MARKOVIAN PAGE RANKING DISTRIBUTIONS: SOME THEORY AND SIMULATIONS

L.A. BREYER

Abstract. We discuss some simple theoretical properties of web page ranking distributions, including PageRank (Brin and Page, 1998), which are based on uniformly ergodic Markov chains. We also propose a modification of PageRank whith reduces the bias for older documents, and discuss details of our simulation programs.

1. Introduction

The purpose of this note is to report on several simulation experiments involving Markov chains defined upon a subset of the World Wide Web. The underlying dataset consists of $916,428$ publicly available web documents provided by the Internet search company Google Inc., as part of its First Annual Programming Contest.

At the time of writing, Google's full repository, which is the largest of all public search engines, references some three billion web pages. Thus the donated dataset represents one three thousandth of the publicly indexed web, which itself is a subset of the full World Wide Web. Estimates of the true size of the Web vary wildly; moreover a recent study (Labovitz et al, 2002) has shown that at least 5% of the Web (the so-called dark address space) is physically unreachable, depending on one's Internet Service Provider. The dataset iself requires some 1.5 Gb of storage space in compressed form.

We shall present here the results of running several Markov chains on the link graph G associated with this dataset, so as to obtain steady state distributions. A full statistical analysis is beyond the scope of this document. Due to the size and format of the data, building the graph in reasonable time on a small machine with

Date: April, 2002.

1991 *Mathematics Subject Classification.* Primary 60J, Secondary 60F.

Key words and phrases. Search engines, Bayes Theorem, Information Retrieval, PageRank, Markov chains, Resolvents.

memory constraints is nontrivial. Several sections of this report are concerned with explaining some of the design decisions, and what can be expected from distributed simulations. The source code for the programs is freely available from the author, but the dataset should be obtained from Google Inc.

The major theoretical motivation for running Markov chains on the link graph G is that the steady state distributions π of these chains can be used to obtain document rankings for Information Retrieval. This approach, which differs substantially from classical probabilistic IR (Sparck Jones et al., 1998), was pioneered by Brin and Page (1998), who used the following formula for the *PageRank*, $\pi_{\mathrm{PR}}(x)$, of a document $x \in G$. Writing $x \to y$ to denote the existence of a direct link from document x to document y, this is defined as

$$\pi_{\mathrm{PR}}(x) = \epsilon / |G| + (1-\epsilon) \sum_{y \to x} \pi_{\mathrm{PR}}(y) / \|y\|, \quad \|y\| = \#\{z : y \to z\}, \tag{1}$$

where $|G|$ denotes the size of G.

Any two documents x and z can be ranked *a priori* according to the total order on G given by $x \geq z$ if and only if $\pi_{\mathrm{PR}}(x) \geq \pi_{\mathrm{PR}}(z)$. It is remarkable that such a simple ranking gives rise to a demonstrably improved Information Retrieval system, and other authors have quickly proposed alternative rankings based on Markov chains (Kleinberg, 1999; Borodin et al, 2001).

The general strategy consists in defining a ranking $\pi(x)$ as the steady state distribution of a Markov chain X_0, X_1, X_2, $\ldots$, whose transition probabilities $P(x,y) = \mathbb{P}(X_{t+1} = y \,|\, X_t = x)$ satisfy

$$\pi(x) = \sum_y \pi(y) P(y,x), \quad x, y \in G. \tag{2}$$

In the special case of PageRank, we have

$$P_{\mathrm{PR}}(x,y) = \epsilon \mu_{\mathrm{unif}}(y) + \frac{(1-\epsilon)}{\|x\|} \sum_{x \to z} \delta_z(y), \tag{3}$$

where $\mu_{\mathrm{unif}}(x) = |G|^{-1}$ is the uniform distribution on G, δ_z is the point mass at $z \in G$, and ϵ is a parameter $0 < \epsilon < 1$. Thus the Markov chain dynamics mimic an "easily bored web surfer": given $X_t = x$, with probability $(1-\epsilon)$ we choose X_{t+1} randomly among all outgoing links from x, while with probability ϵ we choose X_t uniformly among all documents $x \in G$.

One difficulty in using a definition such as (2) alone is that there may be several distinct solutions π, all of which are probability measures. Moreover, some *transient* subsets S of G may not be ranked at all (i.e. $\pi(x) = 0$ for $x \in S$). The existence and multiplicity depends on the actual geometry of the link graph G, which is not directly under control, and in fact must reflect an ever changing World Wide Web. The algorithms of Kleinberg (1999) and Borodin et al. (2001) suffer from this multiplicity, for carefully constructed graphs G. Interestingly, *PageRank* does not, and this is due to the strong minorisation condition (6). In fact, $\pi_{\mathrm{PR}}(x) \geq \epsilon / |G|$ for all $x \in G$.

The output of a search engine such as Google's has a simple Bayesian interpretation as a posterior distribution, which we now illustrate. Take Ω as the set of all valid query tokens (i.e. words), and write $\omega \in x$ if the document x contains the token ω. Valid tokens can be single words, or could conceivably be full sentences. We want to model a query $U \in \Omega$ for an unknown document $Y \in G$. Bayes' formula states

$$\begin{aligned} \mathbb{P}(Y = x \,|\, U = \omega) &= \mathbb{P}(U = \omega \,|\, Y = x)\mathbb{P}(Y = x)/\mathbb{P}(U = \omega), \\ &\propto L(\omega, x)\tilde{\pi}(x), \end{aligned} \tag{4}$$

where $L(\omega, x)$ is the *likelihood* of the query being ω, given the document is x, and $\tilde{\pi}(x)$ is the *a priori* probability of seeking document x.

Perhaps the simplest model of this type is given by $L(\omega, x) = |x|^{-1} 1_{\{\omega \in x\}}$, where $|x|$ is the number of distinct valid query tokens ω contained in x. The interpretation is that all queries resulting in document x are equally likely, and all documents sought must contain the query token. If we take $\tilde{\pi}(x) = Z^{-1} |x| \, \pi(x)$, where Z is the appropriate normalising constant, and π is an *a priori* distribution on G such as PageRank, then we see that (4) models exactly the output of the search engine (which lists documents containing ω in order of decreasing probability), for by our definitions we can write

$$\mathbb{P}(Y = x \,|\, U = \omega) = 1_{\{\omega \in x\}} \pi(x)/\nu(\omega), \qquad \nu(\omega) = \sum_x |x| \, \pi(x). \tag{5}$$

In Section 4, we introduce a modification π_D of π_{PR} which penalizes documents according to their age, or last date of modification. This follows the rationale that

older referenced documents are less likely to offer timely information, and more likely to no longer be directly accessible on the Web.

2. Simple properties of PageRank

In this section we discuss the family of page ranking distributions based upon (2), of which (1) is a special case. We begin by solving these equations.

For page ranking schemes π which are based upon a Markov chain such as (2), the calculation of π is particularly simple if the transition probabilities $P(x, y)$ are in the form

$$P(x, y) = \epsilon\mu(y) + (1 - \epsilon)Q(x, y), \quad 0 < \epsilon < 1. \tag{6}$$

Note that the *PageRank* scheme (3) is of this form. It is well known that this decomposition of P implies that the Markov chain X_t is *uniformly ergodic* (Meyn and Tweedie, 1993), so that the powers of the matrix P must converge to a *unique* distribution π geometrically fast: $\|P^n(x, \cdot) - \pi(\cdot)\| \leq C\epsilon^n$, where C is a constant and $\|\cdot\|$ is total variation (L^1) norm, $x \in G$ being arbitrary.

More recently, work on perfect simulation with Markov chains (Wilson, 2000; Breyer and Roberts, 2001) has uncovered a representation formula which permits a solution of (2) directly in terms of (6). Due to its importance in this report, we state it as a theorem.

