of randomly selected sites during the course of its construction from a regular 2D lattice. The shorter computing time $\mathcal{T}$ in the percolation cluster implies that a smaller number of fill-ins are generated than in the Sierpinski gasket by the minimum-degree ordering.

In Fig. 2(b), we present the computing time of the GMFPT in two scale-free (SF) networks: the BA model networks with $\langle k\rangle = 2$ and 4 and the (1,2)-flower networks. They are not fractal. The node degrees are not bounded and therefore the computing time of steps 4 and 5 at each iteration can be long. The scaling exponent z of the computing time is expected to be larger than the networks with bounded degrees. However, $\mathcal{T}$ is the shortest for the (1,2)-flower networks among the four classes of networks in Fig. 2. On the contrary, the BA model networks with $\langle k\rangle = 2L/N = 4$ show the longest computing time. The origin of such a striking difference can be found in their network structures. The flower networks are constructed in a recursive way with the youngest node having the minimum degree. Eliminating the minimum-degree nodes is thus exactly the reverse of the original construction process and does not create any fill-in. Furthermore, every node has only two neighbors at the moment of elimination, which leads to the almost linear scaling ($z = 1$) of the computing time as shown in Fig. 2(b). On the other hand, the BA model networks are random networks displaying power-law degree distributions, for which many fill-ins can be created during renormalization. These BA networks with $\langle k\rangle = 4$ become almost completely connected already in the early stage of evolution, and thus steps 4 and 5 take $O(N^2)$ time at each iteration, leading to $z = 3$, the largest value of z possible in our algorithm. It should be also noted that the BA networks with $\langle k\rangle = 2$ have the computing time scale in a similar way to that of the (1,2)-flower networks, much shorter than that of the BA networks with $\langle k\rangle = 4$. Their difference is that a BA network with $\langle k\rangle = 2$ is of a tree structure and has a finite spectral dimension ($d_s = 4/3$) in contrast to the BA networks with $\langle k\rangle = 4$ that have loops and $d_s \to \infty$.

In spite of such varying behaviors of the computing time from network to network, the performance of our algorithm in computing the GMFPT is better than that of the the random-walk simulation in all studied networks. Interestingly, in contrast to the simulation time, the computing time of the algorithm tends to be shorter in networks of low dimensionality than those of high dimensionality, characterized by d_f and d_s, meaning that the algorithm is particularly useful for the networks of low dimensionality. We also observe that the structural characteristics other than dimensionality, such as hierarchy and randomness, and the ordering scheme for eliminating nodes may affect the computing time and even the scaling exponent z. It has been shown that there exists an ordering which provides the upper bound of the number of fill-ins less than $O[N^{1/4}(\log N)^{7/2}]$ and therefore $\mathcal{T} \sim N^{5/4}(\ln N)^{7/2}$ for a given sparse matrix [30]. Therefore the computing time can be reduced drastically if the optimal ordering can be found and applied. Various ordering schemes other than the minimum-degree one can be found in, e.g., Ref. [49].

The conventional eigendecomposition method based on the QR algorithm [17] can be applied to obtain the GMFPT if the size of the Laplacian matrix is not so large. The conventional method takes $O(N^3)$ time, whether the matrix is sparse or not [17]. Its computing time for the BA model networks with $\langle k\rangle = 4$ is presented for $N \lesssim 10^4$ in Fig. 2(b). While our algorithm shows the worst performance, $O(N^3)$, for the BA networks with $\langle k\rangle = 4$ in Fig. 2, it is shown to be better than the conventional method with the ratio of the computing times of the two algorithms $\frac{\mathcal{T}_{\text{ours}}}{\mathcal{T}_{\text{conv.}}} \simeq 0.03 \pm 0.015$ almost constant in our simulation range $800 \leqslant N \leqslant 12\,800$. It is obvious that our algorithm outperforms the conventional method for other networks, for which our algorithm shows $O(N^z)$ time complexity with $z < 3$ but the conventional one shows $O(N^3)$ one. Given that the computing time of the conventional method is 10^7 ms (2.7 h) for the BA networks with $\langle k\rangle = 4$ and

