Fuzzy Systems and Data Mining VII
A.J. Tallón-Ballesteros (Ed.)

doi:10.3233/FAIA210212

Multi-Step Low-Rank Decomposition of Large PageRank Matrices

Zhao-Li SHEN [a] Bruno CARPENTIERI [b,1]
[a] *College of Science, Sichuan Agricultural University, Ya'an, Sichuan 625000, P.R. China*
[b] *Faculty of Computer Science, Free University of Bozen-Bolzano, 39100 Bolzano, Italy*

Abstract. The PageRank model, initially proposed by Google for search engine rankings, provides a useful network centrality measure to identify the most important nodes within large graphs arising in several applications. However, its computation is often very difficult due to the huge sizes of the networks and the unfavourable spectral properties of the associated matrices. We present a novel multi-step low-rank factorization that can be used to reduce the huge memory cost demanded for realistic PageRank calculations. Finally, we present some directions of future research.

Keywords. PageRank model, network centrality computation, matrix pre-processing algorithms, low-rank factorization.

1. Introduction

The PageRank model proposed by Google in a series of papers [1,2] can quantify the importance of each Web page efficiently from the linking structure of the World Wide Web. In this model, the linking structure is represented by a binary matrix $G \in \mathbb{R}^{n\times n}$ (where n denotes the number of hyperlinked pages) such that $G(i,j)$ is nonzero (being 1) only when there is a hyperlink pointing from the j^{th} to the i^{th} page. Then the stationary distribution of a random walk on this structure represents the importance of each page. An important assumption of the model is the equal probability of each hyperlink to be selected on a given page. Accordingly, the transition matrix P of this random process is defined as

$$P(i,j) = \begin{cases} \frac{1}{\sum_{k=1}^{n} G(k,j)}, & \text{if } G(i,j) = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

To ensure that the process will not stagnate when the surfer visits a node with 0 out-degree, matrix P is usually modified as $\bar{P} = P + vd^T$, where $d \in N^{n\times 1}$ is a binary vector such that $d(i) = 1$ when the i^{th} node has 0 out-degree. Meanwhile the irreducibility of

[1]Corresponding Author: Bruno Carpentieri; E-mail: bcarpentieri@gmail.com. This author is a member of the *Gruppo Nazionale per il Calcolo Scientifico* (GNCS) of the Istituto Nazionale di Alta Matematica (INdAM) and this work was partially supported by INdAM-GNCS under Progetti di Ricerca 2020.

the Markov chain is needed for a unique solution, hence $\bar{P}$ is often modified into a matrix $\widetilde{P}$ defined as follows

$$\widetilde{P} = \alpha\bar{P} + (1-\alpha)ve^T, \tag{2}$$

where $\alpha \in (0,1)$ is called the damping factor, $v \in R^{n\times 1}$ is a probability distribution vector, and $e = [1,1,\cdots,1]^T$. Finally, the PageRank model can be stated as finding the stationary distribution vector x such that

$$\widetilde{P}x = x, \quad \|x\|_1 = 1, \quad x > 0. \tag{3}$$

The eigen-problem (3) is mathematically equivalent to the solution of the nonsingular linear system [3]

$$Ax = v, \quad A = (I - \alpha P), \quad x > 0. \tag{4}$$

The PageRank model depends solely on the graph topology of G and on the user-defined damping factor α and distribution vector v. Therefore, it represents a general network centrality measure that can be used to analyze large graphs arising in many computational science and engineering applications [4]. The assumption that the relevance of a node is determined by the number and by the importance of the nodes pointing directly to it is appropriate for many classes of networks arising in applications. Indeed, the model is broadly used far beyond search engine rankings, e.g., in feature selection of intelligent systems [5], in large micro-array experiments of computational biology such as the GeneRank [6] and the ProteinRank [7] problems, in ranking authors from co-citation networks [8], to name only a few examples. Recently, the model has been generalized to analyze multiplex networks as well [9].

