# The Push Algorithm for Spectral Ranking

Paolo Boldi Sebastiano Vigna

March 29, 2020

**Abstract**

The push algorithm was proposed first by Jeh and Widom [6] in the context of personalized PageRank computations (albeit the name "push algorithm" was actually used by Andersen, Chung and Lang in a subsequent paper [1]). In this note we describe the algorithm at a level of generality that make the computation of the spectral ranking of any nonnegative matrix possible. Actually, the main contribution of this note is that the description is very simple (almost trivial), and it requires only a few elementary linear-algebra computations. Along the way, we give new precise ways of estimating the convergence of the algorithm, and describe some of the contribution of the existing literature, which again turn out to be immediate when recast in our framework.

## 1 Introduction

Let $M$ be a $n \times n$ nonnegative real matrix with entries $m_{xy}$. Without loss of generality, we assume that $\|M\|_1 = 1$; this means that $M$ is *substochastic* (i.e., its row sums are at most one) and that *at least one row has sum one*.[1]

Equivalently, we can think of the arc-weighted graph $G$ *underlying* $M$. The graph has $n$ nodes, and an arc $x \to y$ weighted by $m_{xy}$ if $m_{xy} > 0$. We will frequently switch between the matrix and the graph view, as linear matters are better discussed in terms of $M$, but the algorithms we are interested in are more easily discussed through $G$.

As a guiding example, given a a directed graph $G$ with $n$ nodes, $M$ can be the transition matrix of its *natural walk*[2], whose weights are $m_{xy} = 1/d^+(x)$, where $d^+(x)$ is the outdegree of $x$ (the number of arcs going out of $x$).

We recall that the spectral radius $\rho(M)$ of $M$ coincides with the largest (in modulus) of its eigenvalues, and satisfies[3]

$$\min_i \|M_i\|_1 \leq \rho(M) \leq \max_i \|M_i\|_1 = \|M\|_1 = 1,$$

[1] If this is not the case, just multiply the matrix by the inverse of the maximum row sum. The multiplication does not affect the eigenspaces, but now the matrix satisfies the conditions above. Of course, the values of the damping factor (see further on) have to be adjusted accordingly.

[2] We make no assumptions on $G$, so some nodes might be dangling (i.e., without successors). In that case the corresponding rows of $M$ will be zeroed, so $M$ would be neither stochastic, nor a random walk in a strictly technical sense.

[3] We use row vectors, so the $\ell_1$ norm of a matrix is the maximum of the norm of the *rows*.

where $M_i$ is the $i$-th row of $M$ (the second inequality is always true; the first one only for nonnegative matrices).

Let $\boldsymbol{v}$ be a nonnegative vector satisfying $\|\boldsymbol{v}\|_1 = 1$ (i.e., a distribution) and $\alpha \in [0\,..\,1)$. The *spectral ranking*[4] of $M$ with preference vector $\boldsymbol{v}$ and damping factor $\alpha$ is defined by

$$\boldsymbol{r} = (1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1} = (1-\alpha)\boldsymbol{v}\sum_{k\geq 0}\alpha^k M^k.$$

Note that the $\boldsymbol{r}$ needs not be a distribution, unless $M$ is stochastic.[5] Note also that the linear operator is defined for $\alpha \in [0\,..\,1/\rho(M))$, but usually estimating $\rho(M)$ is very difficult. The value $1/\rho(M)$ can actually be attained by a limiting process which essentially makes the damping disappear [8].

We start from the following trivial observation: while it is very difficult to "guess" which is the spectral ranking $\boldsymbol{r}$ associated to a certain $\boldsymbol{v}$, the inverse problem is trivial: given $\boldsymbol{r}$,

$$\boldsymbol{v} = \frac{1}{1-\alpha}\boldsymbol{r}(1-\alpha M).$$

The resulting preference vector $\boldsymbol{v}$ might not be, of course, a distribution (otherwise we could obtain *any* spectral ranking using a suitable preference vector), but the equation is always true.

