Approximating Aggregate Queries about Web Pages via Random Walks [*]

Ziv Bar-Yossef[†] Alexander Berg Steve Chien[‡] Jittat Fakcharoenphol[§]
Dror Weitz[¶]

Computer Science Division
University of California at Berkeley
387 Soda Hall #1776, Berkeley, CA 94720-1776
U.S.A.
{zivi,aberg,schien,jittat,dror}@cs.berkeley.edu

Abstract

We present a random walk as an efficient and accurate approach to approximating certain aggregate queries about web pages. Our method uses a novel random walk to produce an almost uniformly distributed sample of web pages. The walk traverses a dynamically built regular undirected graph. Queries we have estimated using this method include the coverage of search engines, the proportion of pages belonging to .com and other domains, and the average size of web pages. Strong experimental evidence suggests that our walk produces accurate results quickly using very limited resources.

1 Introduction

Timely and accurate statistics about web pages are becoming increasingly important for academic and commercial use. Some relevant questions are: *What percent of web pages are in the .com domain? How many pages are indexed by a particular search engine? What is the distribution of sizes, modification times, and content of web pages?* These questions can be written as either aggregate queries or selection queries. We present a method for efficiently and accurately approximating the results of these and other similar queries.

In order to answer these questions we need to estimate the size of certain sets of web pages. In particular, given a user defined boolean predicate, p, we want to estimate the fraction of all web pages that would satisfy a selection query using p. We also want to estimate the results of aggregate queries about all web pages. For example we might estimate the average number of bytes in a web page.

We present an efficient solution, requiring small amounts of computing power, storage space, and network bandwidth. Starting from scratch, we can accurately estimate results for the previously mentioned queries in as little as a few days using one PC with a modest connection to the Internet. Furthermore, given additional time and network bandwidth, our method can produce increasingly accurate results.

It is important to understand that there is no complete, current index of the web. The web is a hypertext corpus connected by a link structure, and it is possible to traverse (crawl) this link structure in order to obtain an index of the web. Unfortunately even the fastest crawls require 3 to 4 months to visit a large fraction of the web, and must utilize significant computer and network resources. When combined with the fact that thousands of pages are added, modified, and deleted every day this means that an index of the web generated by a crawl will be neither current nor complete.

[*] This work used the Berkeley Now machines, supported by grant CDA-9401156

[†] Supported by NSF Grant CCR-9820897

[‡] Supported by NSF Graduate Research Fellowship

[§] Supported by the Faculty of Engineering Kasetsart University Scholarship and by the Fulbright Scholarship

[¶] Supported by the University of California Regents Fellowship

**Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.**

1.1 Our Solution: Random Walks for Uniform Sampling

In order to estimate the results of aggregate queries or the fraction of all web pages that would satisfy a selection query for a given predicate, we will use random sampling. First, a uniformly distributed sample of web pages will be produced. It will then be used to estimate the quantities in question. The fraction of sampled web pages that satisfy a predicate is an estimate for the fraction of all web pages that would satisfy the predicate.

It is important to note that the accuracy of the random sampling technique depends on the selectivity of the predicate. In particular if a large fraction of all web pages satisfy the predicate, then the estimate will be more accurate than if a very small fraction of all web pages satisfy the predicate.

In order to produce a uniformly distributed sample of web pages without an index of all web pages, we will use ideas from the theory of random walks. It is well known that random walks on regular [1] undirected graphs can provide a close to uniformly distributed sample of nodes. Unfortunately the web is neither undirected, nor regular. We present a method to simultaneously walk the web and a dynamically generated graph G. The graph G has a node for each web page discovered on the web, but is regular and undirected. As the walk progresses it produces a close to uniformly distributed sample of web pages.

The next important question is how quickly our process can produce a close to uniformly distributed sample of web pages. The answer to this question is related to the structure of the generated graph G. We have analyzed the structure of the web obtained from a large Alexa [1] crawl circa 1996 to conclude that such a walk can converge extremely fast. We also demonstrate that, in fact, the walk we implement quickly finds an apparently close to uniformly distributed set of pages from this crawl.

Our random walk strategy is finally validated by results on the web today. In section 4 we compare our estimate for the distribution of web pages by domain (.com .edu .net .org, etc.), to that published by Inktomi as a result of a recent and extensive crawl over a period of four months. We also compare estimates for the size of the search engines FAST and AltaVista. In the first case the approximations we obtain from our approach in one to two days are consistent with those of Inktomi's extensive crawl. The coverage estimates for FAST and AltaVista are also consistent with various other estimates. These results provide strong evidence that our walk performs well.