In the past decade or so, considerable research attention has been devoted to the efficient solution of problems (3)-(4), especially when n is very large. Nowadays Web page ranking often deals with over billion pages and social networks graphs may consist of more than 100 millions nodes. Solving eigen- or linear systems of such huge sizes is a computational challenge that demands large costs in hardware and time. Direct solvers based on matrix factorizations are clearly not affordable to solve problems of this dimension in linear time and space. Iterative solvers based on sparse matrix-vector products and dense vector operations are thus mandatory to use. In applications where the damping factor α approaches 1, the convergence rate of classical iterative solvers such as the Power method, the Jacobi method and the Gauss-Seidel method will seriously deteriorate, and more robust algorithms need to be used. For example, the Power method initially used by Google for computing PageRank vectors converges about 10 times slower in the case when $\alpha = 0.99$ compared with $\alpha = 0.9$. Because the size of PageRank problems keeps growing, the development of more sophisticated and efficient solvers for this problem class are an active research topic of computational mathematics. In this paper, we present a new multi-step low-rank factorization method that can be used to significantly reduce the memory costs required for realistic PageRank calculations. Throughout the paper, we adopt MATLAB notation and write $A(i,:)$ to denote the i^{th} row of matrix A, while $A(:,j)$ represents the j_{th} column of A, and $A(i_1 : i_2, j_1 : j_2)$ is the block of A from row i_1 to row i_2 and from column j_1 to column j_2.

2. Multi-step low-rank factorization of PageRank matrices

An iterative method for solving the nonsingular PageRank linear system $Ax = v$ needs to store only the coefficient matrix A and the right-hand side v, plus a few extra temporary vectors. By using sparse storage formats, such as the Compressed Sparse Row (CSR) or the Compressed Sparse Column (CSC) formats, the storage costs mainly depend on the number of nonzeros $nnz(A)$ of A. The algorithmic complexity also depends on $nnz(A)$, since the most expensive operation of an iterative method is the sparse matrix-vector multiplication Au [2]. Therefore, for very large values of n, significant memory and time efforts could be saved by reducing $nnz(A)$. For this purpose, we recall below two important properties of the transition matrix P that determines the structure of $A = I - \alpha P$.

Property 2.1 *(Property 2.1 in [10]) The nonzero entries in each column of the transition matrix P defined in (1) are positive and have the same value.*

For example, the transition matrix P corresponding to the Web matrix G below has the following form:

$$G = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \rightarrow P = \begin{bmatrix} \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{3} \\ 0 & \frac{1}{3} & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} & \frac{1}{3} \\ \frac{1}{2} & \frac{1}{3} & \frac{1}{4} & \frac{1}{3} \end{bmatrix}. \tag{5}$$

Property 2.2 *(Property 2.2 in [10]) Same patterns in the rows of G define identical subrows in P.*

For instance, in the previous example, the last two rows of G have a common pattern in the last two columns. Consequently, the corresponding subrows in P are identical and have values $\left(\frac{1}{4}, \frac{1}{3}\right)$. Property 2.2 indicates that any two rows of P with similar nonzero pattern contain many identical entries that can be zeroed out by substracting one row from the other [10], and full blocks of P consisting of rows with same pattern can be suitably compressed by a low-rank matrix representation. For example, in the matrix below

$$P = \begin{bmatrix} 0 & * & * & 0 & * \\ * & * & 0 & * & * \\ * & * & * & * & 0 \\ * & * & * & * & 0 \\ 0 & 0 & * & 0 & * \end{bmatrix}, \tag{6}$$

the 2^{nd}, 3^{rd} and 4^{th} rows of P have nonzero entries in the 1^{st}, 2^{nd} and 4^{th} columns. Therefore, we can compress those nonzero entries by the following low-rank decomposition:

[2] Hereafter, u means an arbitrary vector with appropriate size.

$$P = \begin{bmatrix} 0 & * & * & 0 & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & * & 0 & 0 \\ 0 & 0 & * & 0 & 0 \\ 0 & 0 & * & 0 & * \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ * & * & 0 & * & 0 \\ * & * & 0 & * & 0 \\ * & * & 0 & * & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & * & * & 0 & * \\ 0 & 0 & 0 & 0 & * \\ 0 & 0 & * & 0 & 0 \\ 0 & 0 & * & 0 & 0 \\ 0 & 0 & * & 0 & * \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{bmatrix} [*, *, 0, *, 0] = P_R + fh^T. \tag{7}$$

