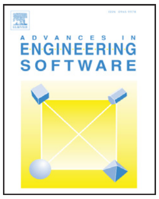
Advances in Engineering Software 000 (2016) 1–8

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft



A heuristic relaxed extrapolated algorithm for accelerating PageRank

Héctor Migallón[a,*], Violeta Migallón[b], Juan A. Palomino[b], José Penadés[b]

[a] Department of Physics and Computer Architectures, University Miguel Hernández, E-03202 Elche, Alicante, Spain
[b] Department of Computer Science and Artificial Intelligence, University of Alicante, E-03071 Alicante, Spain

ARTICLE INFO

Article history:
Available online xxx

Keywords:
PageRank
Parallel algorithms
Power method
Relaxation and extrapolation
Shared memory
Distributed memory

ABSTRACT

The PageRank algorithm for determining the importance of Web pages has become a central technique in Web search. This algorithm uses the Power method to compute successive iterates that converge to the principal eigenvector of the Markov chain representing the Web link graph. In this work we present an effective heuristic Relaxed and Extrapolated algorithm based on the Power method that accelerates its convergence. A hybrid parallel implementation of this algorithm has been designed by combining various OpenMP threads for each MPI process and several strategies of data distribution among nodes have been analyzed. The results show that the proposed algorithm can significantly speed up the convergence time with respect to the parallel Power algorithm.

1. Introduction

One of the most difficult problems in Web search is the ranking of the results recalled in response to a user query. Since contemporary Web crawls discover billions of pages, broad topic queries may result in recalling hundreds of thousand of pages containing the query terms. Only a few dozens of the recalled pages are actually presented to the user. Moreover, these results are presented in order of relevance. A variety of ranking features are used by Internet search engines to come up with a good order. Some of the query-independent features are based on page content. Some utilize the hyperlink structure of the Web. The PageRank algorithm [1] is one of those that introduces a content neutral-ranking function over Web pages. This ranking is applied to the set of pages returned by the Google search engine in response to posting a search query. PageRank is based in part on two simple common sense concepts: A page is important if many important pages include links to it and a page containing many links has reduced impact on the importance of the pages it links to. This model has been used by Google as part of its search engine technology.

PageRank is essentially the stationary distribution vector of a Markov chain whose transition matrix is a convex combination of the Web link graph and a certain rank 1 matrix. A key parameter in the model is the damping factor, a scalar that determines the weight given to the Web link graph in the model. Due to the great size and sparsity of the matrix that represents the Web link graph, methods based on decomposition are considered infeasible; instead, iterative methods are used, where the computation is dominated by matrix-vector products; see e.g., [2]. Traditionally, PageRank has been computed using the Power method. Many methods to accelerate the Power method have been developed such that extrapolation methods, block-structure methods or adaptive methods; see e.g., [3–6] and the references cited therein. In recent years opportunities for parallel execution have broadened their scope; see e.g., [7–10].

In this paper we present an effective heuristic algorithm based on the Power method and the use of both relaxation and extrapolation techniques, and we analyze the parallelization of this heuristic for accelerating the computation of PageRank. The remainder of the paper is structured as follows. In Section 2 we provide a brief description of the PageRank problem and we introduce the Power method and some acceleration methods based on the extrapolation technique. In Section 3, the proposed parallel heuristic algorithm is introduced. The numerical experiments performed in Section 4 show the behavior of this heuristic on both shared and distributed memory multicore architectures. Finally, we conclude in Section 5. This paper is based upon Migallón et al. [11], but the current paper includes the following additional research: more strategies for partitioning the link matrix among nodes are investigated and new numerical experiments are performed and explored.

2. Computing the PageRank

The PageRank vector [1] is a probability distribution used to represent the likelihood that a person randomly clicking on links will arrive at any particular page. This problem can be formulated using matrices. Let $G = [g_{ij}]_{i,j=1}^{n}$ be a Web graph adjacency matrix

[*] Corresponding author. Tel.: +34966658390; fax: +34966658814.
E-mail address: hmigallon@umh.es (H. Migallón).

http://dx.doi.org/10.1016/j.advengsoft.2016.01.024
0965-9978/© 2016 Civil-Comp Ltd. and Elsevier Ltd. All rights reserved.

with elements $g_{ij} = 1$ when there is a link from page j to page i, with $i \neq j$, and zero otherwise. Here n is the number of Web pages. From this matrix we can construct a transition matrix $P = [p_{ij}]_{i,j=1}^{n}$ as follows: $p_{ij} = \frac{g_{ij}}{c_j}$ if $c_j \neq 0$ and 0 otherwise, where $c_j = \sum_{i=1}^{n} g_{ij}$, $1 \leq j \leq n$, represents the number of out-links from a page j. For pages with a nonzero number of out-links, i.e., $c_j \neq 0$ for all $j, 1 \leq j \leq n$, the matrix P is column stochastic. Thus each element of this matrix has values between 0 and 1, and the sum of the components of each column is 1. In this case the PageRank vector can be obtained by solving the eigenvector problem $Px = x$. Since we are interested in a probability distribution, the sum of the components of x is assumed to be one. When the matrix $P \geq 0$ is irreducible (i.e., its graph is strongly connected) and stochastic, the original Power method [12] can be used to compute the PageRank vector. However, the Web contains many pages without out-links, called dangling nodes. Dangling pages present a problem for the mathematical PageRank formulation because in this case the matrix P is non-stochastic and then the Power method can not be used. Moreover, the matrix irreducibility is not satisfied for a Web graph. In order to overcome these difficulties, Page and Brin [1] change the transition matrix P to a column stochastic matrix $\bar{P} = \alpha(P + vd^T) + (1-\alpha)ve^T$, where $d \in \Re^n$ is the dangling page indicator defined by $d_i = 1$ if and only if $c_i = 0$ and the vector $v \in \Re^n$ is some probability distribution over pages. This model means that the random surfer jumps from a dangling page according to a distribution v. For this reason v is called a teleportation distribution. Originally uniform teleportation $v = \frac{e}{n}$ was used. Then, setting α such that $0 < \alpha < 1$ the matrix $\bar{P}$ is column stochastic, irreducible and therefore the Power method can be utilized to solve the stationary distribution of the ergodic Markov chain defined by $\bar{P}$, $\bar{P}x = x$.