1.2 Outline

We begin by describing an ideal but unrealizable random walk on the web in Section 2. We then describe our random walk, a close approximation of the ideal one, in Section 3. We present strong experimental evidence that our walk works well and applications of our walk to aggregate queries about the current web in Section 4. Section 5 discusses related work and Section 6 concludes.

2 Ideal Random Walk on the Web

In this section we develop an idealized random walk on the web. This ideal random walk will be provably accurate and efficient, but unfortunately cannot be implemented. Nevertheless, it provides a useful model for our actual random walk, described in Section 3.

2.1 The Indexable Web

Before explaining our random walk on the web, we must first define what we mean by "the web". A recent experiment [12] suggests that the structure of the web is similar to that shown in Figure 1. According to this model the web graph divides into four parts of roughly equivalent size:
(1) A giant strongly connected component (the largest subset of nodes from which every pair of nodes are mutually reachable from one another by following links).
(2) A "right" side containing pages reachable from (1), but which cannot reach (1) in return.
(3) A "left" side whose pages can reach (1), but are not reachable from (1) in return.
(4) All the rest (other small connected components, or pages that link to the right side or are linked from the left side).

We refer to the union of (1) and (2) as the "indexable" web, since this is the part that is easily explored by most users and search engines, and that arguably contains most of the meaningful content of the web. Our random walk and experiments are conducted mainly on this part of the web.

2.2 Random Walks on Graphs

In its most general form a random walk on a graph is simply what its name suggests. Let $G = (V, E)$ be a (directed or undirected) graph, where $V = \{v_1, v_2, \ldots, v_N\}$ is the set of vertices and E the collection of edges. A random walk is then a stochastic process that iteratively visits the vertices of the graph. The next vertex of the walk is chosen by following a randomly chosen outgoing edge from the current one. The nodes visited by a random walk can be written as a sequence $x_0, x_1, x_2, \ldots$. Furthermore, since the walk is random, it makes sense to describe the state of a random walk after t steps as a probability distribution X_t over the graph's vertices, described as a vector of length N.

[1] A graph is regular if all its nodes have an equal number of incident edges.

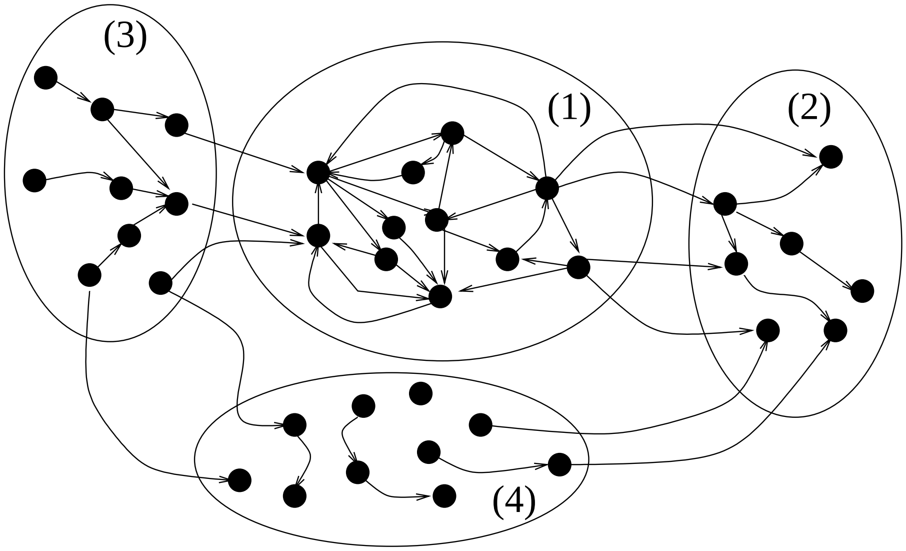

Figure 1: Web Graph Structure

The web can be naturally described as a graph W with pages as vertices and hyperlinks as directed edges, and for the remainder of this paper it will be convenient to identify vertices with pages and edges with links.

Our goal was to develop a random walk on the web's graph structure that satisfies two criteria: first, the walk's state (X_t) should approach the uniform distribution, and second, the number of steps required for the walk to approach this distribution should be reasonably small.