The observation is trivial, but its consequences are not. For instance, consider an indicator vector $\boldsymbol{\chi}_x(z) = [x = z]$. If we want to obtain $(1-\alpha)\boldsymbol{\chi}_x$ as spectral ranking, the associated preference vector $\boldsymbol{v}$ has a particularly simple form:

$$\boldsymbol{v} = \frac{1}{1-\alpha}(1-\alpha)\boldsymbol{\chi}_x(1-\alpha M) = \boldsymbol{\chi}_x - \alpha\sum_{x\to y} m_{xy}\boldsymbol{\chi}_y. \tag{1}$$

We remark that in the case of a natural random walk, $m_{xy} = d(x)^{-1}$, which does not depend on $y$ and can be taken out of the summation. Of course, since spectral rankings are linear we can obtain $(1-\alpha)\boldsymbol{\chi}_x$ multiplied by any constant just by multiplying $\boldsymbol{v}$ by the same costant.

## 2 The push algorithm

If the preference vector $\boldsymbol{v}$ is highly concentrated (e.g., an indicator) and $\alpha$ is not too close to one most updates done by linear solvers or iterative methods to compute spectral rankings are useless—either they do not perform any update, or they update nodes whose final value will end up to be below the computational precision.

[4] "Spectral ranking" is an umbrella name for techniques based on eigenvectors and linear maps to rank entities; see [8] for a detailed history of the subject, which was studied already in the late forties.

[5] If $\boldsymbol{M}$ is the natural walk on a graph, $\boldsymbol{r}$ is not exactly PageRank [7], but rather the *pseudorank* [5] associated with $\boldsymbol{v}$ and $\boldsymbol{\alpha}$. The pseudorank is not necessarily a distribution, whereas technically a PageRank vector always is. The distinction is however somehow blurred in the literature, where often pseudoranks are used in place of PageRank vectors. If $\boldsymbol{G}$ has no dangling nodes, the pseudorank is exactly PageRank. Otherwise, there are some differences depending on how dangling nodes are patched [4].

The *push algorithm* uses the concentration of modifications to reduce the computational burden. The fundamental idea appeared first in Jeh and Widom's widely quoted paper [6], albeit the notation somehow obscures the ideas. Berkhin restated the algorithm in a different and more readable form [2]. Andersen, Chung and Lang [1] applied a specialised version of the algorithm on symmetric graphs. All these references apply the idea to PageRank, but the algorithm is actually an algorithm for the steady state of Markov chains with restart [3], and it works even with substochastic matrices, so it should be thought of as an algorithm for spectral ranking with damping.[6]

The basic idea is that of keeping track of vectors $\boldsymbol{p}$ (the current approximation) and $\boldsymbol{r}$ (the *residual*) satisfying

$$\boldsymbol{p} + (1-\alpha)\boldsymbol{r}(1-\alpha M)^{-1} = (1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}$$

Initially, $\boldsymbol{p} = 0$ and $\boldsymbol{r} = \boldsymbol{v}$, which makes the statement trivial, but we will incrementally increase $\boldsymbol{p}$ (and reduce correspondingly $\boldsymbol{r}$).

To this purpose, we will be iteratively *pushing*[7] some node $x$. A push on $x$ adds $(1-\alpha)r_x\boldsymbol{\chi}_x$ to $\boldsymbol{p}$. Since we must keep the invariant true, we now have to update $\boldsymbol{r}$. If we think of $\boldsymbol{r}$ as a preference vector, we are just trying to solve the inverse problem (1): by linearity, if we subtract from $\boldsymbol{r}$

$$r_x\left(\boldsymbol{\chi}_x - \alpha\sum_{x\to y} m_{xy}\boldsymbol{\chi}_y\right),$$

the value $(1-\alpha)\boldsymbol{r}(1-\alpha M)^{-1}$ will decrease exactly by $(1-\alpha)r_x\boldsymbol{\chi}_x$, preserving the invariant.

It is not difficult to see why this choice is good: we zero an entry (the $x$-th) of $\boldsymbol{r}$, and we add small positive quantities to a small (if the graph is sparse) set of entries (those associated with the successors of $x$), increasing the $\ell_1$ norm of $\boldsymbol{p}$ by $(1-\alpha)r_x$, and decreasing at least by the same amount that of $\boldsymbol{r}$ (larger decreases happening on strictly substochastic rows—e.g., dangling nodes). Note that since we do not create negative entries, it is always true that

$$\|\boldsymbol{p}\|_1 + \|\boldsymbol{r}\|_1 \leq 1.$$

Of course, we can easily keep track of the two norms at each update.

