Internet Mathematics Vol. 3, No. 1: 41-62

Bookmark-Coloring Algorithm for Personalized PageRank Computing

Pavel Berkhin

Abstract. We introduce a novel bookmark-coloring algorithm (BCA) that computes authority weights over the web pages utilizing the web hyperlink structure. The computed vector (BCV) is similar to the PageRank vector defined for a page-specific teleportation. Meanwhile, BCA is very fast, and BCV is sparse. BCA also has important algebraic properties. If several BCVs corresponding to a set of pages (called *hub*) are known, they can be leveraged in computing arbitrary BCV via a straightforward algebraic process and hub BCVs can be efficiently computed and encoded.

1. Introduction

The PageRank vector (PageRank citation ranking weight) introduced in [Brin and Page 98, Page et al. 98] plays an important role in ranking search engine query results. PageRank vector (PRV) is defined over a directed graph $\boldsymbol{W} = (G, L)$ of web pages G. It represents the hyperlink-based authority index. The edges L can be associated with a $n \times n$ (sparse) adjacency matrix that we denote by the same symbol. Element L_{ij} is equal to 1 when there is a link $i \to j$ and is 0 otherwise. The out-degree of a node i is the number of outgoing links $\deg(i) = \sum_j L_{ij}$. This and other definitions can be generalized to a weighted directed graph in which case $L_{ij} \geq 0$. A *transition* matrix is defined as $P_{ij} = L_{ij}/\deg(i)$

1542-7951/05 $0.50 per page

for pages with $\deg(i) > 0$, which are called *nondangling*. Other rows in P are set to zero. Disregarding this trouble for a while, assume that page authorities p_i are such that each page endorses its out-neighbors with equal fraction of its authority. This means that p is a stationary point of the following transformation

$$p_j^{(k+1)} = \sum_{i \to j} p_i^{(k)}/\deg(i) \quad \text{or} \quad p^{(k+1)} = P^T p^{(k)}. \tag{1.1}$$

Getting back to dangling pages, a simple fix is to add artificial links from them to all other pages on the web. Formally,

$$P^{'} = P + d \cdot v^T, \tag{1.2}$$

where v is a so-called *teleportation* distribution as, for example, uniform distribution $v = e, e = e_n = (1/n, \ldots, 1/n)$ and d is a dangling page indicator, $d_i = \delta_0^{deg(i)}$. Here and below we use a common notation $\delta_a^b = 1$, if $a = b$, and $\delta_a^b = 0$, if $a \neq b$. The modified matrix P' is row-stochastic.

Under the conditions of *strict connectivity* and *aperiodicity* of the graph $\boldsymbol{W}$, the *Perron-Frobenius theorem* guarantees that the simple power iterations process (1.1) starting with any nonnegative initial guess $p^{(0)}$ converges to the eigenvector p of a matrix P^T corresponding to its simple principal (unit) eigenvalue. To achieve strict connectivity, the trick used above for dangling pages is repeated by adding some degree of teleportation to all the pages:

$$P^{''} = cP^{'} + (1-c)E, \quad E = ne \cdot v^T, \quad 0 < c < 1. \tag{1.3}$$

Coefficient c is usually picked around 0.85–0.9. After both modifications, PageRank p is defined as the probability distribution over G ($p_1 + \cdots + p_n = 1, p_i \geq 0$) such that

$$p = P''^T p \quad \text{or} \quad p = cP'^T p + (1-c)n(e^T \cdot p)v. \tag{1.4}$$

Finally, PageRank is frequently introduced via the so-called random surfer model. The *Ergodic theorem* claims that over a long random walk the average number of times a surfer visits a page i converges exactly to p_i.

The significance of modification (1.3) goes beyond purely technical motivations of dealing with dangling pages and achieving strict connectivity. If instead of the uniform teleportation a distribution v that reflects certain preferences is used, this leads to a personalized ranking of search results. In the following, we are dealing with personalized PageRank. Different personalization schemes were suggested [Haveliwala 02, Kamvar et al. 03b]. The most advanced development of Jeh and Widom [Jeh and Widom 02, Jeh and Widom 03] deals with teleportation vectors $v = \delta^{(h)} = (\delta_i^{(h)})$. Any PRV can be approximated by a series of such page-specific PRV $p^{(h)}$.

In this paper we consider a model for page-specific PageRank computing that we call a *bookmark-coloring* model. Consider a diffusion of a coloring substance across the graph. At the first moment, a unit amount of paint is injected into a selected node that we call a *bookmark*. A fraction α of the paint sticks to the node, while the remainder $(1-\alpha)$-fraction flows along out-links. This propagation continues down the graph recursively. The intensity of the accumulated color plays the role of the node authority. We assume that the process is initiated with a fixed bookmark (originating node). We call this a *bookmark-coloring algorithm* (BCA). BCA results in a bookmark-coloring vector (BCV).

We will show that BCV can be considered as an efficient and sparse version of a page-specific PageRank that leads to an almost identical ordering of search results. BCV also has excellent algebraic properties. These properties are analogous to the theory developed in [Jeh and Widom 02].

We start with the review of the related work. In Section 3, we discuss the introduced model, the ways to compute BCV, and the relation between BCV and PRV. In Section 4, we present a more efficient way of finding general BCV by leveraging a set of hub page-specific precomputed BCVs. This brings us to the task of simultaneous computing and encoding of the set of hub specific BCVs. It is solved in Section 5. This section also contains related developments, in particular a so-called loop factor concept. In Section 6, we discuss how BCA can be applied to personalization of web search. Finally, in Section 7, we validate the introduced concepts with computational results.

2. Related Work

Different methods to accelerate the simple power iterations process (1.1) have been suggested, including an *extrapolation* method [Haveliwala et al. 03] (based on a striking result concerning the second eigenvalue [Haveliwala and Kamvar 03]), a *block-structure* method [Kamvar et al. 03b], and an *adaptive* method [Kamvar et al. 03a].