A key parameter in this model is the damping factor α that determines the weight given to the Web link graph in the model. In the original formulation of PageRank [1] the Power method was applied using $\alpha = 0.85$. However, a higher value of α (close to 1) yields to a model that is mathematically closer to the actual link structure of the Web but makes the computation more difficult [13]. This parameter α controls the asymptotic rate of convergence and as $\alpha \to 1$, the expected number of iterations required for convergence increases dramatically and new approaches for accelerating the PageRank computation are required. In fact, the calculation of many PageRank vectors with different values of α looks promising in order to detect link spammers [14]. On the other hand, PageRank type algorithms are used in application areas other than Web search ([15], [16]) where the value of α often has a concrete meaning and the graphs have a different structure that may not resemble the bow-tie structure of the Web link graph. It is thus useful to consider a variety of values of α.

As each iteration of the Power method on a web-sized matrix is so expensive, reducing the number of iterations by a handful can save hours of computation. The extrapolation algorithms [13] are successful for reducing the work associated with the PageRank computation. These methods accelerate the convergence of PageRank using successive iterates of the Power method to estimate the nonprincipal eigenvectors of the hyperlink matrix, and periodically subtracting these estimates from the current iterate of the Power method. In [5] a quadratic extrapolation algorithm, based on the same idea as Aitken extrapolation, was presented. This work assumed that none of the nonprincipal eigenvalues of the hyperlink matrix were known.

Algorithm 1 shows the Extrapolation Power methods treated here, where $e = (1, 1, \ldots, 1)^T$ and the L_1 norm $\|x\|_1 = \sum_{i=1}^{n} |x_i|$ is used. This algorithm exploits the knowledge of eigenvalues of the hyperlink matrix. Moreover, the extrapolation needs to be applied only once; see e.g., [13]. Notice that if the extrapolation step (1) of Algorithm 1 is removed, the resulting iteration scheme describes the Power method applied to the matrix $\bar{P}$. Note that although the matrix $\bar{P}$ is dense, Algorithm 1 has been designed such that it is not necessary to construct explicitly the matrix $\bar{P}$.

Algorithm 1: Extrapolation Power method for solving $\bar{P}x = x$.

```
Initialization $x^0 = \frac{e}{n}$, $k = 0$;
repeat
    $x^{k+1} = \alpha P x^k$;
    $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1$;
    $x^{k+1} = x^{k+1} + \gamma v$;
    If $k + 1 == r + 2$ then $x^{k+1} = \frac{x^{k+1} - \alpha^r x^{k+1-r}}{1 - \alpha^r}$;      (1)
    $\delta = \|x^{k+1} - x^k\|_1$;
    $k = k + 1$;
until $\delta < \epsilon$;
```

3. Parallel algorithms

In order to design the parallel algorithms, we consider that P is partitioned into p row blocks. Each block P_i, $1 \leq i \leq p$, is a matrix of order $n_i \times n$, with $\sum_{i=1}^{p} n_i = n$. Analogously, we consider the vectors x^k and v partitioned according to the block structure of P. Then, Algorithm 2 describes the parallelization of the Power method for solving $\bar{P}x = x$, in which each process actualizes a block of the vector x^{k+1} and a synchronization of all processes is performed at each iteration to construct the global iterate vector x^{k+1}.

Algorithm 2: Parallel Power method.

```
Initialization $x^0 = \frac{e}{n}$, $k = 0$;
for $i = 1, 2, \ldots, p$, do in parallel
    repeat
        $x_i^{k+1} = \alpha P_i x^k$;
        Compute $\|x_i^{k+1}\|_1$;
        Perform a sum all-to-all reduction over $\|x_i^{k+1}\|_1$;
        $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1$;
        $x_i^{k+1} = x_i^{k+1} + \gamma v_i$;      (2)
        Perform an all-gather operation to obtain
        $x^{k+1} = [x_1^{k+1}, \ldots, x_p^{k+1}]$;
        Compute $\|x_i^{k+1} - x_i^k\|_1$;
        Perform a sum all-to-all reduction over $\|x_i^{k+1} - x_i^k\|_1$;
        $\delta = \|x^{k+1} - x^k\|_1$;
        $k = k + 1$;
    until $\delta < \epsilon$;
end
```

In general the asymptotic rate of convergence of the Power method applied to a matrix depends on the ratio of the two eigenvalues that are largest in magnitude, denoted by λ_1 and λ_2. Concretely, the asymptotic convergence rate is the rate at which $|\lambda_2/\lambda_1|^k$ tends to 0. The smaller this ratio, the quicker the Power method converges. For stochastic matrices such as $\bar{P}$, $\lambda_1 = 1$, therefore $|\lambda_2|$ governs the convergence. In general, numerically finding λ_2 for a matrix requires computational effort that one is not willing to spend just to get an estimate of the asymptotic rate of convergence. However, taking into account the relation between the spectra of the matrices $P + vd^T$ and $\bar{P}$, the link structure of the

Web makes it very likely that $|\lambda_2| \approx \alpha$; see e.g., [17]. Thus, a rough estimate of the number of iterations needed to converge to a tolerance level $\epsilon = 10^{-\zeta}$ is $\frac{\log_{10}\epsilon}{\log_{10}\alpha}$. Therefore, to produce scores with approximately a degree of accuracy of ζ digits about $\frac{-\zeta}{\log_{10}\alpha}$ iterations must be completed.