The error in the estimate is

$$\left\|(1-\alpha)\boldsymbol{r}(1-\alpha M)^{-1}\right\|_1 = (1-\alpha)\left\|\boldsymbol{r}\sum_{k\geq 0}\alpha^k M^k\right\|_1 \leq (1-\alpha)\|\boldsymbol{r}\|_1\sum_{k\geq 0}\alpha^k\left\|M^k\right\|_1 \leq \|\boldsymbol{r}\|_1.$$

Thus, we can control exactly the *absolute additive error* of the algorithm by controlling the $\ell_1$ norm of the residual.

[6] An implementation of the push algorithm for the computation of PageRank is available as part of the LAW software at `http://law.dsi.unimi.it/`.

[7] The name is taken from [1]—we find it enlightening.

It is important to notice that if $M$ is strictly substochastic it might happen that

$$\left\|(1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}\right\|_1 < 1.$$

If this happens, controlling the $\ell_1$ norm of the residual is actually of little help, as even in the case of natural walks the norm above can be as small as $1-\alpha$. However, since we have the guarantee that $\boldsymbol{p}$ is a nonnegative vector which approximates the spectral ranking *from below*, we can simply use

$$\frac{\|\boldsymbol{r}\|_1}{\|\boldsymbol{p}\|_1} \geq \frac{\|\boldsymbol{r}\|_1}{\left\|(1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}\right\|_1}$$

as a measure of relative precision, as

$$\frac{\left\|(1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}-\boldsymbol{p}\right\|_1}{\left\|(1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}\right\|_1} = \frac{\left\|(1-\alpha)\boldsymbol{r}(1-\alpha M)^{-1}\right\|_1}{\left\|(1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}\right\|_1} \leq \frac{\|\boldsymbol{r}\|_1}{\|\boldsymbol{p}\|_1}.$$

## 2.1 Handling pushes

The order in which pushes are executed can be established in many different ways. Certainly, to guarantee relative error $\varepsilon$ we need only push nodes $v$ such that $r_x > \varepsilon\|\boldsymbol{p}\|_1/n$, as if all nodes fail to satisfy the inequality then $\|\boldsymbol{r}\|_1/\|\boldsymbol{p}\|_1 \leq \varepsilon$.

The obvious approach is that of keeping an indirect priority queue (i.e., a queue in which the priority of every element can be updated at any time) containing the nodes satisfying the criterion above (initially, just the support of $\boldsymbol{v}$) and returning them in order of decreasing $r_x$. Nodes are added to the queue when their residual is larger than $\varepsilon\|\boldsymbol{p}\|_1/n$. Every time a push is performed, the residual of successors of the pushed node are updated and the queue is notified of the changes.

While this generates potentially an $O(\log n)$ cost per arc visited (to adjust the queue), in intended applications the queue is always very small, and pushing larger values leads to a faster decrease of $\|\boldsymbol{r}\|_1$.

An alternative approach is to use a FIFO queue (with the proviso that nodes already in the queue are not enqueued again). In this case, pushes are not necessarily executed in the best possible order, but the queue has constant-time access.

Some preliminary experiments show that the two approaches are complementary, in the sense that in situations where the number of nodes in the queue is relatively small, a priority code reduces significantly the number of pushes, resulting in a faster computation. However, if the queue becomes large (e.g., because the damping factor is close to one), the logarithmic burden at each modification becomes tangible, and using a FIFO queue yields a faster computation in spite of the higher number of pushes.

In any case, to reduce the memory footprint for large graphs it is essential to keep track of the bijection between the set of visited nodes and an identifier assigned incrementally in discovery order. In this way, all vectors involved in the computation can be indexed by discovery order, making their size dependent just on the size of the visited neighbourhood, and not on the size of the graph.