An eigensystem (1.4) can be cast as a linear system

$$p = cP'^{T}p + (1-c)n(e^{T})v = cP^{T}p + k \cdot v. \tag{2.1}$$

Linear systems have a rich theory of accelerated iterative solutions [Axelsson 94, Stewart 99, Golub and Loan 96]. In conjunction with (2.1), they are analyzed in [Arasu et al. 02]. In order for $\|p\| = \|p\,|L_1\| = 1$, scalar k should equal $(1-c) + c \cdot sink(p)$, where $sink(p)$ is a sum of p_i over dangling i. However, the actual value of k is not important, since it only results in a rescaling of the solution. Thus, we can fix $k = 1-c$ (in the absence of dangling pages, this

choice leads to the eigensystem solution). We denote this solution of (2.1) by $p = PR(c, v)$. So defined, PageRank linearly depends on teleportation vector v.

In the rest of this paper, we handle web pages in a very focused (localized) way. In the context of PageRank, the idea to handle pages nonuniformly with different frequencies is exploited in [Abiteboul et al. 03].

Usage of nonuniform teleportation vectors is important for link-based personalization. Haveliwala [Haveliwala 02] advocated topic-sensitive PageRanks. Covered personalization contexts are limited to linear combinations of basic topics (16 were tried). The block-structure method leads to a block-based personalization [Kamvar et al. 03b]. Various other suggestions have been made, going as far as to define a query-specific PageRank [Richardson and Domingos 02]. The most significant progress, in our view, is achieved by Jeh and Widom [Jeh and Widom 02, Jeh and Widom 03]. They propose the most flexible personalization corresponding to page-specific PRVs. Under their approach, users' bookmarks with suitably configured weights naturally induce personalization. The developed theory provides for real scalability. Jeh and Widom show how a small portion of basis PRVs corresponding to so-called *hub* pages H (important selected pages) facilitates computing a general page-specific PRV. Basis hub PRVs can be compressed (encoded). A so-called *hub skeleton*, a relatively small data structure, is instrumental in their decoding. The developed theory is based on a technical apparatus related to inverse P-distance and its modifications. Our development presents a similar theory for BCV computing that is technically straightforward.

Other models besides the random surfer model have been studied. Kleinberg [Kleinberg 99], for example, introduced a framework similar to PageRank computing that utilized a small query-specific subgraph of $\boldsymbol{W}$. It resulted in the algorithm HITS and its variations [Gibson et al. 98, Chakrabarti et al. 98].

3. Bookmark-Coloring Algorithm

In this section we discuss the bookmark-coloring model, its generalizations, BCV computing, and the relation between BCV and PRV.

3.1. Algorithm Description

Consider a linear version of the PageRank algorithm (2.1) with $c = 1 - \alpha$ and teleportation vector $v = \delta^{(b)}$,

$$p = \alpha\delta^{(b)} + (1 - \alpha)P^T p.$$

Dividing both sides by α and setting $y = p/\alpha$, we get the equation

$$y_j = \delta_j^{(b)} + (1-\alpha) \sum_{i \to j \in L} y_i/\deg(i).$$

We want to show that this equation precisely describes the bookmark-coloring model presented in Section 1. The model is straightforward: starting with some node b and a unit amount of paint injected in b, retain an α-portion of color to increment the b-component of authority vector $p^{(b)}$ and distribute the remaining $(1-\alpha)$-portion uniformly among the out-links. According to this model, the total amount of paint y_j that passes through a node j consists of two parts: (1) the original injected unit color $\delta^{(b)}$ (affecting the single page $j = b$) and (2) the color reaching the node j as the result of the propagation along the in-links. In other words, y exactly satisfies the above equation. Only an α portion of paint that passes through a node j actually sticks to a node; hence, the amount of paint actually retained by a node j is equal to $p_j = \alpha \cdot y_j$. We come to an interesting conclusion: mathematically bookmark-coloring vector p is equal to PageRank vector $p = PR(c, \delta^{(b)})$ corresponding to a single-page b teleportation.

We can look at a bookmark coloring vector from a slightly different perspective. Explicitly showing the dependency on b, we can write that

$$p^{(b)} = \alpha\delta^{(b)} + (1-\alpha) \sum_{b \to j \in L} p^{(j)}/\deg(b). \tag{3.1}$$

This equation simply reflects the fact that an α portion of the initially injected unit amount of paint is retained by a node b and that the rest propagates, injecting equal amounts $(1-\alpha)/deg(b)$ into each node accessible from b. This leads to the recursive relation (3.1): to find the result of injecting a unit amount of paint in b, we need to start with the retained amount and then add to it the results of smaller injections into b's out-neighbors that are caused by color propagation. Notice that while conventional equations for PageRank (1.4) and (2.1) relate different components of a single PageRank vector for a single teleportation, equation (3.1) relates many different authority vectors corresponding to different teleportations.

Algorithm 1 presents a conceptual way for finding the bookmark-coloring vector $p = BCA(b, \alpha) = BC(b, w, \alpha)$, with $w = 1$.

The recursion in Algorithm 1 can be stopped when, for example, the amount of color w to be distributed becomes negligible. Some color is lost that does not affect the ranking. The loss happens for two reasons: (a) when the color amount falls below a threshold, it is ignored; (b) a dangling page has nowhere to which to propagate its color.

Algorithm 1. ($p = BC(b, w, \alpha)$ Conceptual BCA.)

Input: A page b, a color amount w, and a retention coefficient α.
Output: Vector p
begin
  set $p_i = \alpha \cdot w$ when $i = b$ and $p_i = 0$ when $i \neq b$
  if *stopping criterion is met or* $\deg(b) = 0$ **then**
    return p
  end
  for *all links* $b \to j \in L$ **do**
    $p = p + BC(j, (1 - \alpha) \cdot w/\deg(b), \alpha)$
  end
  return p
end

Algorithm 2. ($p = BCA(b, \alpha, \epsilon)$ Realistic BCA.)