On the other hand, taking into account that the Power method is equivalent to use a Jacobi type splitting for solving the linear system $(I - \bar{P})x = 0$, to improve the rate of convergence we could use a relaxed Jacobi type splitting. Thus, a relaxation parameter $\beta > 0$ can be introduced in Algorithm 2 and replace the computation of x_i^{k+1} in (2) with the equation $x_i^{k+1} = \beta(x_i^{k+1} + \gamma v_i) + (1-\beta)x_i^k$. Clearly, with $\beta = 1$ equation (2) is recovered. In the case of $\beta \neq 1$ we have a parallel Relaxed Power method. Note that the relaxation parameter needs to be chosen in such a way that the convergence of the method is assured. Since $(I - \bar{P})$ is an irreducible singular M-matrix, the Relaxed Power method converges provided that $0 < \beta < \frac{2}{1+\vartheta(\bar{P})}$, where $\vartheta(\bar{P}) = \max\{|\lambda|, \lambda \in \sigma(\bar{P}), \lambda \neq 1\}$, and $\sigma(\bar{P})$ is the spectrum of $\bar{P}$ [18]. Therefore, it is expected that the Relaxed Power method will converge to the PageRank vector for $0 < \beta < \frac{2}{1+\alpha}$.

In a previous paper [19] we presented parallel Relaxed Extrapolated algorithms based on Algorithm 1. As it was shown, these algorithms can speed up the convergence time significantly with respect to the Power method with the same degree of accuracy. However, its convergence speed is very dependent on the choice of the parameter r used in the extrapolation. An optimal value for this parameter is hard to predict, and a poor choice of r could decelerate the convergence of Algorithm 1 in relation to the Power method. In order to reduce the number of iterations required by the PageRank Power method and accelerate the convergence of Algorithm 2, we design here a heuristic parallel Relaxed Extrapolated algorithm (Algorithm 3) in which the proposed value for r is obtained from the damping factor α by means of the expression $r = \left\lfloor \frac{1}{1-\alpha} \right\rfloor$ and the extrapolation is applied once at iteration $\left\lfloor \frac{1}{1-\alpha} \right\rfloor + 2$. Note that the column sums of $(I - \alpha P)^{-1}$ are less than or equal to $\frac{1}{1-\alpha}$ for the non-dangling nodes and $\|(I - \alpha(P + vd^T))^{-1}\|_1 = \frac{1}{1-\alpha}$ [17]. Table 1 shows the values of r in Algorithm 3 for different damping factors and the degree of accuracy $(-(r+1)\log_{10}\alpha)$ obtained before the extrapolation is performed at the $r+2$ iteration. In this way, we propose to performing the extrapolation before obtaining 1 place of accuracy in the approximate PageRank vector. Therefore, it is reasonable to test the stopping criterion only for the relaxed iterations (i.e. for $k > r+1$). Note the computation of $\|x^{k+1}\|_1$ and δ, in Algorithms 2 and 3, is also performed in parallel in such a way that each process i computes the portions $\|x_i^{k+1}\|_1$ and $\|x_i^{k+1} - x_i^k\|_1$ followed by a reduction of these values.

Algorithm 3: Heuristic Parallel Relaxed Extrapolated Power method (HRELEXT).

```
Initialization $x^0 = \frac{e}{n}$, $k = 0$, $r = \left\lfloor \frac{1}{1-\alpha} \right\rfloor$;
for $i = 1, 2, \ldots, p$, do in parallel
    repeat
        $x_i^{k+1} = \alpha P_i x^k$;
        Compute $\|x_i^{k+1}\|_1$;
        Perform a sum all-to-all reduction over $\|x_i^{k+1}\|_1$;
        $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1$;
        $x_i^{k+1} = x_i^{k+1} + \gamma v_i$;
        if $k+1 == r+2$ then $x_i^{k+1} = \frac{x_i^{k+1} - \alpha^r x_i^{k+1-r}}{1-\alpha^r}$;
        Perform an all-gather operation to obtain
        $x^{k+1} = [x_1^{k+1}, \ldots, x_p^{k+1}]$;
        $k = k + 1$;
    until $k > r + 1$;
    repeat
        $x_i^{k+1} = \alpha P_i x^k$;
        Compute $\|x_i^{k+1}\|_1$;
        Perform a sum all-to-all reduction over $\|x_i^{k+1}\|_1$;
        $\gamma = \|x^k\|_1 - \|x^{k+1}\|_1$;
        $x_i^{k+1} = x_i^{k+1} + \gamma v_i$;
        $x_i^{k+1} = \beta x_i^{k+1} + (1 - \beta) x_i^k$;
        Perform an all-gather operation to obtain
        $x^{k+1} = [x_1^{k+1}, \ldots, x_p^{k+1}]$;
        Compute $\|x_i^{k+1} - x_i^k\|_1$;
        Perform a sum all-to-all reduction over $\|x_i^{k+1} - x_i^k\|_1$;
        $\delta = \|x^{k+1} - x^k\|_1$;
        $k = k + 1$;
    until $\delta < \epsilon$;
end
```

Table 1
Values of r for Algorithm 3 and degree of accuracy obtained at the $r+1$ iteration.