## 2.2 Convergence

There are no published results of convergence for the push algorithm. Andersen, Chung and Lang provide a bound not for convergence to the pseudorank, but rather for convergence to the ratio between the pseudorank and the stationary state of $M$ (which in their case—symmetric graphs—is trivial, as it is proportional to the degree).

In case a priority queue is used to select the nodes to be pushed, when the preference vector is an indicator $\boldsymbol{\chi}_x$ the amount of rank going to $\boldsymbol{p}$ at the first step is exactly $1-\alpha$. In the follow $d(x)$ steps, we will visit either the successors of $x$, whose residual is $\alpha/d(x)$, or some node with a larger residual, due to the prioritization in the queue. As a result, the amount of rank going to $\boldsymbol{p}$ will be at least $\alpha(1-\alpha)$. In general, if $P_x(t)$ is the *path function* of $x$ (i.e., $P_x(t)$ is the number of paths of length at most $t$ starting from $x$), after $P_x(t)$ pushes the $\ell_1$ norm of $\boldsymbol{r}$ will be at most $1-(1-\alpha)\sum_{0\le k\le t}\alpha^k = \alpha^{t+1}$.

## 2.3 Some remarks

**Precomputing spectral rankings.** Another interesting remark[8] is that if during the computation we have to perform a push on a node $x$ *and we happen to know the spectral ranking of* $x$ (i.e., the spectral ranking with preference vector $\boldsymbol{\chi}_x$) we can simply zero $r_x$ and add the spectral ranking of $x$ multiplied by the current value of $r_x$ to $\boldsymbol{p}$. Actually, we could even *never push* $x$ and just add the spectral ranking of $x$ multiplied by $r_x$ to $\boldsymbol{p}$ at the end of the computation.

Let us try to make this observation more general. Consider a set $H$ of vertices whose spectral ranking is known; in other words, for each $x \in H$ the vector

$$\boldsymbol{s}_x = (1-\alpha)\boldsymbol{\chi}_x(1-\alpha M)^{-1}$$

is somehow available. At every step of the algorithm, the invariant equation

$$\boldsymbol{p} + (1-\alpha)\boldsymbol{r}(1-\alpha M)^{-1} = (1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}$$

can be rewritten as follows: let $\boldsymbol{r}'$ be the vector obtained from $\boldsymbol{r}$ after zeroing all entries outside of $H$, and let $\boldsymbol{p}' = \boldsymbol{p} + \sum_{x\in H} r_x \boldsymbol{s}_x$. Then clearly

$$\boldsymbol{p}' + (1-\alpha)\boldsymbol{r}'(1-\alpha M)^{-1} = (1-\alpha)\boldsymbol{v}(1-\alpha M)^{-1}.$$

Note that

$$\|\boldsymbol{r}'\|_1 = \sum_{x\notin H} r_x$$

and

$$\|\boldsymbol{p}'\|_1 = \|\boldsymbol{p}\|_1 + \sum_{x\in H} r_x \cdot \|\boldsymbol{s}_x\|_1.$$

So we can actually execute the push algorithm keeping track of $\boldsymbol{p}$ and $\boldsymbol{r}$ but considering (virtually) to possess $\boldsymbol{p}'$ and $\boldsymbol{r}'$, instead; to this aim, we proceed as follows:

[8] Actually, a translation of Jeh and Widom's approach based on *partial vectors*, which was restated by Berkhin's under the name *hub decompositions* [2]. Both become immediate in our setting.

- we never add nodes in $H$ to the queue;
- for convergence, we consider the norms of $\boldsymbol{p}'$ and $\boldsymbol{r}'$, as computed above;
- at termination, we adjust $\boldsymbol{p}$ obtaining $\boldsymbol{p}'$ explicitly.

Berkhin [2] notes that when computing the spectral ranking of $x$ we can use $x$ as a hub after the first push. That is, after the first push we will never enqueue $x$ again. At the end of the computation, we simply multiply the resulting spectral ranking by $1 + r_x + r_x^2 + \cdots = 1/(1 - r_x)$. In this case, $\|\boldsymbol{p}\|_1$ must be divided by $1 - r_x$ to have a better estimate of the actual norm. Preliminary experiments on web and social graphs show that the reduction of the number of pushes is very marginal, though.