Theorem 1. *Let π be a probability distribution satisfying* (2)*, where P is of the form* (6)*. Then for every $0 < \epsilon < 1$,*

$$\pi(y) = \epsilon \sum_{k=0}^{\infty} (1 - \epsilon)^k \sum_{x \in G} \mu(x) Q^k(x, y), \tag{7}$$

Proof. See Breyer and Roberts (2001), Appendix. □

This solution can be used to sample directly from π, see Section 6. Here we want to concentrate on the special case of *PageRank*, so we state:

Corollary 1. *Let $\pi_{PR(\epsilon)}$ denote the PageRank distribution solving* (1) *uniquely, and let*

$$Q_{PR}(x, y) = \begin{cases} \frac{1}{\|x\|} \sum_{x \to z} \delta_z(y) & \text{if } \|x\| > 0, \\ \delta_x(y) & \text{otherwise.} \end{cases}$$

If μ_{unif} is the uniform distribution on G and $0 < \epsilon < 1$, then

$$\pi_{PR(\epsilon)}(y) = \epsilon \sum_{x \in G} \mu_{unif}(x) \sum_{k=0}^{\infty} (1-\epsilon)^k Q_{PR}^k(x,y), \tag{8}$$

where Q_{PR}^k is the k-fold matrix product of Q_{PR}, the transition matrix of a Markov chain on G which follows outward links randomly.

We can now ask what happens if we change ϵ and/or μ. In both cases, we strictly speaking leave the *PageRank* family π_{PR}, but stay within the broader class of solutions to (2) and (6).

Consider first the effect of a change from μ to μ'. Since μ is a probability distribution on G, there always exists a set of weights (Radon-Nikodym derivative) $u(x)$, $x \in G$ such that $\mu'(x) = u(x)\mu(x)$.

Proposition 1. *Let $u(x)$ be a series of weights with $u(x) > 0$ for all $x \in G$ and $\sum_x u(x)\mu(x) = 1$. Suppose π solves (2) and (6), and put $\pi'(x) = u(x)\pi(x)$. Then π' solves (2) where P is replaced by*

$$P^u(x,y) = \epsilon u(x)\mu(x) + (1-\epsilon)Q^u(x,y), \qquad Q^u(x,y) = Q(x,y)u(y)/u(x). \tag{9}$$

Proof. By Theorem 1 and the definition of Q^u, we have

$$\begin{aligned} u(y)\pi(y) &= \epsilon \sum_{k=0}^{\infty} (1-\epsilon)^k \sum_{x \in G} \mu(x) Q^k(x,y) u(y), \\ &= \epsilon \sum_{k=0}^{\infty} (1-\epsilon)^k \sum_{x \in G} \mu(x) u(x) (Q^u)^k(x,y), \end{aligned}$$

and applying Theorem 1 in reverse yields the claim. □

Note that P^u is not necessarily a *transition probability* matrix for a given set of weights u, but the fundamental representation formula always holds. Since G is finite, by renormalising P^u we can insist that $\sum_y P^u(x,y) \leq 1$ for all x, but the corresponding probabilistic interpretation is of a "bored web surfer who is likely to turn off the computer in disgust".

As described in the introduction, the main motivation for simulating Markov chains on the link graph G is to obtain the steady state distribution as a Bayesian *a priori* measure for the importance of indexed documents. From (5), if we replace $\pi(x)$ with the weighted version $u(x)\pi(x)$, the effect is the same as keeping $\pi(x)$ and changing the likelihood $L(\omega, x)$ to $L'(\omega, x) \propto L(\omega, x)u(x)$.

This has the following practical implication: for a search engine based upon (2) and (6), simple modifications of the list of returned documents are easily accomplished without changing the overall architecture, only the Markov chain back end. For example, the function

$$u_C(x) = \begin{cases} \eta & \text{for } x \in C \subset G, \text{ and } \eta \approx 0 \\ 1 & \text{for } x \notin C \end{cases}$$

allows all documents within the set C to be temporarily filtered out without removing them from the database (this could also be accomplished by an output filter of course). As another, concrete example, $u(x)$ could be a score based on the number of postscript files referenced in document x. The effect is equivalent to sorting the full output of the search engine by this score, without affecting the existing rankings in cases of a tie. Some types of meta search engines can therefore be emulated.

We now leave the topic of weighting functions u, and briefly consider the effect of changing ϵ in (6) and (7). Since $Q(x, y)$ itself is the transition matrix for a Markov chain (which "never gets bored"), it has a *resolvent* R_α (Meyn and Tweedie, 1993; Revuz, 1984), which is a family of matrices such that

$$(10)\quad R_\alpha(x,y) \equiv \sum_{k=0}^{\infty} \alpha^k Q^k(x,y), \qquad \alpha R_\alpha = \beta R_\beta + (\alpha-\beta) R_\alpha R_\beta, \quad 1 > \alpha > \beta > 0,$$

where the second equation above is known as the *resolvent equation*. Comparing with (7), it is clear that $\pi_\epsilon = \epsilon\mu R_{1-\epsilon}$ where we write π_ϵ to denote explicitly the dependence of π on ϵ in (7). From the resolvent equation, we deduce the identities

$$\delta(1-\epsilon)\pi_\epsilon = \epsilon(1-\delta)\pi_\delta + (\delta-\epsilon)\Pi_{\epsilon,\delta}, \qquad \Pi_{\epsilon,\epsilon} = \pi_\epsilon - \epsilon(1-\epsilon)\frac{d\pi_\epsilon}{d\epsilon},$$

where we define $\Pi_{\epsilon,\delta} \equiv \delta\pi_\epsilon R_{1-\delta} \equiv \epsilon\pi_\delta R_{1-\epsilon}$, i.e. a solution of (2) with μ in (6) replaced by π_δ. The measure $\Pi_{\epsilon,\delta}$ can, by this definition, be associated with a "bored web surfer who is aware of the search engine and uses it exclusively", with the implication that the above equations describe the extent of the bias introduced in this simple model by the existence of the search engine.

From the resolvent equation, we deduced an analytic expression to link π_ϵ with π_δ. However, this isn't particularly convenient for obtaining a direct qualitative view of the *PageRank* family π_ϵ. In Figures 1, 2 and 3, we plot π_δ versus π_ϵ for several choices of ϵ and δ with the given dataset. In the plots, each point represents