α	r	Degree of accuracy
0.85	6	0.494068
0.95	20	0.467804
0.97	33	0.449761
0.98	50	0.447470
0.99	100	0.440845
0.995	200	0.437560

4. Experimental setup and results

In order to illustrate the behavior of these methods, the algorithms described here have been implemented on an HPC cluster of 26 nodes HP Proliant SL390s G7 connected through a network of low-latency QDR Infiniband-based. Each node consists of two Intel XEON X5660 hexacore at up to 2.8 GHz and 12MB cache per processor, with 48GB of RAM. The operating system is CentOS Linux 5.6 for x86 64 bit. The parallel environment has been managed using both MPI (Message Passing Interface) [20] and OpenMP (Open Multi-Processing) [21]. That is, an hybrid MPI/OpenMP implementation has been designed by combining various OpenMP threads for each MPI process. Adding OpenMP threading to an MPI code is an efficient way to run on multicore processors and nodes like those on this cluster. Since OpenMP operates in a shared memory space, it is possible to reduce the memory overhead associated with MPI tasks and reduce the need for replicated data across tasks. Concretely, let p be the number of performed processes, $p = s * c$ indicates that s nodes of the parallel platform have been used and for each one of these nodes, c OpenMP threads have been considered. Therefore, we use a philosophy of distributed shared memory using $p = s \times c$ processes or threads. Particularly, if $s = 1$ the algorithms are executed in shared memory by using $p = c$ threads on a single node. Conversely, if $c = 1$, we are working on distributed memory using $p = s$ nodes. To compute PageRank for large domains there is no possible way to work with the matrix in its full format because the memory requirements would be too high. Therefore, a sparse matrix format is needed in order to store the matrices. The Compressed Sparse Row (*CSR*) format is one of the most extensively used storage schemes for general sparse matrices, with minimal storage requirements. We represent the two vectors of indexes of the *CSR* format by integers without sign of

Table 2
Graphs collection.

Graph	Nodes (n)	Arcs (nnz)	Dang. nodes	Density	Memory
it-2004	41,291,594	1,150,725,436	12.76%	27.87	4.75GB
webbase-2001	118,142,155	1,019,903,190	23.41%	8.63	5.12GB
uk-2007-05	105,896,555	3,738,733,648	12.23%	35.31	15.11GB

Table 3
Number of iterations of the HRELEXT method, $\epsilon = 10^{-6}$.

it-2004	$\beta = 0.9$	$\beta = 0.95$	$\beta = 0.96$	$\beta = 0.97$	$\beta = 0.98$	$\beta = 0.99$
$\alpha = 0.85$	52	50	49	49	49	49
$\alpha = 0.95$	144	138	137	136	135	135
$\alpha = 0.98$	308	295	293	290	288	287
$\alpha = 0.99$	540	522	519	516	513	511
$\alpha = 0.995$	1006	984	980	976	973	970
webbase-2001	$\beta = 0.9$	$\beta = 0.95$	$\beta = 0.96$	$\beta = 0.97$	$\beta = 0.98$	$\beta = 0.99$
$\alpha = 0.85$	52	50	49	49	49	49
$\alpha = 0.95$	144	138	137	136	135	137
$\alpha = 0.98$	307	294	292	289	287	288
$\alpha = 0.99$	515	498	495	492	490	490
$\alpha = 0.995$	917	894	891	887	884	883
uk-2007-05	$\beta = 0.9$	$\beta = 0.95$	$\beta = 0.96$	$\beta = 0.97$	$\beta = 0.98$	$\beta = 0.99$
$\alpha = 0.85$	51	49	49	49	48	48
$\alpha = 0.95$	139	134	133	132	131	131
$\alpha = 0.98$	301	290	288	286	284	283
$\alpha = 0.99$	570	554	551	548	545	542
$\alpha = 0.995$	1118	1086	1080	1074	1069	1063

Table 4
Percentage of reduction in the number of iterations of the HRELEXT method with respect to the Power (PW) method, $\epsilon = 10^{-6}$.

Matrix	it-2004			webbase-2001			uk-2007-05		
α	PW	HRELEXT	% R.	PW	HRELEXT	% R.	PW	HRELEXT	% R.
0.85	60	49	18.33	62	49	20.97	56	48	14.29
0.95	180	135	25.00	185	137	25.95	162	131	19.14
0.97	297	224	24.58	306	228	25.49	263	206	21.67
0.98	441	287	34.92	456	288	36.84	386	283	26.68
0.99	869	511	41.20	904	490	45.80	745	542	27.25
0.995	1719	970	43.57	1800	883	50.94	1446	1063	26.49

32 bits, while the values and the iterate vectors are represented by means of double precision floating point with 64 bits. Taking into account that, for each column of the matrix, all nonzero elements are equal to a fixed value, it is stored once in an ordered vector. In this way the memory requirements to store the matrix with this modified *CSR* format (CSR') can be computed by the following expression $M_{CSR'} = 32(n+1) + 32nnz + 64n \approx 32(3n + nnz)$ bits. In the experiments we have used three datasets of different sizes, available from http://law.dsi.unimi.it [22]. These transition matrices have been generated from a web-crawl [23]. Table 2 summarizes, for each graph, the number of nodes n (matrix size), the number of arcs nnz (nonzero elements of the matrix), the percentage of dangling nodes, the density (arcs/nodes) and the memory requirements using the proposed CSR' format. This format [24] has involved a reduction of memory requirements of about $63-73\%$ with respect to the original *CSR* format.