It turns out that the web's graph structure is unwieldy and the straightforward random walk of randomly choosing a link from the current page will not work. The straightforward walk will have a heavy bias towards high-degree nodes, may become trapped in leaves, and requires too many steps to run.

We now describe the ideal (and unattainable) scenario in which such a simple scheme would perform well, and discuss our approximation of it in Section 3.

2.3 The Ideal Walk: Uniform Distribution

Ideally, our random walk will be a Markov chain on a connected, undirected, regular graph. Walks with these properties can be proven to eventually approach a uniform distribution over the vertices of the graph.

A Markov chain is a memoryless walk over a graph in that the choice of the next vertex depends only on the current vertex and not the previous history of the walk. In our case, from its current vertex, the walk will choose the next vertex by following an outgoing edge chosen uniformly at random.

A Markov chain can be described entirely by its transition matrix P, describing one step of the Markov chain as $X_{t+1} = X_t P$. This simplicity makes it relatively easy to analyze specific Markov chains, including our ideal random walk.

In particular, when the underlying graph is connected, undirected, and regular, we can show that our first desired property is satisfied: our random walk will converge to the uniform probability distribution. (The connectedness property ensures that the random walk has a limiting distribution, and undirectedness and regularity guarantee that this distribution is uniform.)

Unfortunately the natural web graph W initially meets none of these three conditions, but we can modify it to satisfy them. We first make each edge in W undirected, meaning that our random walk can now follow links either forwards or backwards. This makes the graph undirected. (It is connected because we consider only the indexable web.) It is still not regular, since some pages will have higher degree than others. This can be fixed by adding a different number of self-loops to each page so that every page ends up with the same number of links as the original vertex of maximum degree. We will denote this maximal degree as d.

The final graph W' thus fits all three requirements and our random walk over this graph will approach a uniform distribution. Since our walk is a Markov chain, we can associate it with a transition matrix $P_{W'}$. See Figure 2 for a complete description of the ideal walk.

2.4 The Ideal Walk: Feasibility

We now need to determine the ideal walk's *mixing time*, or the number of steps the walk must make before reaching a distribution acceptably close to uniform. A well known convergence theorem from the theory of Markov chains shows that for our case, the mixing time is bounded by $O(\frac{1}{\epsilon}\log N)$ steps, where N

```
Ideal_Visit($v$) {
    $I :=$ all in-neighbors of $v$

    $O :=$ all out-neighbors of $v$

    $w := d - |I \cup O|$

    Compute the number of self loop steps
    spent at $v$ (distributes geometrically
    with parameter $1 - \frac{w}{d}$)

    Select a random $u \in I \cup O$

    Ideal_Visit(u)
}
```

Figure 2: Ideal Walk's Visit Function

is the total number of pages in W'. Here ϵ is the *eigenvalue gap* of our Markov chain, or $|\lambda_1| - |\lambda_2|$, where λ_1 and λ_2 are the eigenvalues of $P_{W'}$ with largest and second largest absolute value. The eigenvalue gap is a measure of the conductance of the Markov chain; a large eigenvalue gap indicates that there are few bottlenecks or isolated parts of the graph. In this case the walk will avoid becoming trapped and approach the uniform distribution rapidly.

It is impractical to compute the eigenvalue gap of $P_{W'}$, since this would require complete knowledge of the web's structure. We do have access to a large crawl of the indexable web from 1996, however, and were able to use brute force to determine that the eigenvalue gap for the undirected regular graph there was $\epsilon = 10^{-5}$. If a similar figure is still true today, and estimating the current size of the web as approximately 10^9, the convergence theorem above shows that the mixing time of the ideal random walk is on the order of 3,000,000 steps. We will denote the mixing time as τ.

The prospect of our walk performing this many web accesses would seem a little daunting, but we are saved by the observation that most of the steps of the ideal walk are self-loops, which do not require a web access. Furthermore, we do not need to simulate each self-loop step separately. The number of consecutive self-loops our ideal walk takes at each page can be modeled by a geometric random variable, so that one calculation can simulate many steps of the actual walk. In fact, on the undirected regular 1996 graph, each page had a degree of 300,000, but an average of only ten links which were not self-loops. Thus only 1 in 30,000 steps of the random walk is not a self-loop and requires a web access, meaning that a 3,000,000 step walk requires only 100 actual web accesses, a very feasible number.