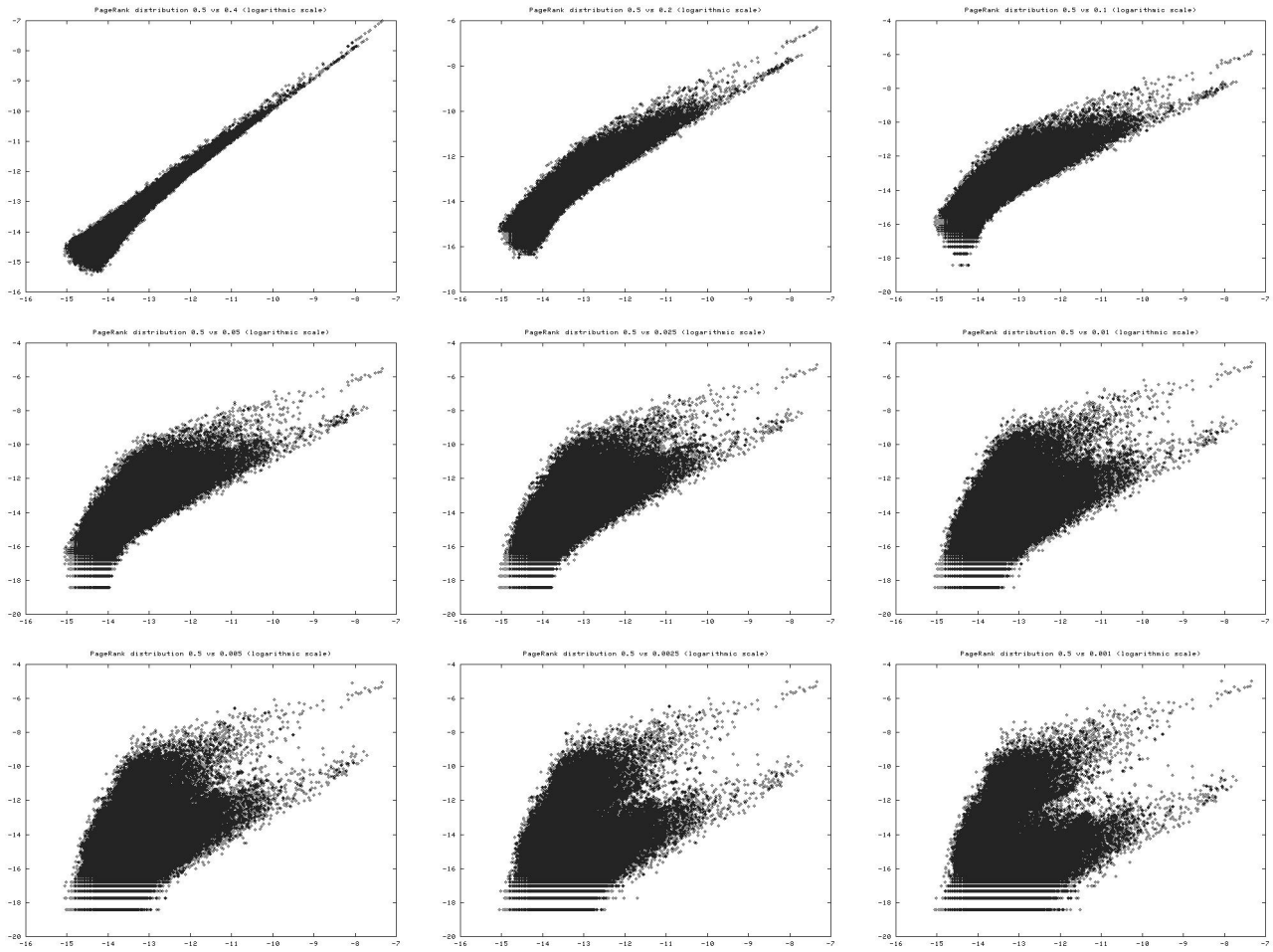

FIGURE 1. Each plot shows $\log \pi_\delta$ (vertical coordinate) against $\log \pi_\epsilon$ (horizontal coordinate) for $\delta < \epsilon = 0.5$

a document in the dataset, with the horizontal coordinate given by $\log \pi_\epsilon(x)$, and the vertical coordinate given by $\log \pi_\delta(x)$. If a point is above the diagonal, it means that $\pi_\delta(x) > \pi_\epsilon(x)$, and conversely.

3. More on perturbations

In this section, we discuss simple examples of additive perturbations to the transition dynamics Q, by utilising the properties of the resolvent, introduced in (10). The topic of perturbations seems natural as a way to analyse the robustness of

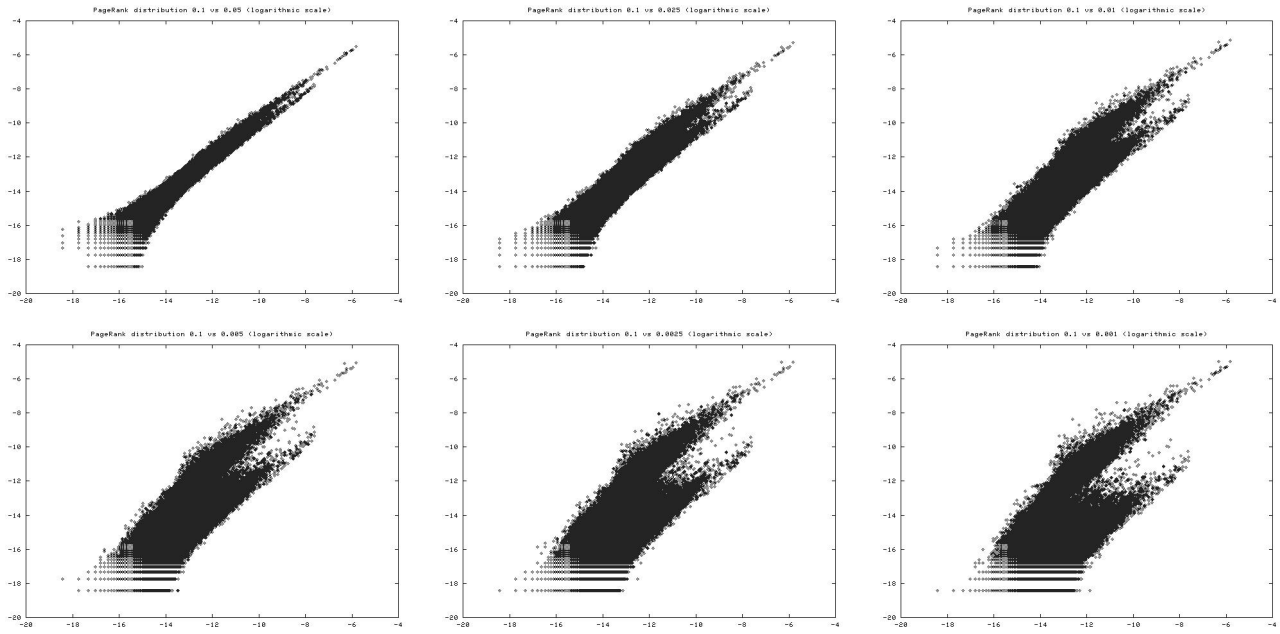

FIGURE 2. Each plot shows $\log \pi_\delta$ (vertical coordinate) against $\log \pi_\epsilon$ (horizontal coordinate) for $\delta < \epsilon = 0.1$

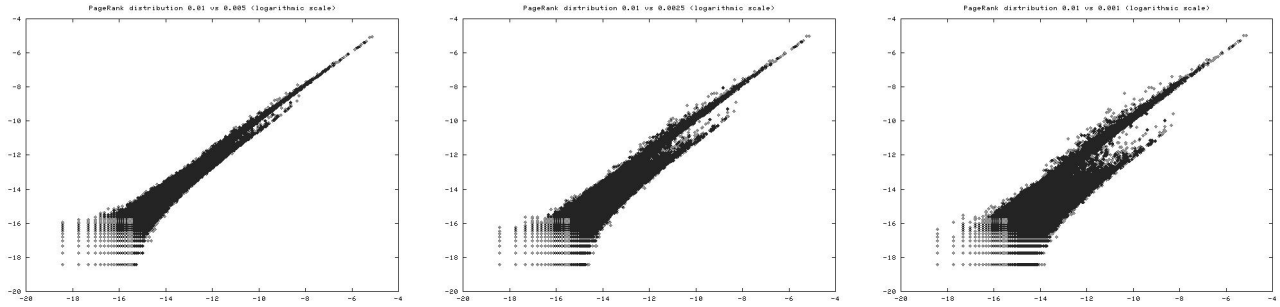

FIGURE 3. Each plot shows $\log \pi_\delta$ (vertical coordinate) against $\log \pi_\epsilon$ (horizontal coordinate) for $\delta < \epsilon = 0.01$

different page ranking algorithms, and as a way of constructing improvements. A detailed exploration is however beyond the scope of this document.

Recall from Theorem 1 that the page ranking distribution π_ϵ, which solves both (2) and (6), has the matrix analytic form

$$\pi_\epsilon = \epsilon\mu \cdot R_{1-\epsilon}(Q), \qquad R_{1-\epsilon}(Q) = \sum_{k=0}^{\infty} (1-\epsilon)^k Q^k, \tag{11}$$

for any $0 < \epsilon < 1$. Here the positive matrix Q is assumed to satisfy $\sum_y Q(x,y) = 1$ for all $x \in G$, although some results will hold if the condition is dropped. For any perturbation matrix $B(x,y)$ on G, the second resolvent equation (Hille and Phillips, 1957, Section 4.8, p.126) holds, provided the resolvents $R_\alpha(Q)$ and $R_\alpha(Q+B)$ both exist (ie have absolutely convergent components):

$$R_\alpha(Q+B) = R_\alpha(Q) + \alpha R_\alpha(Q+B)BR_\alpha(Q), \quad 0 < \alpha < 1. \tag{12}$$

The following result is a straightforward application of this equation.