Input: A bookmark b, a retention coefficient α, and a tolerance threshold ϵ.
Output: BCV p.
begin
  Initialize vector $p = 0$ and queue $Q = \{(b, 1)\}$
  while Q *is not empty* **do**
    pop a queue Q element (i, w)
    $p_i = p_i + \alpha \cdot w$
    if $w < \epsilon$ **then**
      continue
    end
    for *all links* $i \to j \in L$ **do**
      if *pair* $(j, s) \in Q$ **then**
        $s = s + (1 - \alpha) \cdot w/\deg(i)$
      end
      else
        add a new pair $(j, (1 - \alpha) \cdot w/\deg(i)$ to Q
      end
    end
  end
  return p
end

The conceptual BCA is recursive (it would overload the stack). More essentially, different nodes may send their color to the same node. Meanwhile, a node getting color from multiple sources handles it all the same. So, it is the best to wait until as much color as possible reaches a node.

Correspondingly, we prefer to deal with a sequence of transactions, each requesting to distribute a color amount w from a certain node j. We store transactions in a data structure Q that is a *direct-access queue* of pairs (j, w). The direct access means that we can quickly find a pair by its key j. Now we are ready for a computationally feasible implementation of BCA, given in Algorithm 2. We will frequently skip parameter ϵ in our notation. Correspondingly, our formulas, being correct for ideal objects, are in reality only ϵ-approximately correct. They should be understood as such.

3.2. Improvements and Generalizations

The proposed BC Algorithm can be improved and generalized in different ways.

- *Queue Handling.* Consider coupling the FIFO heuristic (pop operation on a queue) with "the largest w goes first" strategy. On an implementation level, a queue can be from time to time partially sorted (split so that the queue's head consists of w above a dynamically updated threshold) or a real priority queue can be employed. Our motivation in selecting a large w for processing is to remove most of the paint as soon as possible to speed up convergence.

- *Caching Link Data.* Though delayed computing (direct queue updates) greatly increases BCA effectiveness, pages actually get reinstated into the queue after being processed. Since getting link data requires I/O, caching (the LRU cache was actually used) greatly increases the speed.

- *Bookmark Sets: Linearity.* While so far we used a single page-specific initial impact, we can actually handle several bookmarks $B = \{b_1, \ldots, b_k\}$ with weights $W = \{w_1, \ldots, w_k\}$ by initializing a queue to $Q = \{(b_1, w_1), \ldots, (b_k, w_k)\}$. We denote a solution by $BCA(B, W, \alpha)$. From linearity we have

$$BCA(B, W, \alpha) = w_1 BCA(b_1, \alpha) + \cdots + w_k BCA(b_k, \alpha).$$

- *Qualified Links.* Both PageRank and BCA frameworks could easily handle links of different weights (e.g., internal-external). While the uniform propagation along all out-links is cheap, assigning to links some weights reflecting their relevance is desirable. The HITS algorithm allows link weighting (ARC) based on a window around anchor-text [Chakrabarti et al. 98].

Such analysis for PageRank is infeasible, since too many pages are involved. Meanwhile, BCA deals with w of various magnitudes. For large w, a link $i \to j$ can be treated nonuniformly, while for a majority of smaller w we may resort to a cheaper default uniform treatment.

3.3. PageRank and BCV

Though, as we have shown in Section 3.1, PRV and BCV satisfy the same mathematical equation

$$p = \alpha\delta^{(b)} + (1-\alpha)P^T p, \tag{3.2}$$

they clearly come from different random surfer and bookmark-coloring models and result from two distinct computational processes. As mathematics suggests, experiments confirm an amazing consistency in the ordering of query search results based on these two authority vectors.

This also raises a question: if the two vectors satisfy the same mathematical equation, what is the difference between them? Were computations to be performed up to infinite precision, we would get the same result. Of course, in practice different algorithms perform differently and lead to similar but quite different outcomes. We want to emphasize that this difference goes far beyond the round-off errors and that the stopping thresholds used in practice are well above machine precision. We now explain what the actual difference is.

Power iterations (1.1) globally treat all the nodes equally and spend most of the time on irrelevant nodes. In comparison, BCA utilizes local propagation—it never touches certain distant nodes. So, BCA is much more focused than PageRank. Another difference is a delay strategy. Updates of existent pairs happen more frequently than adding new pairs to a queue. Updates are cheap, since they do not require access to link data. Therefore, BCA has a built-in effectiveness. For both reasons, BCA is significantly *faster* than the computing of a page-specific PRV.

In addition, BCV has another important property: it is *sparse*. After several iterations page-specific PageRank becomes nonzero on any page that is reachable from b. Meanwhile, a bookmark-coloring vector mostly preserves its sparsity, since the stopping criterion prevents overspill. Affected pages are grouped around the initial b and its successors. BCA grows the vicinity of b adaptively—propagation does not penetrate beyond the threshold-imposed barrier. At the first glance, sparsity means that BCV is only appropriate for ranking a part of the extracted search results. However, relatively low magnitudes of whatever authority vector cannot be trusted in the ordering process (in reality, a relevance mechanism relies on many other features). The locality of BCA that allows it to effectively construct a sparse approximation to PRV is its major advantage over

classic PageRank. Sparsity is very beneficial: compressed sparse objects occupy less memory and can be computed and fetched faster. We also can estimate an L_1 bound between an exact solution and BCV by simply computing the total amount of paint still retained in a queue.

Dangling pages do not present any problem to BCA—the color amount (except the α-fraction) that gets to them is simply abandoned. They do present a problem for PageRank computing, and modification (1.2) technically fixes it. Among many different treatments of dangling pages [Page et al. 98, Eiron et al. 04], this treatment is unique in the sense that it results in a (unscaled) PageRank $p = PR(c, v)$, which is linear with respect to teleportation vector v and coincides with BCV: both satisfy equation (3.2).

In conclusion, BCA efficiently computes a sparse approximation to page-specific PageRank.