The latter form enables to save the storage of three matrix elements. Any submatrix of P consisting of rows with the same nonzero pattern can be compressed. The same operation can be applied to the remaining matrix P_R, recursively, until no more nonzero elements can be further compressed. In practice, such low-rank factorizations will be implemented on the matrix αP to produce a matrix decomposition of the form $\alpha P = \bar{D} + \bar{F}\bar{H}$, where D is the final remaining matrix, F and H are respectively formed by the columns and the rows generated by the low-rank factorizations. Note that since $A = I - \alpha P = (I - \bar{D}) - \bar{F}\bar{H}$, we can also write $A = D + FH$, where $D = I - \bar{D}$ is a nonsingular M-matrix, $F = -\bar{F}$ and $H = \bar{H}$. Therefore, matrix A can be compressed as well and our goal is to maximimize the degree of compression of A.

We define a metric, hereafter referred to as the *compression number*, that counts the reduction of nonzero matrix elements to be stored after compression, i.e. $Comp(A) = nnz(A) - nnz(D) - nnz(H) - nnz(F)$. Note that the larger the compression number, the more efficient the algorithm. To start, we select the, say i^{th}, row of P that can maximize the compression number. According to Property 2.2, the largest amount of nonzero matrix elements that can be saved due to compressions on the i^{th} row is given by the sum of number of nonzeros in the columns of P corresponding to the nonzero elements of this row. In the given Example (6), the compression of any block of P that contains the first row, $P(1,:)$, can save at most all the nonzeros in the three columns $P(:,2)$, $P(:,3)$ and $P(:,5)$. Therefore, an upper bound on the compression number obtained by a rank-1 factorization of any block of P that contains $P(1,:)$ is given by $nnz(P(:,2)) + nnz(P(:,3)) + nnz(P(:,5))$, that is equal to 11. For the i^{th} row, we denote such upper bound on its maximum achievable compression number as $maxcom(i)$.

Motivated by these considerations, we adopt the following strategy: initially, we select the row of P with the largest *maxcom* value. Suppose that it is the i^{th} row $P(i,:)$ of P. Then, we traverse the pattern of P to find other rows that can be merged with $P(i,:)$ to form a suitable block that can be effectively compressed by a rank-1 decomposition. Clearly, $P(i,:)$ should be merged with rows that have the most similar sparsity patterns, as this may potentially lead to a higher compression number. Referring to Example (6), the 3^{rd} and 4^{th} rows $P(3,:)$ and $P(4,:)$ have both the largest *maxcom* value. We select $P(3,:)$ as a reference row. The row with the highest pattern similarity with $P(3,:)$, that is the largest number of nonzero elements in the same column indices, is $P(4,:)$ followed by $P(2,:)$, $P(1,:)$ and $P(5,:)$. The traverse follows this order. The rank-1 factorization of the block $P(3:4, 1:4)$ will increase the compression number by 2 since $P(3,:)$ and $P(4,:)$ have nonzero elements in their first four columns. However, it is convenient to merge $P(4,:)$ and $P(3,:)$ also with $P(2,:)$, because the rank-1 factorization of the nonzero

block $P(2:4,[1,2,4])$ increases the compression number by 3 and therefore a better compression can be achieved. On the other hand, if $P(1,:)$ is merged with $P(2,:)$, $P(3,:)$ and $P(4,:)$, the rank-1 matrix factorization of the block $P(1:4,3)$ can not increase further the compression number, since these four rows are all nonzero only at column 2. Therefore, $P(1,:)$ is not merged with the other three selected rows, the search procedure is stopped, and the rank-1 factorization $P = P_R + fh^T$ is computed as (7). The same process can be repeated on matrix P_R. As mentioned above, such multi-step low-rank factorization in practice is implemented on the matrix αP, yielding the decompositions $\alpha P = \bar{D} + \bar{F}\bar{H}$ and $A = D + FH$.