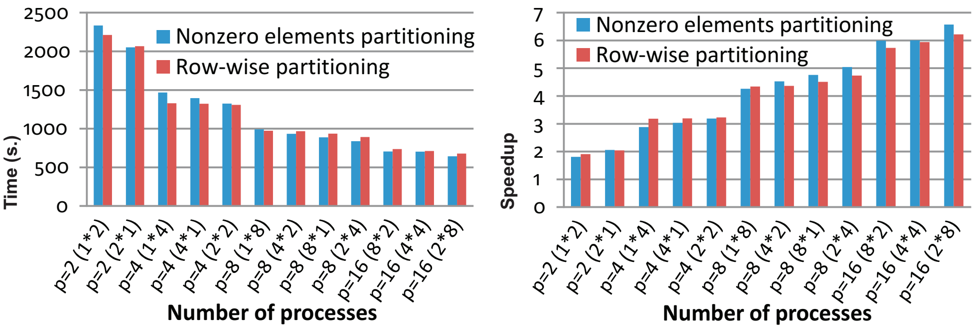
Table 3 shows the convergence rates of the HRELEXT method (Algorithm 3) for the tested matrices, setting $\epsilon = 10^{-6}$ and different values of α. Our experience indicates that, good choices of the relaxation parameter $\beta \in (0, \frac{2}{1+\alpha})$ in the proposed HRELEXT algorithm are between 0.97 and 0.99, obtaining similar convergence rates. This behavior is independent of the damping factor α. This is due to the fact that the number of iterations starts to decrease as β increases up to some good values of β close to 1, after which the number of iterations increases. Moreover, the number of iterations remains pretty constant when setting β between these values. It seems that setting $\beta = 0.99$ achieves a workable compromise between efficiency and effectiveness, being that for any damping factor $0 < \alpha < 1$, $\beta = 0.99$ has a behavior very similar to that of the best relaxation parameter, getting a considerable reduction in the number of iterations with respect to the Power method. Therefore, in our HRELEXT algorithm, we have set this value for the relaxation parameter β. Tabla 4 compares the convergence rates for the Power method and the HRELEXT algorithm for the datasets of Table 2. As it can be seen in this table, the proposed HRELEXT algorithm has substantially improved the convergence rate relative to the Power method, more specially for values of α close to 1.

Table 5
HRELEXT versus In/Out methods, $\alpha = 0.99$, SM $p = 8(1 * 8)$, uk-2007-05 matrix.

	$\epsilon = 10^{-5}$	$\epsilon = 10^{-7}$
POWER It.	531	964
In/Out It. gain	12.8%	11.1%
In/Out time gain	15.3%	13.7%
HRELEXT It. gain	31.8%	21.7%
HRELEXT time gain	37.5%	26.7%

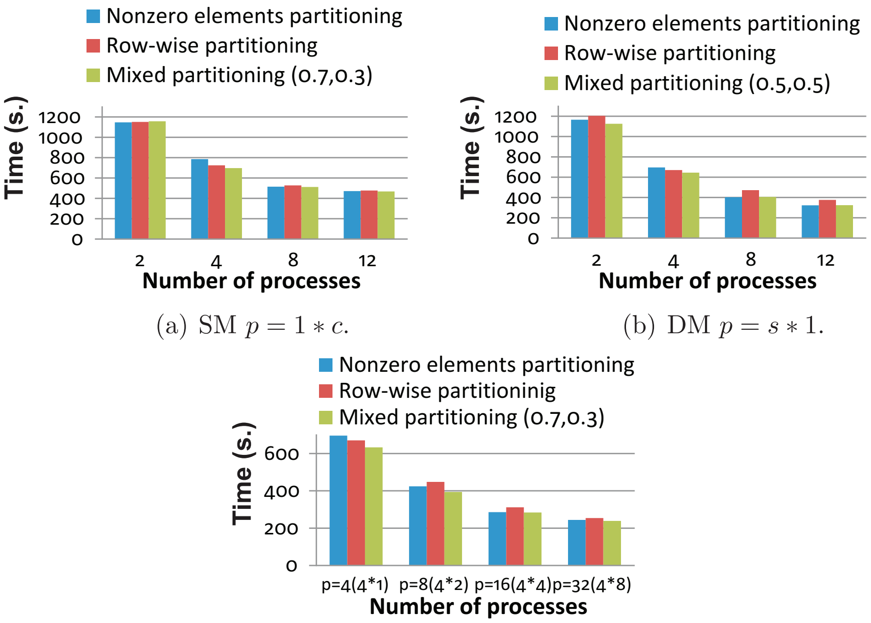
Implementing the PageRank calculations in a parallel environment opens several possibilities of data partitioning (i.e., how the

Please cite this article as: H. Migallón et al., A heuristic relaxed extrapolated algorithm for accelerating PageRank, Advances in Engineering Software (2016), http://dx.doi.org/10.1016/j.advengsoft.2016.01.024



(a) Time. (b) Speedup.

Fig. 1. Row-wise versus nonzero elements distribution. Parallel HRELEXT method, $\alpha = 0.995$, $p = s * c$, $\epsilon = 10^{-6}$, webbase-2001 matrix.



(a) SM $p = 1 * c$. (b) DM $p = s * 1$.

(c) DSM $p = 4 * c$.