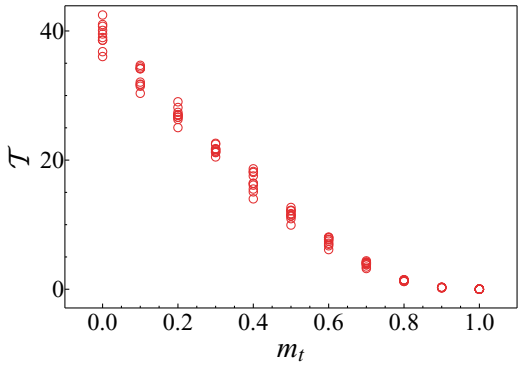

FIG. 3. (Color online) The computing time $\mathcal{T}$ (in seconds) for the GMFPT in the modified BA networks of $N = 10^4$ and $\langle k \rangle = 4$ with the clustering coefficient controlled by m_t. The larger m_t is, the larger the clustering coefficient is. For each given value of m_t, 10 networks are sampled and their computing times are plotted.

$N = 10^4$, one can see that it amounts to 2700 h $\approx$ 115 d for $N = 10^5$, and thus the conventional method does not work for the BA networks with $N = 10^5$.

We should mention that all the computations, whatever algorithms we use, and all the simulations have been performed on the same identical computer equipped with an Intel i7, 3.4 Ghz CPU, and 8 GB memory. In compiling the source code C++, we switched on the gcc's compiler options "-O3 -ffast-math" for optimization. In particular, for the computation by the conventional eigendecomposition method, we used the implementation of the Eigen library, which is believed to be one of the most efficient linear algebra library [50].

Finally, we also investigate the dependence of network clustering on the computing time of our algorithm. The clustering coefficient of a network [51] quantifies the likelihood that two neighbors of a node are also connected to each other. The number of fill-ins is therefore expected to be smaller for a network with high clustering than that with low clustering if both have the same number of nodes and links in the beginning. For a variant of the BA model with a parameter m_t controlling the clustering coefficient [52], the computing time of the GMFPT is indeed decreasing with increasing the clustering coefficient (m_t) in the model network of $N = 10^4$ nodes and $\langle k \rangle = 4$ as shown in Fig. 3.

C. Computing time for GMFPT in real networks

The scaling behaviors of the computing time, $\mathcal{T} \sim O(N^z)$ with $z < 3$, identified in most of the studied artificial networks, suggest that our algorithm can be useful in analyzing the Laplacian matrices of large real-world systems. We constructed the Laplacian matrices $\bar{\boldsymbol{L}}$ of one e-mail communication network, the subgraphs of the World Wide Web (WWW), and two road networks in the United States, all archived in the Stanford Large Network Dataset Collection [1]. These selected networks commonly have a very large number of nodes, N ranging between 2×10^5 and 2×10^6 and the mean degree $\langle k \rangle$ between 2 and 16. The properties of those real-world networks and the computing time of the GMFPTs $\{T_j\}$ by our algorithm are shown in Table I. Most importantly, we found that our algorithm can obtain all the GMFPTs in 5 min for an e-mail network and road networks and in 1 or 2 h for the WWW. Such fast computation of the pseudoinverse of matrices of size millions by millions strongly suggests that our algorithm can be applied to the analysis of diverse big-data systems demanded increasingly in this era.[1]

TABLE I. The number of nodes (N), the number of links (L), the mean degree $\langle k \rangle = 2L/N$, the clustering coefficient (C.C.), the trace of the symmetric Laplacian matrix $\mathrm{Tr}\,\bar{\boldsymbol{L}}^+/N$, and the computing time ($\mathcal{T}$) for the GMFPT are given for each network.