Two important amendments need to be introduced in the above algorithm, in order to limit the time cost of the compression procedure without sacrificing considerably the number of reduced nonzeros. At first, prior to the multi-step low-rank factorization, the rows of αP are permuted in density increasing order, by moving the most sparse ones to the top and the most dense ones to the bottom of the matrix. For costs reduction, the most sparse rows with a number of nonzero elements equaling to $\theta \cdot nnz(A)$ are excluded from the traverses, where θ is a user-defined ratio. Besides, we set a maximum number of implementations of rank-1 decomposition for the multi-step low-rank factorization. We suggest setting this upper-limit as 1%-15% of the matrix dimension. The reason is twofold: 1) at each step, the algorithm selects and compresses the rank-1 block that may increase most the compression number; as the compression increase is expected to decrease with the number of rank-1 factorizations, a relatively small number of such factorizations should be preferred; 2) the number of rank-1 factorizations is equal to the dimension of the dense capacitance matrix $I + HD^{-1}F$ when preconditioner based on the Woodbury formula are constructed from the decomposition $A = D + FH$ as in [11]. Finally, the whole algorithm of the multi-step low-rank factorization is presented as Algorithm 1, where Matlab functions and their calling methods are used for description.

3. Numerical experiments

We investigate the performance of the multi-step low-rank factorization strategy on several Web adjacency matrices G extracted from the matrix repository of [12] and the Laboratory for Web Algorithmics [13,14,15]. The characteristics of each matrix G are reported in Table 1. The PageRank matrix A is constructed using the damping factor $\alpha = 0.9$. All numerical experiments are carried out in MATLAB R2018b on a 64-bit windows 7 computer equipped with an Intel core i3-8100 processor and 24GB RAM memory.

Table 1. Characteristics of the Web adjacency matrices G, where n is the dimension, and nnz is the number of nonzero elements.

Name	n	nnz	nnz/n^2
`uk-2007-100000`	100,000	3,050,615	3.1e-4
`cnr-2000`	325,557	3,216,152	3.0e-5
`in-2004`	1,382,908	16,917,053	8.8e-6

We set the values of θ as $0.1, 0.2, 0.3, 0.4$, and the upper-bound r on the number of rank-1 factorizations divided by n as $0.01, 0.05, 0.1, 0.15, 0.2$. For each matrix and

for each pair of parameter setting, we run the multi-step low-rank factorization on the matrix αP and report the time cost *Time* in seconds and the compression ratio $Comr = (nnz(F*H) - nnz(F) - nnz(H))/nnz(A)$. The results are reported in Table 2.

Table 2. The time cost and the compression ratio of the multi-step low-rank factorization.

Problem:	uk-2007-100000							
$\theta \rightarrow$	0.1		0.2		0.3		0.4	
$r \downarrow$	*Time*	*Comr*	*Time*	*Comr*	*Time*	*Comr*	*Time*	*Comr*
0.01	1.64	67.2%	0.95	65.9%	0.80	59.9%	0.70	51.9%
0.05	3.57	72.1%	1.60	67.4%	1.07	60.1%	0.76	51.9%
0.10	5.05	73.1%	2.28	67.4%	1.08	60.1%	0.77	51.9%
0.15	6.28	73.4%	2.47	67.4%	1.09	60.1%	0.76	51.9%
0.20	7.33	73.4%	2.49	67.4%	1.08	60.1%	0.76	51.9%
Problem:	cnr-2000							
$\theta \rightarrow$	0.1		0.2		0.3		0.4	
$r \downarrow$	*Time*	*Comr*	*Time*	*Comr*	*Time*	*Comr*	*Time*	*Comr*
0.01	11.4	51.9%	6.65	52.8%	4.02	51.9%	2.68	45.9%
0.05	33.6	59.0%	15.4	58.4%	8.34	53.1%	3.86	45.9%
0.10	58.3	60.4%	24.6	58.2%	8.19	53.1%	3.89	45.9%
0.15	81.8	60.5%	26.7	58.2%	8.22	53.1%	3.90	45.9%
0.20	104.9	60.1%	26.5	58.2%	8.05	53.1%	3.98	45.9%
Problem:	in-2004							
$\theta \rightarrow$	0.1		0.2		0.3		0.4	
$r \downarrow$	*Time*	*Comr*	*Time*	*Comr*	*Time*	*Comr*	*Time*	*Comr*
0.01	122	56.4%	45.3	58.6%	16.5	56.6%	9.95	49.7%
0.05	478	66.3%	131	63.2%	30.6	56.9%	10.7	49.7%
0.10	872	67.1%	195	63.1%	30.6	56.9%	10.7	49.7%
0.15	1284	66.8%	196	63.1%	30.9	56.9%	10.7	49.7%
0.20	1632	66.4%	197	63.1%	30.9	56.9%	10.7	49.7%