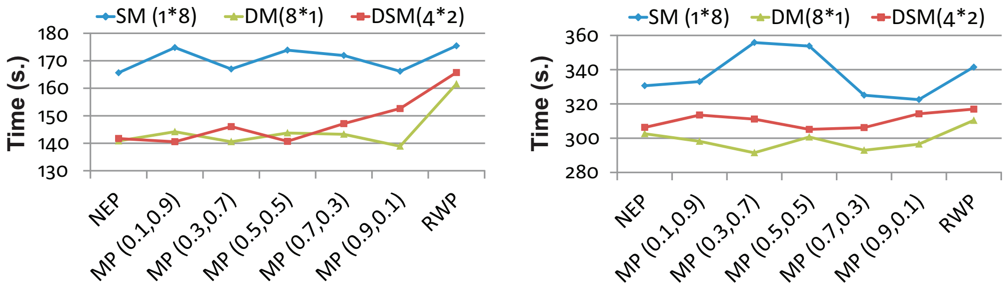
Fig. 2. Comparison of data partitioning strategies. Parallel HRELEXT method, $\alpha = 0.98$, $\epsilon = 10^{-6}$, uk-2007-05 matrix.

data are divided among nodes) and load balancing (i.e., to ensure that all nodes perform similar amount of work). The most expensive operation performed in the calculation of the PageRank values is a matrix-vector multiplication. This is a perfectly parallel operation with several possible methods for partitioning both the matrix and the vector. We have considered several strategies for partitioning the link matrix among nodes such as it is described in Algorithm 4 . In this algorithm, n represents the number of rows of the matrix, nnz_j is the number of nonzero elements in the jth row with $nnz = \sum_{j=1}^{n} nnz_j$, s is the number of partitions to be performing and the vector $[PART_0, \ldots, PART_s]$ stores the indexes of the partitions. Furthermore, ω_n, $\omega_{nnz} \in \Re$ such that $\omega_n + \omega_{nnz} = 1$ attach a weight to the number of rows and to the number of nonzero elements, respectively. Note that when $\omega_n = 1$ and $\omega_{nnz} = 0$, a row-wise distribution (row-wise partitioning, RWP) is chosen, where each node gets the same amount of rows, while by setting $\omega_n = 0$ and $\omega_{nnz} = 1$, each node has to handle the same amount of nonzero elements (nonzero elements partitioning, NEP). In the remainder cases, Algorithm 4 obtains different mixed distributions (mixed partitioning, MP) taking into account the weighting values (ω_n, ω_{nnz}) considered in the algorithm.

Algorithm 4: Partitioning method.

```
$umbral = \frac{n\omega_n + nnz\omega_{nnz}}{s}$;
$PART_0 = cont = j = 0$;
for $i = 1, 2, \ldots, s-1$, do
    while $cont \leq umbral$ do
        $j = j + 1$;
        $cont = cont + (\omega_n + nnz_j\omega_{nnz})$;
    end
    $j = j - 1$;
    $PART_i = j$;
    $cont = 0$;
end
$PART_s = n$;
```

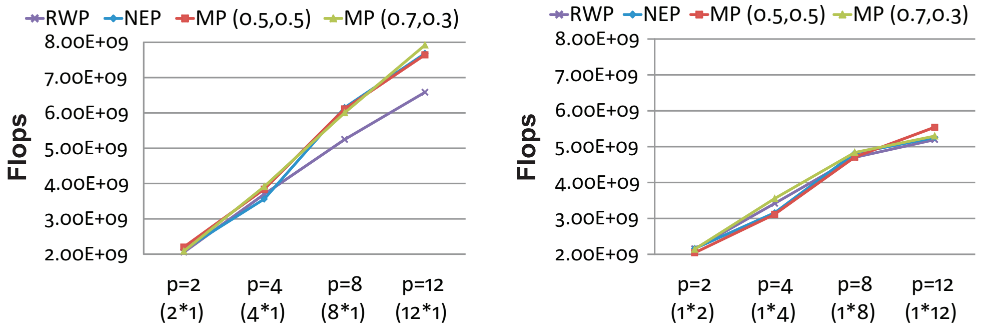
Fig. 1 shows the time that the HRELEXT algorithm takes for computing PageRank using row-wise partitioning and nonzero elements partitioning and the achieved speedup , varying the number of processes, for the webbase-2001 matrix. Figs. 2 and 3 compare these two strategies of data distribution with the use of



(a) it-2004.

(b) webbase-2001.

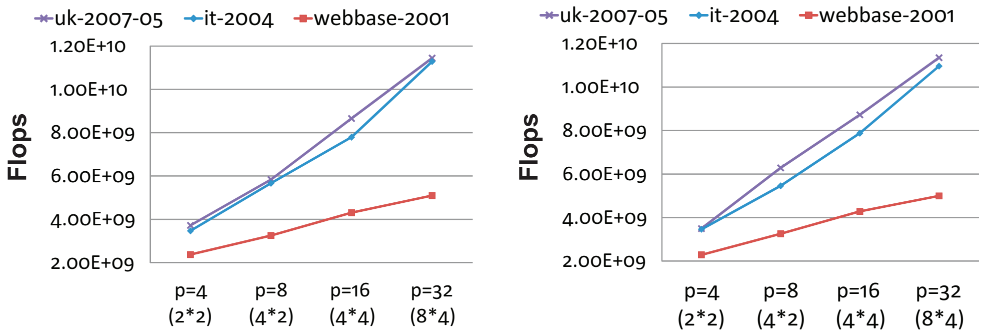
Fig. 3. Comparison of data partitioning strategies. Parallel HRELEXT method, $\alpha = 0.98$, $\epsilon = 10^{-6}$, it-2004 and webbase-2001 matrices,.



(a) Distributed Memory.

(b) Shared Memory.

Fig. 4. Data partitioning strategies performance. Parallel HRELEXT method, $\alpha = 0.98$, $\epsilon = 10^{-6}$, uk-2007-05 matrix.