Network	N	L	$\langle k \rangle$	C.C.	$\mathrm{Tr}\bar{\boldsymbol{L}}^+/N$	$\mathcal{T}$(sec)
Email-EuAll	224 832	339 925	3.02	0.07	21.3529	87.5
web-Stanford	255 265	1 941 926	15.2	0.60	18.7769	2833
web-NotreDame	325 729	1 090 108	6.69	0.23	39.5499	6608
roadNet-CA	1 957 027	2 760 388	2.82	0.05	916.898	351
roadNet-TX	1 351 137	1 879 201	2.78	0.05	862.147	165

Also, it is interesting that the computing times $\mathcal{T}$ are scattered seemingly regardless of the size N; the computing time is shorter for road networks of more than one million nodes than for the WWW consisting of less than a half million nodes. This is not explained by their clustering coefficients, which would predict the longer computing time for the networks of low clustering as in Fig. 3. The trace of the pseudoinverse $\mathrm{Tr}\,\bar{\boldsymbol{L}}^+/N$ is related to the GMFPT by Eq. (15) and is given in Table I. The road networks show larger values of $\mathrm{Tr}\,\bar{\boldsymbol{L}}^+/N$ than the WWW. From Eq. (26), we can conjecture that the spectral dimensions d_s of the road networks are smaller than those of the WWW and suspect that the smaller values of d_s may be related to such a short computing time in the road networks. We have already seen that the computing time is short in the model networks of low dimensionality.

VII. SUMMARY AND DISCUSSION

In this work we proposed an algorithm that computes a set of arbitrary elements of the exact pseudoinverse of a class of singular matrices, which we call the SPDS matrices. This class of matrices plays the role of the time-evolution operators in the Markov chains satisfying the detailed-balance condition, and the elements of their pseudoinverse contain important information such as the MFPT and the correlation function. Therefore fast and efficient algorithms for computing the elements of the pseudoinverse of the SPDS matrices can be greatly useful for analyzing the dynamics of large complex systems in this big-data era. Our algorithm consists of the steps of decimating the variables in the Gaussian integral and renormalizing the Hamiltonian matrix repeatedly. The algorithm runs very fast, occupying little memory space in many cases, which enables us to apply the algorithm to large

[1]We also tried but failed to obtain the GMFPTs in the collaboration network "com-DBLP" of 334 863 nodes and $\langle k \rangle = 5.53$, owing to insufficient memory for the increasing number of nonzero elements during renormalization. As our algorithm works for larger networks, we expect that the optimal ordering for this network, other than the minimum-degree ordering, should enable the computation.

singular matrices, e.g., of size millions by millions, capturing the dynamics of large complex systems.

The optimal order of decimating nodes, once found, would greatly reduce the computing time of our algorithm, which needs further investigation for practical applications. We have shown that our algorithm allows us to obtain the diagonal elements of the pseudoinverse of the Laplacian matrices of real-world networks such as the WWW, e-mail communication, and road networks of millions of nodes within minutes or a few hours, which suggests strongly the potential of our algorithm in analyzing the relaxation processes in big-data systems.

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NFR) grants funded by the Korean Government (MSIP and MEST) [No. 2010-0015066 (B.K.) and No. 2012R1A1A2005252 (D.-S.L.)].

APPENDIX: A FASTER ALGORITHM COMPUTING TrV^+

For computing quantities like the roughness w defined in Eq. (17), only Tr $\boldsymbol{V}^+$ is needed. In such a case, the auxiliary variables $|J\rangle$ are not needed nor is the extended matrix $\tilde{\boldsymbol{H}}$, which greatly reduces the running time of the algorithm.

Let us consider a SPDS matrix $\boldsymbol{V}$ and the coupling matrix $\boldsymbol{H}(\mu) = \mu\boldsymbol{I} + \boldsymbol{V}$. Since $\mathrm{Tr}\,\boldsymbol{V}^+ \equiv \sum_{\ell=2}^{N} \frac{1}{\lambda_\ell}$ with λ_ℓ being the eigenvalues of $\boldsymbol{V}$, one can use the expansion of det $\boldsymbol{H}$ as

$$\det \boldsymbol{H} = \prod_{n=1}^{N}(\mu + \lambda_n) = a_N\mu^N + a_{N-1}\mu^{N-1} + \cdots + a_2\mu^2 + a_1\mu \quad \text{(A1)}$$