In our experiments, both the time cost and the compression ratio of the multi-step factorization generally decrease with θ while they increase with r. Consistently with our analysis, a small value of r can guarantee a good compression ratio that tends to raise very slowly with r. In Table 2, the compression ratio is almost constant for each θ and for values of r in the range between 0.05 to 0.20. This observation suggests that a suitable setting for the value of r can be $r \leq 0.05$. Besides, when θ increases, the time cost decreases much faster than the compression ratio, and setting $\theta = 0.3$ or 0.4 is an advisable choice. In such cases, the compression ratio tends to increase little when r increases from 0.01 to 0.05. Thus, it is better to set $r = 0.01$. Seen from this table, the multi-step low-rank factorization can achieve large compression ratio at low to moderate computational cost. Therefore, it can be useful to reduce significantly the storage cost of Web graphes and the PageRank computations. This may be an attractive advantage when solving PageRank problems, especially when the damping factor approaches 1 or several PageRank problems corresponding to the same Web structure need to be solved.

Algorithm 1 Multi-step low-rank factorization for PageRank problems

Input: G, αP, θ, r

Compute the number $rownnz$ of nonzero elements for each row of G, set $k = 1$;
Permute and mark the rows of G and αP in a $rownnz$ increasing order until the amount of nonzero elements in the marked rows reaches $\theta nnz(G)$;
Suppose the first unmarked row is row_l, generate $PickG = G(l:n,:)$ and $PickA = A(l:n,:)$;
Compute the number of nonzeros in each column of $PickG$, store it in vector $colsum$;
Compute the $maxcom$ value of each unmarked row by $maxcom = pickG * colsum'$;
for $i = 1 : r * size(P,2)$ **do**
  Pick a row row_j from the unmarked rows with the largest $maxcom$ value;
  Set $common_j = row_j$;
  if $maxcom(j) == 0$ **then**
    Break;
  end if
  Find the column indexes of the nonzero elements of row_j, store them in vector $neighbors$;
  Compute the number of nonzeros of columns $neighbors$ distributed in each row by $repeats = sum(pickG(:,neighbours),2)$;
  Permute the rows by $[sumrow, order_r] = sort(repeats,'descend')$;
  Set $compressed_j$=0, $index_j = 1$, $N(k).group = [j]$;
  if $sumrow(2) < 1$ **then**
    Set $maxcom(j) = 0$ and $rownnz(l+j-1) = 0$;
    Continue;
  end if
  for $s = 1 : n-l+1$ **do**
    Compute the common nonzero indexes by $tem_{common} = common_j .* pickG(order_r(s),:)$;
    Compute the number of common nonzero indexes by $temcommon_{nnz} = nnz(tem_{common})$;
    if $temcommon_{nnz} * index_j > compressed_j$ **then**
      Set $N(k).group = [N(k).group, order_r(s)]$, $common_j = tem_{common}$;
      Set $compressed_j = temcommon_{nnz} * index_j$;
      Set $index_j = index_j + 1$;
    end if
  end for
  Set $neighbour = find(common_j)$;
  Update $rownnz(N(k).group) = rownnz(N(k).group + l - 1) - nnz(common_j)$;
  Update $maxcom = maxcom - sum(pickG(:,neighbour),2) * index_j$;
  Set $sumneighbours = sum(pickG(:,neighbour),2)$;
  Update $maxcom(N(k).group) = maxcom(N(k).group) - sum(sumneighbours) + sumneighbours(N(k).group) * index_j$;
  Compute $common_{neighbour} = find(common_j)$;
  Set $N(k).common = common_j .* pickA(j,:)$, $N(k).col = k * ones(1, index_j)$, $k = k+1$;
end for
Set $rowf = [N(1:k-1).group]$, $colf = [N(1:rank).col]$, $lenv = length(rowf)$, $vf = ones(1,lenv)$;
Set $temF = sparse(rowf, colf, vf, size(PickG,1), k-1)$;
Set $F = sparse(n, k-1)$, $F(i:n,:) = temF$, $H = cat(2, N.common)$, $H = H'$;
Permute the $rownnz$ back to the original order, and permute the rows of F accordingly;
return F and H.