2.5 Subset Sampling Procedure

The above ideal random walk meets our initial requirements: it produces nearly uniform samples from the web and runs in a small amount of time. We now show how we can apply this to one of our main applications, estimating the relative size of subsets of the web. (We will also use the same procedure to estimate the results of aggregate queries.)

Suppose we have a subset A of all web pages V_W (with $|V_W| = N$) whose relative size we wish to estimate. The most obvious method would be to repeatedly run the walk T times for τ steps to obtain a nearly uniform distribution, and then choose the walk's current page as a sample point. While workable, this approach is not ideal. Aldous [10] proposes a more efficient sampling procedure: Instead of running T walks of length τ Aldous suggests we run only one walk of length $\tau + k$, and use the last k nodes as the sample points, disregarding the first τ steps as *sample delay.* Gilman [13] and Kahale [16] prove that $k = O(\frac{1}{\epsilon}\frac{1}{\beta^2\frac{|A|}{N}}\log\frac{1}{\delta})$ steps are sufficient to obtain a (β, δ) approximation. This means that after this many steps, we will have $Pr[(1-\beta)\frac{|A|}{N} \leq \frac{s}{k} \leq (1+\beta)\frac{|A|}{N}] \geq 1 - \delta$, where s is the number of sample pages that belong to A. This allows us to save a factor of $\log N$ random walk steps over the naive approach.

With the above estimates of the eigenvalue gap ϵ and the size of the web N, this result shows that if we wish to estimate the size of a subset A that comprises at least 20% of the web to within 10% accuracy with 99% confidence, we will need 350,000,000 random walk steps, of which only about 12,000 will not be self-loops.

2.6 Lessons from the Ideal Walk

What all of this demonstrates is that the ideal walk, if we could realize it, would provide a reliable and excellent tool for uniform random sampling and subset estimation on the web. Unfortunately we cannot implement this walk, since we cannot follow links backwards, a crucial step in our design. We now describe how our actual walk approximates this model, and the results we obtain from it.

3 WebWalker - Realization of a Random Walk on the Web

Realization of the ideal random walk scheme described in Section 2.3 requires the following primitives:

1. Determining the degree of a given page v (i.e., the number of incoming and outgoing links it has).

2. Given a page v and an index $i \in \{1, \ldots, d\}$, selecting the i^{th} neighbor of v.

The first primitive is needed for computing the self loop weight of v, and the second for choosing a random neighbor of v. A straightforward solution for both is to retrieve the list of all incoming and outgoing links of v, and then infer from this list the degree of v and the chosen random neighbor. Obtaining all the outgoing links of v is easy: we just have to extract the links from its HTML text. Getting the list of incoming links is, however, more intricate. We can acquire incoming links from two sources:

1. The walk itself - if the walk visits a page u with a link to v before it visits v, it can know of this link at the time it visits v.

2. Search engines - some search engines (AltaVista [2], Go [4], Google [5], HotBot [6], and Lycos [8]) allow one to retrieve all the pages they index that contain a link to a given page v.

These two sources of incoming links are not sufficient to implement the primitives mentioned above. First, they might miss some of the incoming links. Suppose there exists a link $u \to v$. If the walk does not visit u before it visits v and none of the search engines indexes u, then the walk will not be aware of the link $u \to v$ at the time it visits v. Second, the search engines return at most 1000 links for each queried page. Thus, for pages with a much larger number of incoming links we have access to only a small fraction of these links.

The inability to obtain all the incoming links of a page v, or even to know their exact number, prevents us from fully realizing the above primitives. This means that there is no obvious way to realize the ideal undirected regular random walk on the web. We now describe *WebWalker* - a new random walking process that attempts to come close to the ideal random walk, given only limited knowledge of incoming links.

3.1 WebWalker Specification

WebWalker performs a regular undirected random walk while exploring the web, using the above mentioned link resources (i.e., the HTML text, the walk itself, and the search engines). As WebWalker traverses the web it builds a d-regular undirected graph that is used to determine the next step of the walk. Each time WebWalker visits a new page it adds a corresponding node v to the graph. At this point we will determine and fix for the remainder of the walk the d edges (possibly including self-loops) incident to v. More explicitly, denote by $N(v)$ the set of v's neighbors that are available from the above resources the first time WebWalker visits v. WebWalker considers only pages in $N(v)$ to be neighbors of v throughout the walk. If it happens to discover a new link $u \to v$ later, it ignores this link and does not add u to $N(v)$. This is done in order to ensure *consistency* in the graph on which WebWalker walks: we want to make sure that the exact same set of neighbors is available to WebWalker every time it visits v. The importance of this feature will become apparent in Section 3.2.