with

$$\mathrm{Tr}\,\boldsymbol{V}^+ = \frac{a_2}{a_1}. \quad \text{(A2)}$$

Considering the application of the procedures in Sec. V to $\boldsymbol{H}$, not to $\tilde{\boldsymbol{H}}$, one finds that

$$\det \boldsymbol{H} = \prod_{n=0}^{N-1} H^{(n)}_{v_n v_n}. \quad \text{(A3)}$$

Using the expansion of $\boldsymbol{H}^{(n)}_{v_n v_n}$ in terms of μ as

$$\boldsymbol{H}^{(n)} = \begin{cases} \boldsymbol{H}^{(n)}_0 + \boldsymbol{H}^{(n)}_1\mu + O(\mu^2) & (0 \leqslant n < N-1) \\ \boldsymbol{H}^{(N-1)}_1\mu + \boldsymbol{H}^{(N-1)}_2\mu^2 + O(\mu^3) & (n = N-1) \end{cases} \quad \text{(A4)}$$

one can obtain a_1 and a_2 in Eq. (A1). Finally, Tr $\boldsymbol{V}^+$ is evaluated as

$$\mathrm{Tr}\,\boldsymbol{V}^+ = \sum_{n=0}^{N-2} \frac{H^{(n)}_{1,v_n v_n}}{H^{(n)}_{0,v_n v_n}} + \frac{H^{(n)}_{2,v_{N-1}v_{N-1}}}{H^{(n)}_{1,v_{N-1}v_{N-1}}}. \quad \text{(A5)}$$

[1] SNAP, http://snap.stanford.edu.
[2] J. D. Noh and H. Rieger, Phys. Rev. Lett. **92**, 118701 (2004).
[3] Y. Meroz, I. M. Sokolov, and J. Klafter, Phys. Rev. E **83**, 020104 (2011).
[4] A. Langville, *Google's PageRank and Beyond: The Science of Search Engine Rankings* (Princeton University Press, Princeton, NJ, 2006).
[5] M. Barahona and L. M. Pecora, Phys. Rev. Lett. **89**, 054101 (2002).
[6] Y. Wang, D. Chakrabarti, C. Wang, and C. Faloutsos, in *Proceedings of the 22nd International Symposium on Reliable Distributed Systems* (IEEE, Florence, 2003), pp. 25–34.
[7] G. Korniss, Z. Toroczkai, M. A. Novotny, and P. A. Rikvold, Phys. Rev. Lett. **84**, 1351 (2000).
[8] N. Biggs, *Algebraic Graph Theory* (Cambridge University Press, Cambridge, 1994).
[9] F. R. K. Chung, *Spectral Graph Theory* (American Mathematical Society, Providence, RI, 1996).
[10] D. Bonchev, A. T. Balaban, X. Liu, and D. J. Klein, Int. J. Quantum Chem. **50**, 1 (1994).
[11] N. V. Kampen, *Stochastic Processes in Physics and Chemistry*, vol. 1, 2nd ed., North-Holland Personal Library (North Holland, Amsterdam Boston London, 1992).
[12] C. Meyer, Jr., SIAM Rev. **17**, 443 (1975).
[13] W. Rising, Adv. Appl. Prob. **23**, 293 (1991).
[14] A. Ben-Israel and T. N. Greville, *Generalized Inverses: Theory and Applications* (Springer, New York, 2010).
[15] J. G. Kemeny and J. L. Snell, *Finite Markov Chains* (Springer, New York, 1983).
[16] J. J. Hunter, Linear Algebra Appl. **447**, 38 (2014).
[17] W. Press, *Numerical Recipes: The Art of Scientific Computing* (Cambridge University Press, Cambridge, 2007).
[18] Y. Saad, *Iterative Methods for Sparse Linear Systems*, 2nd ed. (Society for Industrial and Applied Mathematics, Philadelphia, 2003).
[19] G. G. Batrouni, A. Hansen, and M. Nelkin, Phys. Rev. Lett. **57**, 1336 (1986).
[20] G. G. Batrouni and B. Svetitsky, Phys. Rev. B **36**, 5647 (1987).
[21] G. G. Batrouni and A. Hansen, J. Stat. Phys. **52**, 747 (1988).
[22] A. George, SIAM J. Numer. Anal. **10**, 345 (1973).
[23] A. George and J. W. Liu, *Computer Solutions of Large Sparse Positive Definite Systems* (Prentice-Hall, New York, 1981).
[24] S. Li, S. Ahmed, G. Klimeck, and E. Darve, J. Comp. Phys. **227**, 9408 (2008).
[25] L. Lin, C. Yang, J. C. Meza, J. Lu, L. Ying, and W. E, ACM Trans. Math. Softw. **37**, 40 (2011).
[26] S. Eastwood and J. W. L. Wan, Numer. Linear Algebra Appl. **20**, 74 (2013).
[27] N.-D. Ho and P. V. Dooren, Appl. Math. Lett. **18**, 917 (2005).
[28] N. K. Vishnoi, *LX = B: Laplacian Solvers and Their Algorithmic Applications* (Now Publishers, Boston, 2013).
[29] G. Ranjan, Z.-L. Zhang, and D. Boley, arXiv:1304.2300 (2013).
[30] A. George, J. R. Gilbert, and J. W. H. Liu, *Graph Theory and Sparse Matrix Computation* (Springer, New York, 2011).
[31] M. Benzi and M. Tůma, Appl. Numer. Math **41**, 135 (2002).
[32] F. Chung and S.-T. Yau, J. Combin. Theor. A **91**, 191 (2000).
[33] H. Qiu and E. Hancock, IEEE Trans. Pattern Anal. Machine Intel. **29**, 1873 (2007).