Proposition 2. *Let θ be a probability distribution on G, and take $0 < \delta < 1$. Given a transition matrix Q, let Q^B be defined by*

$$Q^B(x,y) = (1-\delta)Q(x,y) + \delta\theta(y). \tag{13}$$

Let $\pi_{\epsilon,\mu}$ be a solution (probability measure) of (2) and (6), and similarly let $\pi^B_{\epsilon,\mu}$ solve (2) and (6) with Q replaced by Q^B. We have

$$\pi^B_{\epsilon,\mu} = \lambda\pi_{\epsilon+\delta(1-\epsilon),\mu} + (1-\lambda)\pi_{\epsilon+\delta(1-\epsilon),\theta}, \qquad \lambda = \epsilon/(\epsilon+\delta(1-\epsilon)). \tag{14}$$

Proof. In (12), let $\alpha = 1-\epsilon$, and multiply both sides of the equation by $\epsilon\mu$ on the left. Using (11), we have

$$\begin{aligned}\epsilon\mu R_{1-\epsilon}(Q^B) &= \epsilon\mu R_{1-\epsilon}\big((1-\delta)Q\big) + (1-\epsilon)\epsilon\mu R_{1-\epsilon}(Q^B)\delta(1\otimes\theta)R_{1-\epsilon}\big((1-\delta)Q\big),\\ &= \epsilon\mu R_{(1-\epsilon)(1-\delta)}(Q) + \delta(1-\epsilon)\epsilon\mu R_{1-\epsilon}(Q^B)1\cdot\theta R_{(1-\epsilon)(1-\delta)}(Q),\end{aligned}$$

which is equivalent to (14), since $\epsilon\mu R_{1-\epsilon}(Q^B)1 = 1$. □

The perturbed transition matrix Q^B corresponds to a transition dynamic which follows Q with probability $(1-\delta)$, and with probability δ resets the state independently according to θ.

In a different direction, suppose that B commutes with Q. We can then obtain a general perturbation formula for π^B in terms of a simple family of page ranking measures which generalise π.

Proposition 3. *Suppose that $QB = BQ$, and define*

$$Q^B = (1-\delta)Q + \delta B, \quad 0 < \delta < 1,$$

assuming that $\sum_y Q(x,y) = \sum_y B(x,y) = 1$ *for all* x. *Let* $\pi^B_{\epsilon,\mu}$ *solve* (2) *and* (6), *with* Q *replaced by* Q^B. *Then*

$$\pi^B_{\epsilon,\mu} = \lambda \sum_{j=0}^{\infty} (1-\lambda)^j \xi^{j+1}_{(1-\epsilon),\mu} B^j, \qquad \xi^r_{\alpha,\mu} = (1-\alpha)^r \sum_{n=0}^{\infty} \binom{n+r-1}{n} \alpha^n \mu Q^n,$$

where $\lambda = \epsilon/(1-(1-\delta)(1-\epsilon))$.

Proof. From the second resolvent equation (Hille and Phillips,1957, Theorem 4.8.3):

(15)
$$R_\alpha\Big(\gamma Q + (1-\gamma)B\Big) = R_{\gamma\alpha}(Q)\left\{ I + \sum_{j=1}^{\infty} \Big(\alpha(1-\gamma)BR_{\gamma\alpha}(Q)\Big)^j \right\}, \quad 0 < \alpha, \gamma < 1.$$

Since B commutes with Q, the sum can also be written

$$\begin{aligned} R_{\gamma\alpha}(Q) \sum_{j=1}^{\infty} \Big(\alpha(1-\gamma)BR_{\gamma\alpha}(Q)\Big)^j &= \sum_{j=1}^{\infty} \alpha^j (1-\gamma)^j \left[\sum_{k=0}^{\infty} (\gamma\alpha)^k Q^k\right]^{j+1} B^j \\ &= \sum_{j=1}^{\infty} \alpha^j (1-\gamma)^j \left[\sum_{n=0}^{\infty} \binom{n+j}{n} (\gamma\alpha)^n Q^n\right] B^j. \end{aligned}$$

Using this in (15), we obtain the formula

$$(1-\gamma\alpha) R_\alpha\Big(\gamma Q + (1-\gamma)B\Big) = \sum_{j=0}^{\infty} \Big(\frac{\alpha-\gamma\alpha}{1-\gamma\alpha}\Big)^j \left[(1-\gamma\alpha)^{j+1} \sum_{n=0}^{\infty} \binom{n+j}{n} (\gamma\alpha)^n Q^n\right] B^j,$$

and hence the result after setting $\alpha = 1-\epsilon$, $\gamma = 1-\delta$, and multiplying both sides by μ on the left. Note that $\xi^r_{\alpha,\mu}$ is indeed a probability measure as the coefficient of μQ^n is the probability mass of a negative binomial distribution. □

4. Modifying PageRank for page obsolescence

In this section, we describe the original motivation for the paper, a page ranking scheme of the type (2) and (6) which penalizes web pages according to their age.

The standard *PageRank* measure defined in (1) handles all links emanating from a web node in G equally. Thus the importance of a node is dependent solely on the number and importance of nodes linking to it. This gives a static view of the page ranking algorithm as well as the link graph, which fails to capture some of the dynamic aspects inherent in the World Wide Web.

Among the interesting properties shared by all World Wide Web documents is the creation/modification date. As this property is not directly referenced by the *PageRank* definition (1), is not immediately obvious that π_{PR} is, in fact, biased

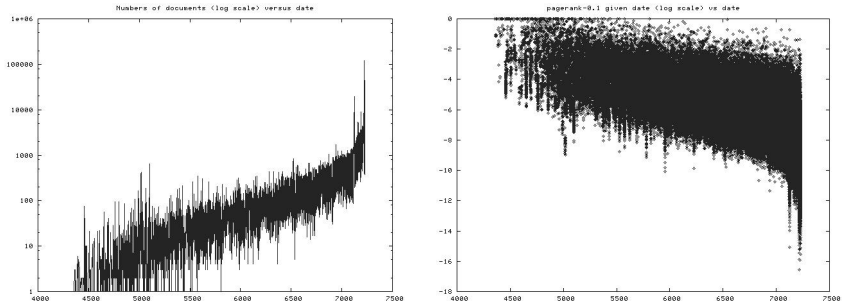

FIGURE 4. Document numbers increase exponentially with the date, but conditionally on the date, PageRank measure favours older documents.

towards older web documents. An explanation is easily found: Since the World Wide Web grows by aggregation, it follows that any new document must, if it contains external links at all, link to already existing, hence older, documents. The exception occurs when an older document is updated. Now *PageRank* favours documents which are linked to by many others, hence which tend to be older.

For the purposes of this section, we shall define the *date* of a document (i.e. a node in the web link graph) as its last modification or creation date. Assuming continuous crawling of the World Wide Web, the date of a document can be approximated by the earliest crawl date such that the contents haven't changed since that date. This is what we do for our dataset, where we take the minimum of the last modification date (if it is recorded) and the crawl date. A small percentage of documents have neither, and we set their date arbitrarily to zero.

The bias towards older documents can be observed in our dataset, which contains $916,428$ documents with valid dates ranging (in days) from $\tau_{\min} = 4358$ to $\tau_{\max} = 8094$. However, a naive plot of document rank $\pi(x)$ against document date $\tau(x)$ won't exhibit the effect, because the number of documents with a given date increases exponentially with the date, see Figure 4. Instead, we look at the *conditional page ranking* distribution given the date,

$$\pi(x \mid t) = \frac{\pi(x) 1_{\{\tau(x)=t\}}}{\sum_{\tau(y)=t} \pi(y)},$$

which can be estimated through simulation as described later in this paper. In Figure 4, we plot for each document x an estimate of $\log \pi_{\mathrm{PR}}(x \mid \tau)$ against $\tau = \tau(x)$.

Clearly, the conditional PageRank tends to be higher for older documents. A similar effect is observed if we plot instead $\pi(x)/\#\{x : \tau(x) = t\}$ versus t.

Independently of *PageRank*, age also affects the documents on the World Wide Web in a negative way: Older documents are more likely to disappear or be moved/modified in such a way as to be unrecognizable. We can discriminate against older documents, while keeping the general flavour of *PageRank* if we calculate the page ranking distribution π_D, based on (2), (6) and

$$Q_D(x,z) = C_\lambda^{-1}(x) \sum_{x \to y} \exp\left[-\lambda\left(\frac{\tau(x)-\tau(y)}{\tau_{\max}-\tau_{\min}}\right)_+\right]\delta_y(z), \tag{16}$$

where $C_\lambda(x)$ is the normalizing constant such that $\sum_y Q_D(x,y) = 1$. If $\lambda = 0$ in (16), we see that $Q_D = Q_{\mathrm{PR}}$, and π_D reduces to π_{PR} (see Corollary 1). If $\lambda > 0$ however, we see that transitions from x to older documents (where $\tau(y) < \tau(x)$) are penalized, while transitions to newer documents are still chosen equally, as with *PageRank*. Note that Q_D is *not* of the form Q^u for some function u, hence Proposition 1 does not apply here.