4. H-Relative BCA

To cover a variety of personalization contexts in PageRank-based personalization, we need to handle different teleportation vectors. Computing and storing a large number of PageRanks is infeasible. We show that BCA provides a surprising opportunity for effective cooperative computing and compression of a set of BCVs. Deviating from the developments of [Haveliwala 02] that restricted personalization contexts to a linear combination of several topics, Jeh and Widom [Jeh and Widom 02, Jeh and Widom 03] suggested a way to leverage precomputed information in finding arbitrary page-specific PageRanks. In this section we develop a similar framework. This development is based on the excellent algebraic properties of BCA.

4.1. Hub Decomposition

Here we build a theory dealing with *cooperative computing*, *encoding*, and *leveraging* of precomputed BCVs. Assume that a subset of important pages $H \subset G$, called a *hub*, is selected and that $N = |H|$ different basis authority vectors $r^{(h)} = BCA(h, \alpha)$ for $h \in H$ are computed. (Notice that a term *hub* has a different meaning in the context of the HITS algorithm). Issues of cooperative computing and encoding are tackled in the next subsection. Here we answer the question: how does one leverage basis vectors in computing arbitrary $p = p^{(b)} = BCA(b, a)$ when $b \notin H$?

To this end, we suggest constructing a projection

$$p = s_1 \cdot r^{(h_1)} + \cdots + s_N \cdot r^{(h_N)} + u.$$

It represents p as a linear combination of basis vectors corrected by a residual term u that is in some sense "transversal" to the basis vectors. If computing s and u is easier than computing the original p, our goal is achieved.

With this in mind, we introduce a modification of BCA that we call H-relative BCA. It regards H as a *blocking subset*. If a color-propagation process hits a page outside H, H-relative BCA treats it as the regular BCA. If, on the other hand, color w_h reaches $h \in H$, the whole amount w_h is fully retained by h without further propagation (it is *blocked*). Hub H serves as a bank of paint that accumulates color in h-saving accounts to handle it later. We split the result of the blocked propagation into two terms. The first term, u, called the H-relative BCV, is the color amounts propagated to non-hub pages $G \setminus H$ extended to H by zeroes, $u_{|H} = 0$. The second term, $s = \{s_h, h \in H\}$, is the color blocked by H. Mathematicians would call H-relative BCA a modulo H algorithm. Using notation $[u, s] = BCA(b, a\,|H\,) = BC(b, 1, a\,|H\,)$ for the relative BCA, we present it in Algorithm 3.

This algorithm can be implemented via a direct access transaction queue Q similar to the base BCA. We skip the obvious details. The H-relative BCA is more effective than the base BCA. First, by the nature of blocking, lesser nodes are reachable. Thus, a larger H results in a sparser u. Second, the blocked version is also significantly faster: a queue Q is not filled whenever a hub page

Algorithm 3. ($[u, s] = BC(b, w, \alpha\,|H\,)$ Conceptual H-Relative BCA.)

Input: A page b, a color amount w, a retention coefficient α, and a hub H.
Output: H-relative BCV u and blocked s.
begin
  $u = 0, s = 0$
  if $b \in H$ **then**
    $s_b = s_b + w$
  end
  else
    $u_b = \alpha \cdot w$
    if *stopping criterion is met or* $\deg(b) = 0$ **then**
      return $[u, s]$
    end
    for *all links* $i \to j \in L$ **do**
      $[u, s] = [u, s] + BC(j, (1 - \alpha) \cdot w/\deg(b), \alpha\,|H\,)$
    end
  end
  return $[u, s]$
end

h is encountered. The following simple but powerful fact generalizes the basic recursive relation (3.1). It states that the full BCV p can be reconstructed from its H-relative BCA counterparts.

Theorem 4.1. (BCV Hub Decomposition.) *If* $p = p^{(b)} = BCA(b, \alpha)$ *and* $[u, s] = BCA(b, \alpha\,|H)$, *then*

$$p_i = u_i + \sum_{h \in H} r_i^{(h)} \cdot s_h. \tag{4.1}$$

Proof. Notice that each $h \in H$ knows how to distribute color from h—this is what the $r^{(h)}$ are about. Therefore, to reconstruct p all we need to know is how much color gets to each $h \in H$ (this is s) and where the color that has not hit H on its way landed (this is u). Equation (4.1) can be expressed in a vector form

$$p = u + R \cdot s, \tag{4.2}$$

where the matrix $R = \left(r_i^{(h)}\right)$, $i \in G, h \in H$, has basis BCVs as its columns, $\dim(R) = n \times N$, u is zero on H and is sparser than p, $\dim(u) = n$, and $\dim(s) = N$. □

By skipping s_h below certain threshold η, the H-relative BCV can be computed with low accuracy very fast.

4.2. Motivational Example

So far we have provided a formal treatment of BCA and relative BCA. Now we want to explain why this actually works using an analogy with air traffic over a world travel graph. BCA computes authority of each airport given that traffic originates in a particular node. Assume that we start with b corresponding to Houston, TX. Obviously, many passengers go to Chicago and New York. Due to vacation destinations and business affiliations, some traffic lands in Hawaii and Düsseldorf. However, very few people consistently go from Houston to Ukraine or Tibet. From a business standpoint such rare destinations do not matter. They do matter, however, for traffic originating in Moscow or Beijing. PageRank iterates (teleporting to Houston) until all magnitudes including those of O'Hara or JFK are sufficiently converged. Unfortunately, these two important nodes are treated by PageRank absolutely equivalently to any obscure landing field in Ukraine or Tibet! Meanwhile, BCA keeps its focus and produces a sparse solution.

When traveling from Houston to Düsseldorf, travelers most probably land in two major Germany hubs, Munich or Frankfurt. Obviously, in thinking about traffic from Houston, it would be very useful to know in advance and reuse

traffic distributions corresponding to hubs, since these are the airports where most people make their connections. Such modulo hubs computation is what H-relative BCA does.

5. Efficient Hub Computing

As we have shown, knowledge of basis BCVs $r^{(h)}, h \in H$, enables effective computation of $p^{(b)}$ for $b \notin H$ via H-relative BCA. Equation (4.2) assumes that matrix R, whose columns are full basis BCVs, is available. Also, we may simply use linear combinations of R's columns with some coefficients that are externally supplied (e.g., from user profiles). In either case, matrix R is important. However, its effective computing and maintenance present two challenges: (1) computing very many basis BCVs $r^{(h)}$ is not cheap and (2) fetching basis vectors $r^{(h)}$ in memory is I/O bound.