The self loop weight assigned to v is $d - |N(v)|$. Each time WebWalker visits v, it first calculates the number of self loop steps spent at v, and then picks a random page in $N(v)$ to be visited next. Figure 3 specifies the Visit function run by WebWalker when it visits a page v.

```
WebWalker_Visit($v$) {
   If $v$ was already visited, skip to SELF

   $I :=$ $r$ random in-neighbors of $v$ from the
        ones returned by the search engines

   $O :=$ all out-neighbors of $v$

   For all $u \in (I \cup O) \setminus N(v)$ do {
       if $u$ was not visited yet {
          add $u$ to $N(v)$
          add $v$ to $N(u)$
       }
   }

SELF:
   $w := d - |N(v)|$
   Compute the number of self loop steps
   spent at $v$ (distributes geometrically
   with parameter $1 - \frac{w}{d}$)

SELECT:
   Select a random $u \in N(v)$
   If $u$ is ``bad'' go back to SELECT
   WebWalker_Visit(u)
}
```

Figure 3: WebWalker's Visit Function

We also choose the following implementation parameters for WebWalker:

- WebWalker's starting point is an arbitrary page in the largest strongly connected component of the web. We usually start from www.yahoo.com.

- WebWalker uses AltaVista and HotBot as sources for incoming links. For each visited page it retrieves up to 20 links from AltaVista and up to 100 links from HotBot.

- From the set of candidate in-neighbors of v returned by the search engines, WebWalker picks only r at random (r is a parameter of WebWalker). This is done in order to reduce bias towards pages covered by the search engines (see Sections 3.2 and 4.1 for details).

- The degree d with which WebWalker computes the self loop weight of each visited node needs to bound the web's maximal degree. Note that choosing a loose upper bound makes no difference, since it only increases the number of self loop steps. Therefore, we pick a crude overestimate of $d = 10,000,000$.

- We consider a page u to be "bad" (in which case we do not visit it, even if it is selected) in one of the following cases: (1) it is not a static HTML page, (2) we cannot establish a connection to it within 3 minutes, or (3) its URL address is longer than 300 characters.

3.2 WebWalker Analysis

We next analyze WebWalker's performance. We show that WebWalker is a Markovian random walk not on the web graph itself, but rather on a subgraph of the web. This subgraph is built on the fly as WebWalker explores the web. Since this subgraph is designed to be connected, undirected, and regular, WebWalker converges to a uniform stationary distribution over its nodes.

We then show that this subgraph always covers the whole indexable web, which means WebWalker can generate uniform samples from the indexable web. We finally address the question of how long it takes WebWalker to converge to the uniform stationary distribution. We do not have a satisfactory theoretical answer to this question, but we point out the factors that influence the mixing time. We show that in early stages of the walk WebWalker might suffer from biases towards high degree pages and pages covered by the search engines. Experiments (described in Section 4) show that in practice, these biases are small.

Assume we could run WebWalker infinitely long. Let G be a subgraph of the web that contains all the nodes visited and all the edges traversed by WebWalker during this run. We call G the *induced subgraph* of WebWalker's run.

G does not necessarily contain all the edges of the web. At the first time WebWalker visits a page v, it fixes its neighbor set $N(v)$, and ignores any further links to v it encounters later. Thus, only edges of the form (v, u) for $u \in N(v)$ will be traversed by WebWalker, and therefore included in G.

The subgraph G depends on the random choices made by WebWalker. For example, if v is a page that is not covered by any of the search engines and $v \to u$ is an outgoing edge from v, then only if WebWalker happens to visit v before it visits u, is $v \to u$ included in G. We conclude that G is a *random variable* that depends on the walk itself. Different runs of WebWalker may yield different induced subgraphs.

WebWalker behaves as a Markovian random walk on G. Furthermore, by definition G is connected, undirected, and regular. Therefore, WebWalker is guaranteed to converge to a uniform stationary distribution over its nodes. There are two questions left open: (1) what fraction of the web is G guaranteed to cover? and (2) how fast does WebWalker approach the uniform distribution over the nodes of G? The following proposition provides an encouraging answer to the first question.