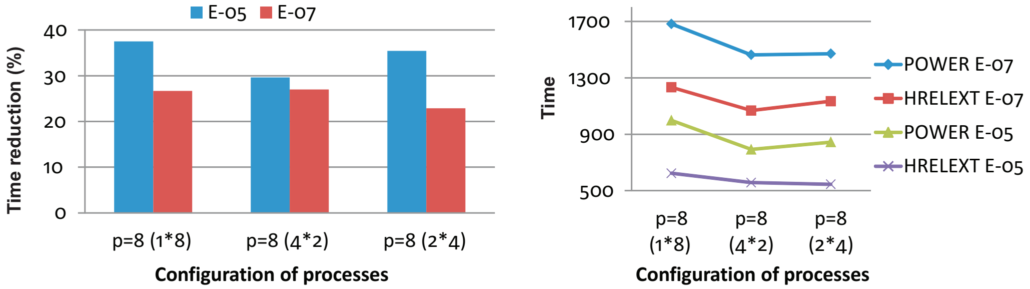
(a) Nonzero elements partitioning, NEP.

(b) Mixed partitioning, MP (0.7,0.3).

Fig. 5. Performance of Parallel HRELEXT method on shared distributed memory, $\alpha = 0.98$, $\epsilon = 10^{-6}$.

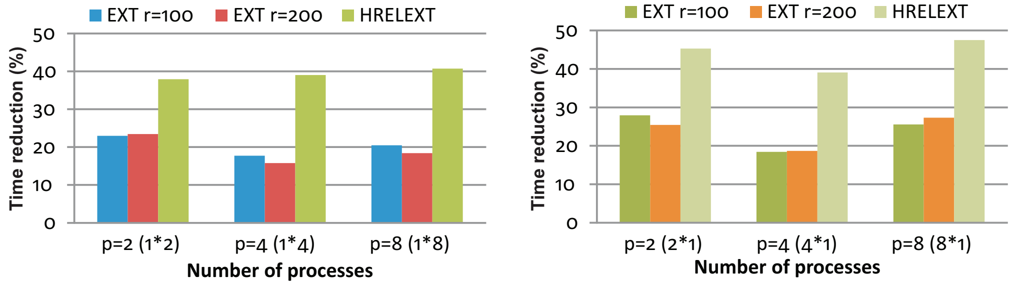
several mixed distribution according to Algorithm 4. Generally, the nonzero elements partitioning is better than the row-wise partitioning. However, we can find a mixed distribution between rows and nonzero elements that reduces the running time with respect to the nonzero elements partitioning. For example, for the uk-2007-05 matrix, Fig. 2 tell us that for both shared memory (SM) and distributed shared memory (DSM) with $p = 4 * c$ processes, the mixed partitioning using $\omega_n = 0.7$ and $\omega_{nnz} = 0.3$ behaves better than those distribution strategies. For this matrix, using distributed memory (DM), we have obtained that $\omega_n = \omega_{nnz} = 0.5$ is a good choice for the mixed partitioning. However, the choice of optimal weighting values (ω_n, ω_{nnz}) depends on the structure of the matrix, the number of nodes used and the proposed processes configuration; see also Fig. 3. Figs. 4 and 5 provide comparisons for the HRELEXT algorithm in floating-point operations per second (Flops) for various implementations and configurations. We obtain an acceptable speedup with OpenMP using the 12 available cores of one node but not comparable with the speedup obtained with MPI using distributed memory. Note that, for the uk-2007-05 matrix, 5.6 GFlops are achieved with 12 cores in Fig. 4(b). However, using distributed memory 8 Gflops are achieved for this matrix; see Fig. 4(a). However, in order to deal with larger problems and to use all the available memory in an actual SMP computer, the best strategy of parallelization needs to use at the same time the benefits of shared and distributed memory multiprocessors. Usually the best parallel results have been obtained using 1, 2 or 4 threads in each node. As it can be seen, in Fig. 5, the higher the density of the matrix, the better the results obtained using distributed shared memory.

Fig. 6 compares the execution time of the Power method and the HRELEXT methods varying the stopping criterion for the uk-2007-05 matrix. As it can be seen the HRELEXT algorithm speeds



(a) Percentage of time reduction. (b) Time.

Fig. 6. HRELEXT versus Power method, DSM $p = 8 = s * c$, $\alpha = 0.99$, uk-2007-05 matrix.



(a) SM $p = 1 * c$. (b) DM $p = s * 1$.

Fig. 7. HRELEXT versus extrapolated Power methods. Percentage of time reduction with respect to the parallel Power method, $\alpha = 0.99$, $\epsilon = 10^{-6}$, it-2004 matrix.

up the convergence time significantly with respect to the parallel Power algorithm. We have also compared the HRELEXT algorithm with the Extrapolation Power algorithm (see Algorithm 1) and with the inner-outer (In/Out) iterative algorithms proposed in [7]. Fig. 7 compares the percentage of time reduction of the HRELEXT algorithm and the best parallel Extrapolation Power algorithms obtained when varying the value of r, in relation to the parallel Power method. Table 5 illustrates the gain achieved for the parallel HRELEXT algorithm and the parallel inner-outer algorithms in relation to the parallel Power method. As it can be appreciated in Fig. 7 and Table 5 our proposed HRELEXT algorithm has a better convergence rate and accelerates the convergence time more significantly than those algorithms.

5. Conclusions

In this paper an effective heuristic parallel algorithm (HRELEXT) for accelerating PageRank is presented. The HRELEXT algorithm is based on the Power method and uses relaxation and extrapolation techniques. The parallel implementation have been developed using a mixed MPI/OpenMP model and several strategies for partitioning the link matrix among nodes. We have compared HRELEXT with the extrapolated methods and some inner-outer methods obtaining that HRELEXT has considerably improved the convergence rate relative to the Power method and behaves better than those methods.