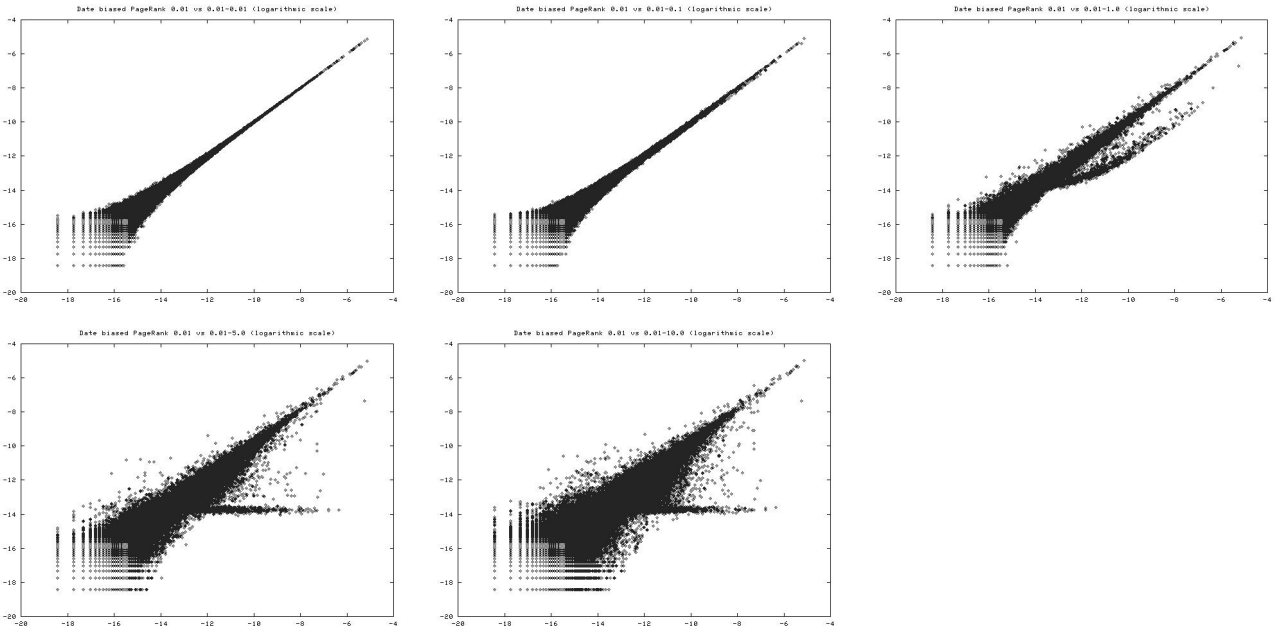

FIGURE 5. Each plot shows $\log \pi_D$ (vertical coordinate) against $\log \pi_{\mathrm{PR}}$ (horizontal coordinate) for $\epsilon = 0.01$ and various λ

A detailed analysis of π_D is beyond the scope of this paper, partly due to the absence of a theoretical framework for meaningful comparisons with π_{PR}. We show here some simulation plots and discuss them very briefly.

Both Figures 5 and 6 show plots of $\log \pi_D$ against $\log \pi_{\mathrm{PR}}$ for fixed values of ϵ and a range of λ values.

It is noteworthy that the qualitative behaviour appears not to depend on ϵ, even though $\pi_{PR(0.01)}$ and $\pi_{PR(0.3)}$ differ somewhat. This is borne out by other plots not shown here. Examining the plots more closely, it is apparent that increasing λ tends to reduce (i.e. send below the diagonal) the rank of a certain subset of documents, with most of the others keeping their rank somewhat unchanged (i.e. staying close to the diagonal). The limiting horizontal branch, which can be seen in the last two plots of Figure 5 ($\lambda \geq 5$), is hard to explain.

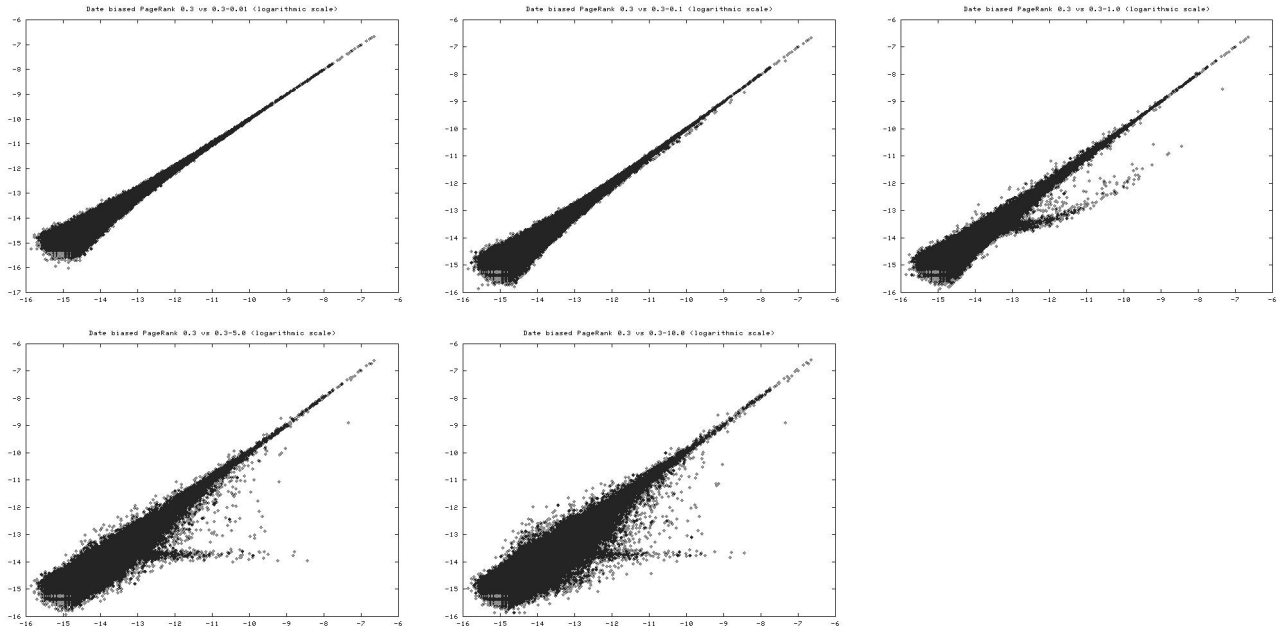

FIGURE 6. Each plot shows $\log \pi_D$ (vertical coordinate) against $\log \pi_{\mathrm{PR}}$ (horizontal coordinate) for $\epsilon = 0.3$ and various λ

5. Constructing the link graph

In this section, we describe the data structures and algorithms used to construct an in-memory model of the link graph G. This model is used to run the various Markov chains descripbed in this paper.

The dataset provided by Google, Inc., consists of a collection of 916,428 documents in hypertext markup language format (html). Each document has a web address (url) which is a string of the general form

$$[scheme:][//netloc][/path][?query][;parameters][\#fragment],$$

where each element in brackets is optional, and each element in italics is a string of ASCII characters (see RFC 2396). All links which appear in a document employ urls to designate the targeted document. By decoding this string, and knowing the url of the document on which it appears, we can deduce the targeted document's fully qualified url, and in particular whether or not this document exists in our dataset, warranting a link in G.

Some documents have several urls pointing to them, this being due to web server redirections. Links to documents not appearing in G (but existing on the Web) are called *dangling links*, and as these do not appear in the formula (1), they must be recognised as such in the dataset.

Capitalisation in urls is a source of problems. Some elements, such as the *scheme*, are not case sensitive. The *netloc* element isn't either, provided it contains a DNS address. The *path* however can be case sensitive or not, depending upon the operating system installed on the file server. Moreover, most authors of web pages aren't aware of these difficulties, which compounds the effect. These aspects imply that it is impossible to decide completely reliably if two urls refer to the same document, so that any link graph G which we construct will necessarily be only approximately correct. The convention we shall use is to make case sensitive comparisons for all elements except *scheme* and *netloc*.