We show in this section how finding a set of N basis vectors $r^{(h)}$ can be shortened by their cooperative computing and how they can be simultaneously compressed! This compressed form can be encoded and accessed during the I/O stage. In effect, at the cost of one extra multiplication by a $N \times N$ matrix, we buy effective computing and encoding. We achieve this by a careful analysis of H-relative BCA.

5.1. Hub Equation

Until now we assumed that $b \notin H$. If $b \in H$, H-relative BCA stops right away with the trivial solution $u = 0, s = \delta^{(b)}$. Consider a simple generalization of the H-relative propagation model: at the first propagation step from $b = h$, we retain the α of the initial $w = 1$ color amount assigning it to the h-component of $r^{(h)}$ and distribute the remaining $1 - \alpha$ part over the out-links. Corresponding out-neighbors may include both H and non-H pages. From this moment on, we revert to the regular H-relative BCA: w propagates over nodes outside H or get blocked by nodes within H. For any $b \in H$ and $i \in G$, we get an analog of (4.1):

$$r_i^{(b)} = \alpha\delta_i^{(b)} + u_i^{(b)} + \sum_{h \in H} r_i^{(h)} \cdot s_h^{(b)}. \tag{5.1}$$

In this formula the δ-term comes from the very first step, $u^{(b)}$ are H-relative vectors, $u_{|H} = 0$, and $s^{(b)}$ are blocked weights. We have come to the following

Theorem 5.1. (Hub Equation.)

$$R = \alpha I_{n \times N} + U + R \cdot S. \tag{5.2}$$

Here, $R = R(H)$ is a matrix of basis BCVs columns, and $\dim(R) = n \times N$; $U = U(H)$ is an H-relative version of $R, \dim(U) = n \times N$, and U is zero on rows $i \in H$; $S = S(H)$ is a blocked component of H-relative BCA, and $\dim(S) = N \times N$.

What are the implications of equation (5.2)? First, U is much sparser than the original R and can be kept in encoded (compressed) form. Second, computing a pair $[U, S]$ is faster than computing a full R, as is the case with any H-relative computing. Solving system (5.2) provides a way to restore R from its cheap and sparse counterparts. Since

$$R \cdot (I_{N\times N} - S) = \alpha I_{n\times N} + U,$$

we have

$$R = (\alpha I_{n\times N} + U) \cdot (I_{N\times N} - S)^{-1} = (\alpha I_{n\times N} + U) \cdot K, \tag{5.3}$$

where $K = (I_{N\times N} - S)^{-1}$ is the $N \times N$ inverse to a diagonally dominant matrix $I_{N\times N} - S$. In practice, the following geometric power series approximation works fine

$$K = I_{N\times N} + S + S^2 + \cdots + S^k + \cdots . \tag{5.4}$$

If we only want to use linear combinations of $r^{(h)}$, we are done. If we desire to find $p = p^{(b)}$ for $b \notin H$, then combining (4.2) and (5.3), we get

$$p = u + R \cdot s = u + (\alpha I_{n\times N} + U) \cdot K \cdot s. \tag{5.5}$$

In other words, instead of fetching in the core longer columns of R as required by (4.2), I/O can be limited to much smaller columns of U (stored in encoded form) as required by (5.5). Matrix K can be kept in core memory. In addition, the precomputing of relative data $[U, S]$ is faster than that of the original R. These improvements come with the price tag of an extra multiplication by K. As always, some accuracy is lost on the way.

5.2. Hub Selection

Now we confront the problem of hub selection. Computing BCV p for a single b would strongly benefit if the hub H contains $h^* = \arg\max_{h,h\neq b} \left\{p_h^{(b)}\right\}$, since this is the page where most color (except of page b itself) has gone and so it is strongly "looped." Averaging over all $b \in G$, we are interested in a page h delivering a maximum to the average of all BCVs $p^{(b)}$. But in view of linearity, the average of all BCVs is the PageRank vector corresponding to a teleportation v that is the average of all $\delta^{(b)}$, and this is simply the uniform teleportation vector. We come to an interesting observation: should we only consider a single-page hub H, we should select it as a page that delivers the maximum to a uniform

Algorithm 4. (Hub Selection.)

$H = 0$
for $m = 1, 2, \ldots$ **do**
  $[u^{(m)}, s^{(m)}] = BCA(B, W, \alpha, |H|)$
  $h^* = \arg\max_{h, h \notin H} \left\{ u_h^{(m)} \right\}$
  $H \leftarrow H \bigcup \{h\}$
end

PageRank. By the same token, for $N = |H| > 1$, we may nominate to the hub the top-N uniform PageRank pages. Two problems undermine this reasoning.

First, not every web page serves uniformly probably as a bookmark. This is quite a generic problem with uniform teleportation. For example, people also do not start new sessions on a uniformly random page and, thus, uniform teleportation is unsatisfactory from a random surfer model point of view as well.

Second, two pages may be perfectly fit in terms of cutting loops, but they may be "correlated" in the sense that they may cut mostly common loops. Including one after another does not lead to a significant improvement. This attribute selection problem is also generic, for example, in the context of variable selection in statistics and data mining.

With the first problem, we may restrict ourselves to a set of, say, 10K "realistic" bookmarks B. We will not elaborate here on the heuristics of how to construct B and will use identical weights, $W = \{1/|B|, \ldots, 1/|B|\}$. Regarding the second problem, consider naïve greedy Algorithm 4. This algorithm takes care of a correlation between already selected pages H and a new incoming page h. Realisticaly, more than a single h may be selected at each step and forward/backward passes may be added.

5.3. Loop Factor and Virtual Hub

Now, when we have investigated a concept of hub, it is natural to suggest that even in the absence of a hub, the originating bookmark b itself can be considered as a perfect single-page hub. This simple device further speeds up BCV computing.