**Patching dangling nodes.** Suppose that, analogously to what is usually done in power-method computations, we may patch dangling nodes. More precisely, suppose that we start from a matrix $M$ that has some zero rows (e.g., the natural walk of a graph $G$ with dangling nodes), and then we obtain a new matrix $P$ (for "patched") by *substituting* each zero row with some distribution $\boldsymbol{u}$, as yet unspecified.

It is known that avoiding at all the patch is equivalent to using $\boldsymbol{u} = \boldsymbol{v}$ [5], modulo a scale factor that is computable starting from the spectral ranking itself. More generally, if $\boldsymbol{u}$ coincides with the distribution that is being used for preference, no patching is needed provided that the final result is normalized.

For the general case (where $\boldsymbol{u}$ may not coincide with $\boldsymbol{v}$), we can adapt the push method described above as follows: we keep track of vectors $\boldsymbol{p}$ and $\boldsymbol{r}$ and of a scalar $\theta$ representing the amount of rank that went through dangling nodes. The equation now is

$$\boldsymbol{p} + (1-\alpha)(\boldsymbol{r} + \theta\boldsymbol{u})(1-\alpha P)^{-1} = (1-\alpha)\boldsymbol{\chi}_x(1-\alpha P)^{-1}$$

When $\boldsymbol{p}$ is increased by $(1-\alpha)r_x\boldsymbol{\chi}_x$, we have to modify $\boldsymbol{r}$ and $\theta$ as follows:

- if $x$ is not dangling, we subtract from $\boldsymbol{r}$ the vector

$$r_x\left(\boldsymbol{\chi}_x - \alpha\sum_{x\to y} m_{xy}\boldsymbol{\chi}_y\right);$$

- if $x$ is dangling, we subtract just

$$r_x\boldsymbol{\chi}_x$$

  and *increase* $\theta$ by $\alpha r_x$.

At every computation step the approximation of the spectral ranking will be given by $\boldsymbol{p}' = \boldsymbol{p} + \theta\boldsymbol{s}$, where $\boldsymbol{s}$ is the spectral ranking of $P$ with preference vector $\boldsymbol{u}$ and damping factor $\alpha$.[9] As on the case of hubs, we should consider $\|\boldsymbol{p}'\|_1 = \|\boldsymbol{p}\|_1 + \theta\|\boldsymbol{s}\|_1$ when establishing convergence.

[9] Of course, $\boldsymbol{s}$ must be precomputed using any standard method. If $\boldsymbol{M}$ is the natural walk of a graph $\boldsymbol{G}$, this is exactly the PageRank vector for $\boldsymbol{G}$ with preference vector $\boldsymbol{u}$.

# References


[1] Reid Andersen, Fan R. K. Chung, and Kevin J. Lang. Using PageRank to locally partition a graph. *Internet Math.*, 4(1):35–64, 2007.

[2] Pavel Berkhin. Bookmark-coloring approach to personalized PageRank computing. *Internet Math.*, 3(1), 2006.

[3] Paolo Boldi, Violetta Lonati, Massimo Santini, and Sebastiano Vigna. Graph fibrations, graph isomorphism, and PageRank. *RAIRO Inform. Théor.*, 40:227–253, 2006.

[4] Paolo Boldi, Roberto Posenato, Massimo Santini, and Sebastiano Vigna. Traps and pitfalls of topic-biased PageRank. In William Aiello, Andrei Broder, Jeannette Janssen, and Evangelos Milios, editors, *WAW 2006. Fourth Workshop on Algorithms and Models for the Web-Graph*, number 4936 in Lecture Notes in Computer Science, pages 107–116. Springer–Verlag, 2008.

[5] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. PageRank: Functional dependencies. *ACM Trans. Inf. Sys.*, 27(4):1–23, 2009.

[6] Glen Jeh and Jennifer Widom. Scaling personalized web search. In *Proc. of the Twelfth International World Wide Web Conference*, pages 271–279. ACM Press, 2003.

[7] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank citation ranking: Bringing order to the web. Technical report, Stanford Digital Library Technologies Project, Stanford University, Stanford, CA, USA, 1998.

[8] Sebastiano Vigna. Spectral ranking, 2009.