Besides the problem of redirections, there is a rare issue of consistency of the dataset which we address as follows: whenever two physical documents have the same url, we *merge* the two representations. Thus, instead of trying to decide which of the two is the true document, we construct a representation containing

all links emanating from each constituent, etc. The consistency issue can occur due to nonstandard server settings, for example if the default *index.html* file is renamed. We use the convention that a url ending in a directory name refers to the file *index.html* within that directory, although we cannot be certain.

The procedure to create the link graph G described here requires that each individual document $x \in G$ be associated with one or more url strings, which are kept in memory during construction. After properly formatting and fully qualifying the urls into a standard form (which tends to increase their relative sizes, but is necessary for later reference), this means that on the order of 650 Mb of storage is required for approximately 4 million document urls and link urls combined. The test machine has only 256 Mb of memory, so it is worth considering ways of compressing these requirements somewhat. In light of the true size of the Web, the amount of compression achieved also has an impact on the scalability of future analyses with larger datasets.

A second and related design decision concerns the number of passes through the original data needed to build the graph. If we only store document urls in memory, we need two passes through the dataset, first to build the nodes $x \in G$, then to build the links $x \to y$ through comparisons with the urls in memory. This requires an efficiently searchable data structure to store the strings, such as a hash table. The supplied dataset has on average 12 links per document, 6 of which are dangling. Alternatively, if we use a single pass through the dataset (reading the dataset on disk is relatively slow), we need to keep both document urls and link urls in memory. For the full dataset, this translates to approximately 4 million distinct url strings.

The method we use here is to store the url strings in a trie, which allows us to take advantage of the natural hierarchical structure of urls. For example, the following urls all occur in the dataset:

- `http://www.bu.edu/`
- `http://www.bu.edu/iscip/`
- `http://www.bu.edu/iscip/news.html`
- `http://www.bu.edu/iscip/perspective.html`
- `http://www.bu.edu/com/html/events.html`

- `http://www.bu.edu/com/html/faculty1.html`

It is clearly wasteful to store the common prefix for every url. The arborescence in Figure 7 shows how a trie structure (Knuth, 1997, Vol. 3, Section 6.3) prevents wasteful duplication. The arborescence itself can be easily stored in a linear structure (array) by using branch links as shown in Figure 8. The table of branch links itself is taken as a hash table, which allows fast ($O(1)$) retrieval of every branch link when needed.

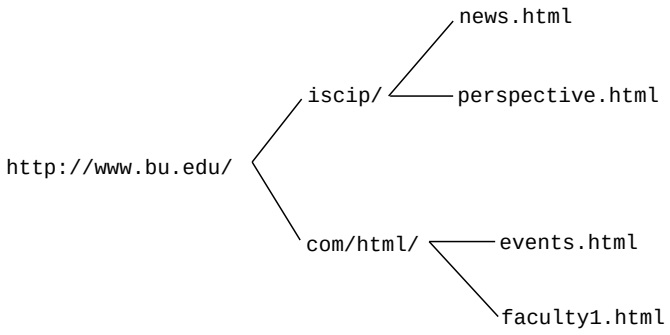

FIGURE 7. Trie structure prevents duplication of common url prefixes.

To determine the existence of a particular url s within the trie, we start at the root of the trie, and compare prefixes up to the first encountered branch. We then compare the first character in each branch to determine which to follow, and repeat. As each url string is composed of ASCII characters (128 exist in total), there can be at most 128 branches for each character in our search string s, thus giving an upper bound of $128 \cdot \text{lenght}(s)$ for the number of comparisons required to find s, regardless of the size of the trie. In practice, this will be considerably less.

In the linearised situation of Figure 8, each branch, here represented by a curved arrow, gives rise to a single pair (k, v) stored in a hash table. The value k, which is used as the key in this hash table, is the number of characters from the beginning to the point where the arrow representing the branch originates from. The value v is a pointer to the character which represents the destination of the arrow. Since there are as many branches in the trie as there are distinct inserted url strings, this number is also the required size of the branch hash table. The total storage requirements are therefore given by the number of characters (one byte each) stored

in the trie plus eight bytes for every branch link (four bytes are needed to encode a number up to 4 billion).

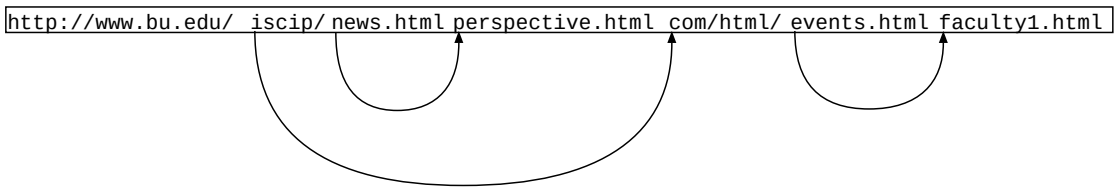

FIGURE 8. Linearised trie. Each arrow is a branch link, with origin k and destination v.

The reason we store the branches in a hash table is speed and memory savings. Were we to embed the branches directly into the trie, we would have to reserve at arbitrary locations enough space for a character pointer, wasting much memory. On the other hand, each trie search requires tens of branch lookups, so an $O(\log N)$ data structure such as a tree would be noticed, due to the large number of branches (approximately 4 million) inserted.

A further (but smaller) memory saving is obtainable from noting another property of url strings. Most such strings refer to html documents, which typically end in the string *.html* or *.htm*. Since these suffixes occur at the end of the inserted strings, they are not shared within the trie and take up much space. We therefore encode common suffixes by a single non printable ASCII character, which further reduces the size of the trie for our dataset by approximately 20 Mb.

A second, much smaller hash table is used to associate each document url string in the trie with the constructed document node within the graph. This allows the trie to be queried as to the existence of a particular document in the graph, and is useful for building connections between the nodes. As the documents are read from the dataset, a node's *to* links (pointing to other nodes) are represented firstly as character pointers to the end of the corresponding url string within the trie. This is necessary since it is not known at that stage whether a linked to document belongs to the dataset, or whether the link is *dangling*. Once the full dataset has been processed, we pass a second time through the graph nodes, checking for each link if the corresponding url in the trie is associated with a node, and if so convert the character pointer into a proper node pointer.

Again, this hash table is used instead of embedding the node pointers directly in the trie, since the latter contains very many urls which aren't associated with a node in the graph.

Because the trie allows large memory savings, the final program can use a single pass over the dataset, and stores both document and link urls in memory. With $916,428$ documents and roughly 3 million unrelated link urls, the storage requirements are only about 75 Mb, as opposed to theoretically 650 Mb for the full uncompressed set of standardised strings. This is quite sufficient for our purposes.

The storage requirements for the web graph G itself vary depending upon the amount and type of information retained. Here, we make no particular effort to keep the overall size of the graph small, preferring to store extra information so as to keep simulations fast and simple to program. For each document (ie node $x \in G$), we store a list of links to other documents $y \in G$ (approximately 6.5 per node) as well as the list of documents in G linking to x (approximately 6.5 per node also, by coincidence). Since a single pass through the dataset doesn't allow us to build the links between the nodes directly (for we don't know until the end if a linked to document y belongs to our dataset or not), we at first make each document's links point to their own urls as stored in the trie. Whenever a new document's url is inserted in the trie, we give it a pointer to the actual node $y \in G$. After the full dataset was read in, a second pass through the nodes in memory allows us to link the nodes directly, bypassing the trie.

Table 1 summarises the actual sizes encountered in the dataset. The parity between average *from*links and average *to*links seems to be an artifact of the dataset.