Imagine that during the process of color propagation started at page b, we hit page b again. This means that the whole process mirrors itself, since all the operations done with the originally injected color have to be repeated with the newly arrived color. The reason it does not go on forever is that the process starts with $w = 1$ and hits b again with a smaller w. Since we only propagate w above

a certain threshold, the process eventually stops. In experiments, originating bookmarks actually get many secondary hits.

We want to speed up the process by cutting down on looping. Let us accumulate the entire color amount s that ever returns to b after the very first step instead of propagating it again and again in a mirror fashion. Geometrically this means that we cut the loops coming back to b. This is why we call s a *loop factor*. Let p' represent the vector computed by only propagating the initial color ($w = 1$) and holding (not propagating) the color s that returns to b. In particular, $p'_b = \alpha$. The accumulated amount s needs the same processing as the original $w = 1$. Thus, $p^{(b)} = p' + s \cdot p^{(b)}$ and we get

$$p^{(b)} = p'/(1 - s). \tag{5.6}$$

A loop factor can be found from a simple formula $p_b^{(b)} = \alpha/(1 - s)$. Equation (5.6) is similar to (5.3), where $1/(1 - s)$ plays the role of $(I_{N\times N} - S)^{-1}$.

Now we briefly review another idea. If it is beneficial to cut b-loops, it may be beneficial to cut other loops as well. This is exactly what the hub framework is designed to do. But, a hub is an external entity that is not specific to b. With or without it, we may try to add some b-specific pages to a hub. It is natural to call a so-assembled set a *virtual hub*. Assume for simplicity that there is no external hub. When dealing with the loop factor, we efficiently utilized a trivial hub $H = \{b\}$ to cut b-loops. Going a step further, imagine that some reasonable hub H (depending on b) is assembled, $h_1 = b$. Then, equations similar to (5.1) hold for corresponding basis vectors $r^{(h)}, p^{(b)} = r^{(h_1)} = BCA(b, \alpha)$:

$$r^{(h)} = w^{(h)} + \sum_{q\in H} r^{(q)} \cdot s_q^{(h)}, \qquad h \in H. \tag{5.7}$$

The only difference is that in equation (5.1) each $u^{(b)}$ was zero on H and only the diagonal α-term was nonzero on H, while in (5.7) we combined them together, since H actually evolves during the computation and more than an α-fraction of color could be accumulated on hub pages. This means that a BCV $p^{(b)} = BCA(b, \alpha)$ can be found as the first column in the linear system (5.7). Here is how it works. We start propagating from b as usual, blocking only b-color. At some moment we add a new h to H and from now on we block also h-color. The larger H grows, the more color is blocked (more loops are cut). The nontrivial questions are (1) why all this would be faster than the regular BCA and (2) how to assemble a virtual hub H.

The first question looks puzzling. To find a single $p^{(b)}$, we suggest finding N different $w^{(h)}$ plus S! However, things are not as gloomy. Only for the first $h_1 = b$ relative BCA pair $[w^{(b)}, s^{(b)}]$ has to be found with the input tolerance ϵ. The

ϵ = 1.e-10			ϵ = 1.e-8		
Time	Support	Error	Time	Support	Error
120	945381	2.2e-6	17	184416	4.3e-5
80	645723	1.17e-6	8	118195	2.12e-5
128	93565	1.61e-6	14	244735	2.83e-5
93	811069	7.77e-7	12	195094	1.45e-5
37	340027	4.04e-7	3	34010	5.50e-6
189	1299458	1.82e-6	25	378099	3.93e-5
198	1306594	1.77e-6	30	517603	6.94e-5
52	670617	8.30e-7	5	111380	1.02e-5
38	331877	2.26e-6	5	307651	1.74e-4
23	356062	1.42e-6	3	310175	9.60e-5
40	336028	2.45e-6	5	307690	1.91e-4
68	573814	1.24e-6	10	141057	6.38e-5
66	547160	1.27e-6	8	107618	5.38e-5
32	229158	4.73e-7	9	85226	3.69e-5
56	587740	1.15e-6	6	152512	7.78e-5
204	1099815	1.79e-6	33	444004	1.20e-4
95	821543	1.13e-6	11	203706	1.75e-5

Table 1. BCA performance.

rest are needed with lesser accuracy. This, indeed, is how the stopping criterion works in the absence of any blocking. Here we simply delay some computing by cutting certain loops with the intent to resolve the introduced uncertainty later by solving system (5.7). Moreover, important optimizations such as LRU cache are uniformly applied to the whole process.

Getting to the second question, different strategies of picking H can be contemplated. We simply assembled statistics about the first 10,000 pops of a queue Q and nominated the top frequented pages with frequencies above some threshold as a virtual hub H. Different N from 2 to 60 have been tried. Running time actually decreases, but accuracy may deteriorate in comparison with the vanilla BCA.

6. Application to Personalization

In this section we discuss application of BCA to a personalized web search. We would like to start with the clear acknowledgement of the fact that different meaningful approaches to search personalization coexist. One approach, for example, starts with the concept of broad "personalizable" queries that have multiple meanings. We may try guessing a particular meaning from a user profile or from a current session. Being clearly a personalization, this approach has

many advantages, but it also has problems, such as (a) it is limited to special queries, (b) it partially intersects with the query disambiguation process, (c) it relies on a user profile, which is hard to mine, and (d) it can actually shift a focus contrary to a user's intent.

In this paper we think of personalization as a process orthogonal to the business of query disambiguation. We also do not analyze user profiles. We simply assume that some important bookmark pages are either mined from user profile data or explicitly specified by a user via some appropriate UI. Along with [Jeh and Widom 02], we concentrate on the problem of utilizing this user-specific data. Notice, in this regard, that different topics as well as nontopical user attributes such as gender or preferences can be well represented by sets of bookmarks. This makes our approach very flexible. For example, it allows personalizing queries not through their extensions, but through certain user-specific contexts, i.e., a query "free trade" may be personalized by a user's political orientation, and a query "history" may be personalized by a user's interest in the topic "computers and algorithms." The suggested bookmark-based personalization process consists of the following steps:

1. We choose a hub web subset H (see Section 5.2). We experimented with $N = |H|$ equal to 10^3–10^4 pages.

2. For each h relative $[u(h), s(h)] = BCA(h, \alpha\,|H\,)$ are computed as described in Section 5.1, full matrices U and S are assembled, and U is stored on disk in a compressed encoded form.