4. Conclusions and future work

We have presented a novel multi-step low-rank factorization method to reduce the large memory cost demanded for realistic PageRank calculations. Differently from the low-rank factorizations proposed in [11], the proposed method can be implemented to the whole matrix A instead of only the off-diagonal blocks of A, thus leading to higher compression level. The outcome of the algorithm is a low-rank decomposition of the form $A = D + FH$ that is suitable to construct effective preconditioners, e.g. based on the Woodbury formula [11]. Additionally, matrix-vector multiplications that account for the most expensive operation in classical iterative solvers are expected to be cheaper to compute using the low-rank representation $A = D + FH$. The proposed technique can be especially effective when the network has many nodes with similar in-link distributions.

In future work, it will be interesting to investigate the structure characteristics, such as the distributions of the out-degrees and in-degrees and the similarity of in-link (out-link) distributions of large networks from more fields to develop optimal or highly efficient pre-processing techniques for reducing the memory and algorithmic costs of computing large-scale PageRank centralities. An important application of these techniques can be the solution of PageRank problems with multiple damping factors like [16] or with slight modification of the network, that are computationally very demanding.

References

[1] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. 1998.
[2] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.
[3] Langville AN, Meyer CD. A reordering for the PageRank problem. SIAM J. Sci. Comput. 2006 Feb;27(6):2112-20.
[4] Gleich DF. PageRank beyond the web. SIAM Rev. 2005 Aug;57(3):321-63.
[5] Henni K, Mezghani N, Gouin-Vallerand C. Unsupervised graph-based feature selection via subspace and pagerank centrality. Expert Systems with Applications. 2018;114:46-53.
[6] Morrison JL, Breitling R, Higham DJ, Gilbert DR. GeneRank: using search engine technology for the analysis of microarray experiments. BMC bioinformatics. 2005 Sep;6(1):1-14.
[7] Wu G, Zhang Y, Wei Y. Accelerating the Arnoldi-type algorithm for the PageRank problem and the ProteinRank problem. Journal of Scientific Computing. 2013 Feb;57(1):74-104.
[8] Ding Y, Yan E, Frazho A, Caverlee J. PageRank for ranking authors in co-citation networks. Journal of the American Society for Information Science and Technology. 2009;60(11):2229-43.
[9] Pedroche F, Romance M, Criado R. A biplex approach to PageRank centrality: From classic to multiplex networks. Chaos: An Interdisciplinary Journal of Nonlinear Science. 2017; 26(6): 065301.
[10] Shen ZL, Huang TZ, Carpentieri B, Gu XM, Wen C. An efficient elimination strategy for solving PageRank problems. Appl. Math. Comput. 2017 Apr; 298(1):111-22.
[11] Shen ZL, Huang TZ, Carpentieri B, Wen C, Gu XM, Tan XY. Off-diagonal low-rank preconditioner for difficult PageRank problems. J. Comput. Appl. Math. 2019; 346 (1):456-70.
[12] Davis TA, Hu Y. The University of Florida sparse matrix collection. ACM Trans. Math. Softw. 2011; 38(1): 1:1-1:25. Available at the URL: `http://www.cise.ufl.edu/research/sparse/matrices`.
[13] Boldi P, Vigna S. The WebGraph Framework I: Compression Techniques. In: Proc. of the 13th international conference on World Wide Web. 2004: 595-602.
[14] Boldi P, Rosa M, Santini M, Vigna S. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In: Proc. of the 20th international conference on World wide web. 2011: 587-596.
[15] Boldi P, Codenotti B, Santini M, Vigna S. Ubicrawler: A scalable fully distributed Web crawler. Software: Practice and Experience. 2004; 34: 711-726.
[16] Constantine PG, Gleich DF. Random alpha pagerank. Internet Mathematics. 2009; 6: 189-236.