6. Simulating page ranking schemes

In this section, we consider again the case when π is of the form (2), with P satisfying (6). By Theorem 1, we have the formula (7), which shows that we can obtain a sample from π by first choosing a document from the distribution μ, and then applying T transitions with probabilities $Q(x, y)$, where T is geometrically distributed with mean ϵ^{-1}. This approach to simulating π is particularly useful for *parallel processing*, since each sample from π can be generated independently, thus minimising the communication needed between processors. To obtain n samples

Number of documents (size of G)	916,428
Number of unique url strings inserted in trie	4,069,633
Total size (characters) of standardised url strings	668,245,361
Memory needed for strings in trie (bytes)	43,796,267
Memory needed for branch hash table (bytes)	32,557,056
Memory needed for (url $\rightarrow$ node) pointer hash table (bytes)	7,451,984
Minimum size (bytes) of each web node (excl. *link pointers)	32
Average number of *to*links for each web node	6.58442
Average number of *from*links for each web node	6.68272
Average number of *dangling* tolinks for each web node	5.58942
Total memory used by G (bytes)	98,448,388

TABLE 1. Data structure statistics for Google's dataset

from π, we would first generate n independent random variables (nodes/documents) distributed uniformly on the link graph G, and then evolve them simultaneously or sequentially. This approach is more efficient in traversing the link graph G. Moreover, it can be readily parallelized, see Section 7.

For completeness, let us mention that, besides simulating from π, it is also quite easy to obtain random samples from the measures $\Pi_{\epsilon,\delta}$ described at the end of Section 2. As above, we begin by generating a π_ϵ random variable. This however now takes the place of μ as a starting point, and we generate another variable T', which is geometrically distributed with mean δ^{-1}. We then perform T' extra transitions according to Q.

Finally, we mention that the ability to produce i.i.d. samples from π makes the construction of confidence intervals for $\pi(x)$ trivial. Suppose that n samples $Z_1, \ldots, Z_n$ from π are available. For each $x \in G$, the occupation number $k_n = \#\{t : Z_t = x\}$ is binomially distributed, with success probability $\pi(x)$. In particular,

$$\mathbb{P}\left(\frac{k_n}{n} - \frac{1.96}{n}\sqrt{\frac{k_n(n-k_n)}{n}} < \pi(x) < \frac{k_n}{n} + \frac{1.96}{n}\sqrt{\frac{k_n(n-k_k)}{n}} \right) \approx 0.95. \tag{17}$$

We can also obtain absolute (but very loose) bounds on the number n of samples needed to achieve a fixed *relative* error for each estimated $\pi(x)$. This uses the universal lower bound $\pi(x) \geq \epsilon\mu(y)$ for all $x \in G$, which comes from (6). Combining

this with (17) in the case of *PageRank*, we write

$$\mathbb{P}\left(\left|\frac{(k_n/n)-\pi_{\mathrm{PR}(\epsilon)}(x)}{\pi_{\mathrm{PR}(\epsilon)}(x)}\right| \leq \frac{3.92\,|G|}{\epsilon\sqrt{n}}\sqrt{\frac{k_n}{n}\Big(1-\frac{k_n}{n}\Big)}\right) \approx 0.95,$$

and since $k_n \leq n$, this gives $\mathbb{P}\Big(|\text{relative error}| \leq 3.92\,|G|\,/4\epsilon\sqrt{n}\Big) \approx 0.95$. This bound is unsatisfactory for large graphs G since it requires $O(|G|^2)$ samples to guarantee a (universally small) relative error. Tighter bounds can be expected to depend upon the actual geometry of G.

In contrast to the preceding statements, the situation becomes theoretically more interesting if, as is suggested by (5), we attempt to compute π only on a relatively small subset S of G, say $S = \{x : \omega \in x\}$. Due to the typically very small magnitude of $\pi(x)$ when $|G|$ is large (typically less than $|G|^{-1}$), the occupation numbers k_n include a very large relative error. Essentially, the overwhelming majority of samples fall outside of S, while only the samples falling in S are informative (about π restricted to S). To solve this problem, we can try to sample directly from $\pi|_S(x) = \pi(S)^{-1}\pi(x)1_{(x\in S)}$, the conditional distribution of π on S.

The simplest (rejection sampling) technique consists in sampling $Z_1, \ldots, Z_n$ from π as explained earlier, and then throwing away all samples which don't end up in S. The usefulness of this scheme hinges on being able to limit dramatically the wasted computations, that is, to decide early if a sample will end up in S or not. This is possible for certain forms of the matrix Q, including Q_{PR}.

To see how this works, consider that each random variable Z_t requires $T(t)$ transition steps $Z_t^0, Z_t^1, \ldots, Z_t^{T(t)} \equiv Z_t$, where $Z_t^0 \sim \mu$ and $Z_t^{s+1} \sim Q(Z_t^s, \cdot)$ for $s = 0, \ldots, T(t)-1$. Crucially, the number of steps $T(t)$ is *independent* of the path $Z_t^0, \ldots, Z_t^{T(t)}$. Let $S(0) = S$, and define

$$S(-(t+1)) = \{x \in G : x \to y \text{ for some } y \in S(-t)\}, \quad t = 0, 1, 2, \ldots, \tag{18}$$

then a *necessary* condition for accepting $Z_t \in S$, given that $T(t) = k$ say, is that $Z_t^0 \in S(-k)$. Provided that the sets $S(-k)$ are small compared to G, the following will be an improvement over naive rejection sampling:

Theorem 2. *Let Q satisfy $Q(x,y) > 0$ if and only if $x \to y$, and $\sum_y Q(x,y) = 1$ for all x. The following procedure yields a sample from $\pi|_S$: Let $k > 0$ be a fixed constant and compute the sets $S(-1), S(-2), \ldots, S(-k)$ as in* (18).

1 *Let T be a random variable with distribution*

$$\mathbb{P}(T=t)=\begin{cases}\epsilon(1-\epsilon)^t\mu(S(-t))/\mu(G) & \text{if } t\le k,\\ \epsilon(1-\epsilon)^t & \text{otherwise.}\end{cases}$$

2 *Independently, set $Z^0\sim\mu|_{S(-T)}$ if $T\le k$, or $Z^0\sim\mu$ if $T>k$, and for each $t\le T$, put $Z^t\sim Q(Z^{t-1},\cdot)$. If $Z^t\notin S(-T+t)$ for some $t\le T$, we reject the proposal and begin anew at step 1. If all $Z^t\in S(-T+t)$, then the variable Z^T is a sample from $\pi|_S$, and we stop.*

Proof. To obtain a sample from $\pi|_S$, it suffices to sample from π repeatedly until the result falls into S. Using the definition (18), we have the identity $1_{\{S(-j)\}}(x)Q^j(x,y)=Q^j(x,y)$ for all $y\in S$. Consequently, from (7) and for $y\in S$, the probability density $\pi(y)$ can be written

$$\pi(y)=\epsilon\sum_{j=0}^{\infty}(1-\epsilon)^j\sum_{x\in G}\big(\mu(x)/\mu(G)\big)Q^j(x,y)=$$
$$\epsilon\sum_{j=0}^{k}(1-\epsilon)^j\big(\mu(S(-j))/\mu(G)\big)\sum_{x\in G}\big(\mu(x)1_{\{S(-j)\}}(x)/\mu(S(-j))\big)Q^j(x,y)$$
$$+\epsilon\sum_{j=0}^{\infty}(1-\epsilon)^j\sum_{x\in G}\big(\mu(x)/\mu(G)\big)Q^j(x,y).$$

The right hand side is simulated directly by steps 1 and 2, while the event

$$\{Z^t\in S(-T+t)\text{ for all }t\le T\}$$

is clearly equivalent to $\{Z^T\in S\}$. □

For best results, we should choose k such that the probability mass represented by $(p_1,\dots,p_k)$, where $p_j=\epsilon(1-\epsilon)^j\mu(S(-j))/\mu(G)$, is close to one. This is of course highly dependent on the geometry of the link graph G and the size of S.

Implementing this algorithm requires the construction and representation of the sets $S(-j)$. This can be handled efficiently with a bit field associated to each node $x\in G$. Suppose we take $k=16$, then two bytes are needed for each node, and the membership of x in $S(-j)$ is signalled by setting the j-th bit to one (otherwise leaving at zero). Since we already store *from*links as well as *to*links, the construction of the sets $S(-j)$ is easy and requires at the most k passes through the whole graph G.

We note also that for reasons of efficiency in traversing the link graph G, the samples from $\pi|_S$ should not be generated sequentially. Instead, it is better to build a large number $N > n$ of candidate proposals $Z_1^0, \ldots, Z_N^0$ and evolve these simultaneously, discarding the failing candidates.