3. A particular user model (we used explicit nomination) supplies some bookmarks $B = \{b_1, \ldots, b_k\}$ and weights $W = \{w_1, \ldots, w_k\}$.

4. The H-relative algorithm is run: $[u, s] = BCA(B, W, \alpha\,|H\,)$. It uses a simple modification, described in Section 3.2, in which a queue Q is initiated with several bookmarks and weights. Small coefficients s_h are pruned.

5. The vector p is generated according to formula (5.5), which requires fetching sparse columns of matrix U and performing the multiplication by $K \cdot s$. Vector p is used as a ranking feature.

Introduced personalization has many advantages: (1) User can try different bookmarks on-line. Potentially, a user can play with a variety of personas. Linearity allows for their blending. (2) Computed objects u, s are reusable (they are, so to speak, "personalization cookies"). In principle, such sparse objects can be stored on a user's desktop, which may have an advantage from privacy point perspective. (3) Teleportation-specific BCV constitutes only a single relevancy

Support	Pops	Gets	Adds	Saving in %
150418	418460	262144	5947802	93.00
325849	784809	425984	7357943	89.30
118333	332889	196608	4811865	93.10
63050	189842	98304	3605833	94.70
450889	1323793	851968	15946446	91.70
128595	362638	229376	3050021	88.10
563	11885	<32000	<32000	——
116807	490712	360448	8969216	94.50

Table 2. Queue optimization.

feature. It can be blended (rank aggregation) with overall search ranking in many meaningful fashions. (4) The whole business of query disambiguation is left aside, and therefore, we deal with a well-confined component that may be combined with any other personalization processes.

7. Numerical Experiments

We would like to reflect first that BCA is much faster than page-specific PageRank. For example, on the AltaVista Connectivity Server with a web graph of $n \approx 1.1 \cdot 10^9$ pages and approximately $6.4 \cdot 10^9$ hyperlinks, it takes around two hours to run simple power iterations for page-specific PageRank, optimized and distributed over eight machines, to achieve between the iterations an accuracy of 1.e-6 in the L_1-norm. It takes around three minutes to run optimized BCA on a single machine with $\epsilon =$ 1.e-9. Numerical study of BCA on a large web data is underway and will be reported separately. In this section we provide results related to testing different flavors of BCA on a small subgraph obtained from core 4500 ODP pages corresponding to several broad topics via depth-five crawling. The resulting graph of 25M pages was pruned to 3,131,099 pages by several rounds of deleting the dangling pages.

We start presenting in Table 1 the results for the vanilla (nonrelative, no caching, no queue optimization) BCA: for several randomly chosen pages we computed BCV with $\epsilon =$ 1.e-10 and $\epsilon =$ 1.e-8 and $\alpha = 0.1$. We report run time, support (number on nonzero pages), and error defined to be the maximum deviate from a page-specific PRV computed with L_1 accuracy 1.e-9. This table demonstrates that BCA is fast and that the resulting BCV is sparse.

To study the effects of queue Q optimization, we assembled statistics related to the usage of a directly accessed, periodically partially sorted queue (see Section 3.2). Our findings for some randomly selected bookmarks are presented in

TIME				Hit Ratio in %		
No Cache	50K	100K	200K	50K	100K	200K
453	114	41	40	80.54	92.04	92.04
641	539	119	73	26.95	89.28	93.30
467	29	38	27	95.09	95.09	95.09
31	8	10	8	94.03	94.03	94.03
3526	3169	1465	183	11.82	63.87	93.80
146	17	18	17	94.65	94.65	94.65
2	1	2	1	98.19	98.19	98.19
5163	864	134	134	82.95	95.25	95.25

Table 3. Effects of LRU caching.

Table 2. It shows results for BCV computing with $\alpha = 0.1$ and $\epsilon =$ 1.e-7. A loop factor was used. The Support column contains the number of nonzero pages. The Pops column contains the number of queue pops (that are main elements of the BCA iterative process). The Gets column is equal to the number of actual different pages for which link data has been requested from a connectivity server. There are two reasons for the number of gets to be smaller than the number of pops: (1) after some pops we have w below a threshold, and so link information for further propagation is not requested; (2) one and the same page sometimes gets reinstated in queue, each time resulting in an extra pop. The Adds column shows the number of formal adds to a queue; some of them are handled as updates (delayed computing) to currently present transactions, and some are actual adds (pops). The Saving column is equal to the percent of formal adds that were resolved as updates rather than as actual adds. We see that queue optimization is very effective.

Next, we demonstrate the results of LRU caching of link data. With a cache of size 50K, 100K, and 200K, we get improvements presented in Table 3. We used $\alpha = 0.1$ and $\epsilon =$ 1.e-8. In the No Cache column we give run times when no caching was artificially enforced. The next columns present run times and hit ratios (percentages of times when a requested link data resides in the cache). We see that caching leads to a significant improvement.

To investigate the difference between full and H-relative BCA, we used the same pages as in Table 1 and computed full and relative versions of BCV with $\alpha = 0.1$ and $\epsilon =$ 1.e-10. Parameter η introduced at the end of Section 4.1 was set equal to ϵ. This allows corrections in (4.1) only when $s_h > \eta$. $N = 1000$ was used. Run times for full and relative BCA versions, support sizes, and maximum difference between the full and corrected relative versions of corresponding BCVs are presented in Table 4. On average, H-relative BCA is 5.4 times faster than full regular BCA. It also generates 6.5 times sparser vectors.