[34] S. Hwang, D.-S. Lee, and B. Kahng, Phys. Rev. Lett. **109**, 088701 (2012).
[35] A.-L. Barabási and H. E. Stanley, *Fractal Concepts in Surface Growth* (Cambridge University Press, Cambridge, 1995).
[36] B. Kozma, M. B. Hastings, and G. Korniss, Phys. Rev. Lett. **92**, 108701 (2004).
[37] H. Guclu, G. Korniss, and Z. Toroczkai, Chaos **17**, 026104 (2007).
[38] S. Hwang, https://github.com/sungmin817/pseudoinverseDBCMarkovChain (2014).
[39] V. Tejedor, O. Bénichou, and R. Voituriez, Phys. Rev. E **80**, 065104 (2009).
[40] B. Meyer, C. Chevalier, R. Voituriez, and O. Bénichou, Phys. Rev. E **83**, 051116 (2011).
[41] D. ben Avraham and S. Havlin, *Diffusion and Reactions in Fractals and Disordered Systems* (Cambridge University Press, Cambridge, 2005).
[42] R. Rammal, J. C. Angles d'Auriac, and A. Benoit, Phys. Rev. B **30**, 4087 (1984).
[43] A.-L. Barabási and R. Albert, Science **286**, 509 (1999).
[44] B. Durhuus, T. Jonsson, and J. Wheater, J. Stat. Phys. **128**, 1237 (2007).
[45] A. N. Samukhin, S. N. Dorogovtsev, and J. F. F. Mendes, Phys. Rev. E **77**, 036115 (2008).
[46] H. D. Rozenfeld, S. Havlin, and D. ben Avraham, New J. Phys. **9**, 175 (2007).
[47] H. D. Rozenfeld and D. ben-Avraham, Phys. Rev. E **75**, 061102 (2007).
[48] M. Hinczewski and A. N. Berker, Phys. Rev. E **73**, 066126 (2006).
[49] G. Karypis and V. Kumar, MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, version 4.0, http://www.cs.umn.edu/∼metis.
[50] G. Guennebaud *et al.*, Eigen, version 3, http://eigen.tuxfamily.org.
[51] D. J. Watts and S. H. Strogatz, Nature (London) **393**, 440 (1998).
[52] P. Holme and B. J. Kim, Phys. Rev. E **65**, 026107 (2002).