Informal simulations suggest an efficiency gain in the number of conditional samples produced ranging between 1.5 and 3 times, compared to the standard (unconditional) sampling scheme, depending upon the set S.

7. Distributed simulations

As the statements in the introduction made clear, while the dataset provided is too large to be conveniently manipulated with most current analysis tools on commodity computers, it is nonetheless very small compared to the true size of the World Wide Web. Here we propose a brief analysis of the prospects for distributing the simulations on a cluster of computers. The code for this has been implemented by the author, and uses the Parallel Virtual Machine framework (Geist et al., 1994).

Distributed web link graph. There is no conceptual difficulty in distributing a large dataset of web documents onto several machines. If $|G|$ is the total number of web documents, J is the number of separate (identical) computers, each of these will require enough memory to store $|G|/J$ documents, say. Taking the memory requirements outlined in Section 5 as approximately 200 Mb for one million documents, we expect that a machine with up to 4 Gb of RAM can accommodate up to 20 million documents. A set of 3 billion documents would require 150 such machines.

The memory requirements for a single machine include the need to store the trie of urls: In building a distributed web link graph $G = G_1 \cup \cdots \cup G_J$, we want to take into account the links whose origin and destination belong on different machines. On each machine a, the nodes $x \in G_a$ are labelled by a unique integer identifier. However, the raw dataset represents links by their url addresses, so these need to be used for the construction of links between machines.

The construction of the distributed link graph proceeds in two stages. In the first stage, each machine a builds interconnected nodes for all the documents it receives, and writes a list of (standardised) urls for all the nodes it owns into a file. In the second stage, this file is read by all other machines $b_1, \ldots, b_{J-1}$, which build

leaf nodes for each url which also exists in their own trie. Since the trie in machine b contains a url s if and only if s is either the address of a node in G_b or a node linked by a node in G_b, this procedure builds one leaf node for each connection to some other machine, while dangling links are ignored. A leaf node contains much less information than a full document node. It is there only as a proxy, to reduce the communication requirements between machines during a simulation.

The full cost of building the distributed web graph is $O(|G|)$ for each machine, or $O(J\,|G|)$ in total. This is due to the sparseness of the incidence matrix of G. For example, each node in the dataset links to 12 documents on average, six of which are dangling. The first stage passes through the dataset once to build the nodes, and once to build the internal connections, obtaining G_a. In the second stage, each machine $b \neq a$ reads $J-1$ files, containing $|G|\,/J$ urls, in a single pass to obtain the leaf nodes. These are connected to the graph G_b in a single pass.

Distributed sampling. We now discuss the simulation of a page ranking scheme π satisfying (2) and (6). Let n be the total number of samples required. We shall only discuss simulating the full π distribution, not a restricted version $\pi|_S$. Initially, there are n active particles, chosen independently according to μ. So each particle moves independently, performing a geometric number of moves with success probability ϵ. Afterwards, it is removed, and its last location x is recorded, by incrementing the occupation count associated with the node x. The total number of transitions required for the full calculation has a negative binomial distribution with success probability ϵ and n required successes. This means an average n/ϵ transitions with a standard deviation of $\sqrt{n(1-\epsilon)}/\epsilon$.

In the distributed case, we must also take into account transitions in which a particle crosses from machine a to another machine b. This costs a message from a to b, but can be amortised by batching. The idea is that machine a should evolve all particles it owns until it cannot continue. Each particle performs its transitions on G_a according to Q until it either stops (with probability ϵ) or it reaches a leaf node. Stopped particles are recorded by incrementing the occupation counts for the relevant nodes. Occupied leaf nodes are grouped by destination machine, which is sent a single message consisting of the number and location of relevant particles. A maximum of J messages is sent during this propagation step.

On the receiving side, the propagated particles are placed on the relevant nodes and a new partial simulation is done. Due to the Markov property, only the current locations of the particles are needed for the simulation. Also, the messages received can be processed in any order, and even together by superposition of the particle counts. The full simulation stops when all particles have been removed.

It is easy to see that the simulation must in fact stop, since at the end of each partial simulation, the number of particles on the leaf nodes of a is less than or equal to the number of particles on G_a at the beginning of the partial simulation. With positive probability, this is a strict inequality. Thus, the total number of propagated particles is monotone decreasing. Once it reaches zero, the simulation must stop.

The total number of messages sent is of interest because the cost of each message is an order of magnitude larger than a single transition, although the individual messages tend to become smaller over the course of the simulation.

A simple worst case bound can be worked out as follows: A propagation event is needed whenever there is a transition from one machine to another. Each such event sends at most J messages from each computer, or a maximum of J^2 messages in total. The number of propagation events is less than or equal to the number L of transitions of the longest lived particle. Since the k-th particle makes a geometric number $T(k)$ of transitions, independently of the others, we must have

$$\mathbb{P}(L \leq k) = \mathbb{P}\big(\max\{T(1), \ldots, T(n)\} \leq k\big) = [1 - (1 - \epsilon)^k]^n,$$

and consequently an average worst case bound for the total number of messages transmitted within the computing cluster is $J^2\mathbb{E}[L] = J^2 \sum_{k=0}^{\infty} \mathbb{P}(L > k)$. Note that the more informative average case bounds are difficult to obtain analytically, as they would require assumptions on the geometry of G and the way the data is distributed on the various machines. Table 2 shows a single distributed simulation run with two machines.

References

[1] Berners-Lee, T., Fielding, R. and Masinter, L. (1998) Uniform Resource Identifiers (URI): Generic Syntax. *IETF RFC 2396*.

[2] Breyer, L.A. and Roberts, G.O. (2001) Catalytic perfect simulation *Methodology and computing in applied probability*.

Machine A		Machine B	
Particles	Leaf nodes	Particles	Leaf nodes
46,495,390	-	53504610	-
27,169,094	282,006	23,850,799	302,294
12,196,554	253,010	14,027,057	272,527
7,171,160	233,537	6,297,161	244,963
3,215,618	204,828	3,706,939	221,836
1,890,765	181,620	1,660,393	184,284
846,022	142,625	947,435	155,782
496,777	115,949	436,892	113,198
222,822	78,110	256,401	86,834
130,264	57,287	114,774	53,832
58,181	33,236	67,227	37,786
34,313	22,310	29,983	20,544
15,254	11,602	17,707	13,382
9,051	7,413	7,843	6,625
4,035	3,545	4,661	4,094
2,352	2,154	2,008	1,845
1,047	1,004	1,189	1,125
576	560	252	509
266	260	295	293
140	140	145	142
64	63	67	67
36	36	37	37
14	14	19	19
12	12	8	8
4	4	8	8
5	5	3	3
1	1	4	4
3	3	1	1
-	-	1	1

TABLE 2. A distributed simulation run with two machines producing $100,000,000$ samples of PageRank with $\epsilon = 0.1$.

[3] Brin, S. and Page, L. (1998) The anatomy of a large-scale hypertextual web search engine. *7th International World Wide Web Conference*, Brisbane, Australia, 1998.

[4] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. and Sunderam, V. (1994) *PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing.* MIT Press.

[5] Hille, E. and Phillips, R.S. (1957) *Functional Analysis and Semi-groups.* AMS.

[6] Kleinberg, J. (1999) Authoritative Sources In A Hyperlinked Environment. *Journal of the ACM* **46**.

[7] Knuth, D. (1997) *The Art Of Computer Programming, Volume 3: Sorting and Searching.* Addison-Wesley.

[8] Labovitz, C. Ahuja, A. and Bailey, M. (2001) Shining Light On Dark Internet Address Space. Arbor Networks, Inc. Technical Report.
`http://research.arbornetworks.com/up_media/up_files/dark_address_space.pdf`

[9] Meyn, S.P. and Tweedie, R.L. (1993) *Markov Chains And Stochastic Stability.* Springer.

[10] Revuz, D. (1984) *Markov Chains.* North-Holland.

[11] Sparck Jones, K., Walker, S. and Robertson, S.E. (1998) A probabilistic model of information retrieval: Development and status. Tech. Rep. 446, Cambridge University Computer Laboratory.

[12] Wilson, D. (2000) How to couple from the past using a read-once source of randomness. *Random Structures and Algorithms* Vol. 16, Number 1, pp.85–113.

E-mail address: `laird@lbreyer.com`