ϵ = 1.e-10				
BCA		H-rel. BCA		
Time	Support	Time	Support	Diff.
120	945381	18	170336	1.36e-5
80	645723	2	37473	7.22e-06
128	993565	54	532353	4.81e-06
93	811069	62	588109	8.02e-07
37	340027	26	240048	2.04e-07
189	1299458	33	477225	2.57e-05
198	1306594	10	180824	3.98e-05
52	670617	0	17	5.49e-06
38	331877	0	238	4.59e-06
23	356062	1	959	3.75e-05
40	336028	0	224	2.35e-06
68	573814	23	307307	2.88e-05
66	547160	29	319253	1.20e-05
32	229158	5	45458	1.91e-05
56	587740	15	242261	3.09e-05
204	1099815	1	12096	9.24e-05
95	821543	10	158098	1.43e-05

Table 4. Comparison of BCA and H-relative BCA.

Convergence of a power series (5.4) to K is very fast. In the first eight iterations we get for elements of a matrix term S^k the following maximum absolute values $M(k)$: $M(04)$ = 6.6e-1, $M(08)$ = 4.3e-01, $M(16)$ = 1.8e-1, $M(32)$ = 3.4e-2, $M(64)$ = 1.2e-3, $M(128)$= 1.4e-6, $M(256)$ = 1.9e-12.

Acknowledgements. We thank Qi Lu for bringing our attention to the problem, Byron Dom and Farzin Maghoul for discussions, Josh Coalson for helping with numerical implementation, and Michael Potts for careful editing.

References

[Abiteboul et al. 03] Serge Abiteboul, Mihai Preda, and Gregory Cobena. "Adaptive On-Line Page Importance Computation." In *Proceedings of the 12th International Conference on World Wide Web*, pp. 280–290. New York: ACM Press, 2003.

[Arasu et al. 02] Arvind Arasu, Jasmine Novak, Andrew Tomkins, and John Tomlin. "PageRank Computation and the Structure of the Web: Experiments and Algorithms." In *The Eleventh International World Wide Web Conference Poster Proceedings*, 2002. Available from World Wide Web (http://www2002.org/CDROM/poster/173.pdf).

[Axelsson 94] Owe Axelsson. *Iterative Solution Methods*. New York: Cambridge University Press, 1994.

[Brin and Page 98] Sergey Brin and Lawrence Page. "The Anatomy of a Large-Scale Hypertextual Web Search Engine." *Computer Networks and ISDN Systems* 33 (1998), 107–117.

[Chakrabarti et al. 98] Soumen Chakrabarti, Byron Dom, Prabhakar Raghavan, Sridhar Rajagopalan, David Gibson, and Jon Kleinberg. "Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text." In *Proceedings of the Seventh International Conference on World Wide Web*, 1998. Available from World Wide Web (http://theory.stanford.edu/people/raghavan/www7/181.html).

[Eiron et al. 04] Nadav Eiron, Kevin McCurley, and John Tomlin. "Ranking the Web Frontier." In *Proceedings of the 13th International Conference on World Wide Web*, pp. 309–318. New York: ACM Press, 2004.

[Gibson et al. 98] David Gibson, Jon Kleinberg, and Prabhakar Raghavan. "Inferring Web Communities from Link Topology." In *Proceedings of the Ninth ACM Conference on Hypertext and Hypermedia*, pp. 225–234. New York: ACM Press, 1998.

[Golub and Loan 96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*, Third edition. Baltimore: Johns Hopkins University Press, 1996.

[Haveliwala 02] Taher Haveliwala. "Topic-Sensitive PageRank." In *Proceedings of the 11th International Conference on World Wide Web*, pp. 517–526. New York: ACM Press, 2002.

[Haveliwala and Kamvar 03] Taher Haveliwala and Sepandar Kamvar. "The Second Eigenvalue of the Google Matrix." Technical report, Stanford University, 2003.

[Haveliwala et al. 03] Taher Haveliwala, Sepandar Kamvar, Dan Klein, Chris Manning, and Gene Golub. "Computing PageRank using Power Extrapolation." Technical report, Stanford University, 2003.

[Jeh and Widom 02] Glen Jeh and Jennifer Widom. "Scaling Personalized Web Search." Technical report, Stanford University, 2002.

[Jeh and Widom 03] Glen Jeh and Jennifer Widom. "Scaling Personalized Web Search." In *Proceedings of the 12th International Conference on World Wide Web*, pp. 271–279. New York: ACM Press, 2003.

[Kamvar et al. 03a] Sepandar Kamvar, Taher Haveliwala, and Gene Golub. "Adaptive Methods for the Computation of PageRank." Technical report, Stanford University, 2003. Available from World Wide Web (http://citeseer.ist.psu.edu/kamvar03adaptive.html).

[Kamvar et al. 03b] Sepandar Kamvar, Taher Haveliwala, Christopher Manning, and Gene Golub. "Exploiting the Block Structure of the Web for Computing PageRank." Technical report, Stanford University, 2003. Available from World Wide Web (http://citeseer.ist.psu.edu/kamvar03exploiting.html).

[Kleinberg 99] Jon Kleinberg. "Authoritative Sources in a Hyperlinked Environment." *Journal of the ACM* 46:5 (1999), 604–632.

[Page et al. 98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. "The PageRank Citation Ranking: Bringing Order to the Web." Technical report, Stanford University, 1998.

[Richardson and Domingos 02] Mathew Richardson and Pedro Domingos. "The Intelligent Surfer: Probabilistic Combination of Link and Content Information in PageRank." In *Advances in Neural Information Processing Systems 14*, pp. 1441–1448. Cambridge, MA: MIT Press, 2002.

[Stewart 99] William J. Stewart. "Numerical Methods for Computing Stationary Distribution of Finite Irreducible Markov Chains." Chapter 3 in *Advances in Computational Probability*, edited by Winfried Grassmann. Dordrecht: Kluwer Academic Publishers, 1999.

Pavel Berkhin, Yahoo!, 701 First Avenue, Sunnyvale, CA 94089 (pberkhin@yahoo-inc.com)

Received March 25, 2005; accepted December 16, 2005.