Acknowledgments

This research was partially supported by the Spanish Ministry of Science and Innovation under Grant Number TIN2011-26254 and Grant Number TIN2015-66972-C5-4-R, and by the European Union FEDER (CAPAP-H5 network TIN2014-53522-REDT).

References

[1] Page L, Brin S, Motwani R, Winograd T. The pagerank citation ranking: Bringing order to the web. Technical report. Stanford Digital Library Technologies Project; 1999.
[2] Berkhin P. A survey on pagerank computing. Internet Math 2005;2(1):73–120.
[3] Kamvar SD, Haveliwala TH, Golub GH. Adaptive methods for the computation of pagerank. Linear Algebra Appl 2004;386:51–65.
[4] Kamvar SD, Haveliwala TH, Manning CD, Golub GH. Exploiting the block structure of the web for computing pagerank. Technical report, sccm-03-02. Stanford University; 2003.
[5] Kamvar SD, Haveliwala TH, Manning CD, Golub GH. Extrapolation methods for accelerating pagerank computations. In: Proceedings of the twelfth international world wide web conference. ACM Press; 2003. p. 261–70.
[6] Wu G, Wei Y. An arnoldi-extrapolation algorithm for computing pagerank. J Comput Appl Math 2010;234:3196–212.
[7] Gleich D, Gray A, Greif C, Lau T. An inner-outer iteration for computing pagerank. SIAM J Sci Comput 2010;32(1):349–71.
[8] Gleich D, Zhukov L, Berkhin P. Fast parallel pagerank: a linear system approach. Technical report yrl-2004-038. Yahoo! Research Labs; 2004.
[9] Rungsawang A, Manaskasemsak B. Fast pagerank computation on a GPU cluster. In: Stotzka R, Schiffers M, Cotronis Y, editors. Proceedings of the 20th euromicro international conference on parallel, distributed and network-based processing. Los Alamitos, California: IEEE Computer Society; 2012. p. 450–6.
[10] Rungsawang A, Manaskasemsak B. Parallel adaptive technique for computing pagerank. In: Cantarella JD, editor. Proceedings of the 14th euromicro international conference on parallel, distributed, and network-based processing, PDP'06. Los Alamitos, California: IEEE Computer Society; 2006. p. 15–20.
[11] Migallón H, Migallón V, Palomino JA, Penadés J. Parallel computation of pagerank using extrapolation and relaxation techniques. In: Iványi P, Topping BHV, editors. Proceedings of the ninth international conference on engineering computational technology. Stirlingshire, United Kingdom: Civil-Comp Press; 2014. Paper 29. doi: 10.4203/ccp.105.29.
[12] Wilkinson JH. The algebraic eigenvalue problem. Oxford: Oxford University Press; 1998.
[13] Kamvar SD. Numerical algorithms for personalized search in self-organizing information networks. Princeton, New Jersey: Princeton University Press; 2010.
[14] Zhang H, Goel A, Govindan R, Mason K, Van Roy B. Making eigenvector-based reputation systems robust to collusion. Lect Notes Comput Sci 2004;3243:92–104.
[15] Morrison JL, Breitling R, Higham DJ, Gilbert DR. Generank: Using search engine technology for the analysis of microarray experiments. BMC Bioinform 2005;6:233.

Please cite this article as: H. Migallón et al., A heuristic relaxed extrapolated algorithm for accelerating PageRank, Advances in Engineering Software (2016), http://dx.doi.org/10.1016/j.advengsoft.2016.01.024

[16] Singh R, Xu J, Berger B. Pairwise global alignment of protein interaction networks by matching neighborhood topology. In: Speed T, Huang H, editors. Research in computational molecular biology. Lect Notes Bioinform, vol. 4453. Heidelberg, Berlin: Springer-Verlag; 2007. p. 16–31.

[17] Langville A, Meyer CD. Google's Pagerank and beyond: the science of search engine rankings. Princeton, New Jersey: Princeton University Press; 2006.

[18] Song Y. Semiconvergence of extrapolated iterative methods for singular linear systems. J Comput Appl Math 1999;106(1):117–29.

[19] Arnal J, Migallón H, Migallón V, Palomino JA, Penadés J. Parallel relaxed and extrapolated algorithms for computing pagerank. J Supercomput 2014;70:637–48.

[20] Dongarra J, Huss-Lederman S, Otto S, Snir M, Walkel D. MPI: the complete reference. 2nd ed. Cambridge, MA: The MIT Press; 1998.

[21] OpenMP official site, 2008. http://www.openmp.org.

[22] Laboratory for Web Algorithmics. 2002. http://law.di.unimi.it.

[23] Boldi P, Codenotti B, Santini M, Vigna S. Ubicrawler: a scalable fully distributed web crawler. Softw Pract Exp 2004;34:711–26.

[24] Migallón H, Migallón V, Palomino JA, Penadés J. Parallelization strategies for computing pagerank. In: Topping BHV, Adam JM, Pallaés FJ, Bru R, Romero ML, editors. Proceedings of the seventh international conference on engineering computational technology. Stirlingshire, United Kingdom: Civil-Comp Press; 2010. Paper 29. doi: 10.4203/ccp.94.29

Please cite this article as: H. Migallón et al., A heuristic relaxed extrapolated algorithm for accelerating PageRank, Advances in Engineering Software (2016), http://dx.doi.org/10.1016/j.advengsoft.2016.01.024