Proposition 1 *In any infinite run of WebWalker, G covers the whole indexable web.*

Proof: Let v be some node in the indexable web. Thus, there exists a directed path $v_0, \ldots, v_k = v$ of "good" (i.e., not "bad") pages from WebWalker's starting point v_0 to v (since v_0 belongs to the largest strongly connected component). We prove by induction on i ($i = 0, \ldots, k$) that $v_i \in G$. It follows that in particular $v = v_k$ belongs to G.

The base case $i = 0$ is easy: WebWalker starts from v_0, therefore $v_0 \in G$. Assume $v_0, \ldots, v_i \in G$. Consider the edge (v_i, v_{i+1}). Since $v_i \in G$, v_i is visited by WebWalker. Let t be the first time WebWalker visits v_i. v_{i+1} will not be included in $N(v_i)$ only if it was visited before. But in this case v_{i+1} already belongs to G, and we are done.

Therefore, assume $v_{i+1} \in N(v_i)$. Since the graph G on which WebWalker walks is finite, undirected, and connected, in any infinite run WebWalker visits each node in G (and in particular, v_i) infinitely often. Since $N(v_i)$ is finite, all the nodes in $N(v_i)$ will be picked to be visited next at some point. When v_{i+1} is picked by WebWalker it is also visited, since v_{i+1} is a "good" page. Hence, $v_{i+1} \in G$. □

Note that G may also contain parts of the non-indexable web, if the search engines index these parts.

In order for WebWalker to be useful for sampling from the indexable web, we need to make sure that with high probability it converges quickly to the uniform stationary distribution over the vertex set of G. The eigenvalue gap analysis made for the web graph (see Section 2.4) does not hold for G, since G is only a subgraph of the web. If G misses many edges from the web its eigenvalue gap may be much smaller, implying a long mixing time for WebWalker. Unfortunately, we currently do not have any theoretical analysis of the mixing time of WebWalker as a random walk on the graph G.

We suspect that during its early stages WebWalker might be biased towards certain kinds of pages. We identified three sources of potential bias:

(1) Bias towards high degree nodes. The walk will tend to discover high degree nodes earlier than low degree nodes, because high degree nodes have many more short paths leading to them from the walk's starting point than low degree nodes.

(2) Bias towards nodes covered by the search engines. Since WebWalker uses some search engines as a source of incoming links, it is more likely to visit pages covered by these search engines than ones that are not.

(3) Bias towards the neighborhood of the walk's starting point.

The lack of theoretical foundation compels us to resort to experimental evaluation of WebWalker's convergence rate. Section 4.1 presents the results of such experiments, in which we ran WebWalker on a copy of the web from 1996. We use the above potential biases as indicators for evaluating how close WebWalker is to the uniform distribution (small biases indicate convergence).

4 Experiments

We performed experiments using WebWalker to estimate the size of certain subsets of webpages, and to estimate the answer to various aggregate queries about web pages. These experiments were run in February 2000. In addition we also ran WebWalker on a graph created from a large crawl of the web performed in 1996. Because we had access to all the nodes in the graph we can quantitatively evaluate WebWalker's performance and biases on the 1996 web graph.

4.1 Evaluation Experiments

In order to observe WebWalker's effectiveness and bias, we performed a 100,000 page walk on the 1996 web graph. Ideally the samples from this walk, when weighted by self-loops, would be uniformly distributed over all nodes. In order to see how close the samples were to uniform, we compared them to a number of known sets. The sets we considered were: 10 sets each containing one decile of nodes ordered by degree, 10 sets each containing one decile of nodes in breadth first search order from the starting node for WebWalker, and the set of nodes contained in the search engine (see below) used for incoming links.

As discussed in Section 3.2, WebWalker may suffer from biases at early stages of the walk. The experiments were designed to address three sources of possible bias: (1) bias towards nodes with high degree in the web graph, (2) bias towards the neighborhood of WebWalker's starting page, and (3) bias towards nodes indexed by the search engine(s) used for incoming links.

An obstacle to running WebWalker on the 1996 web graph is that we do not know which of the 1996 pages were covered by the search engines at that time. Instead, we designate an arbitrary subset of the nodes as the search engine index. Thus, the incoming edges WebWalker discovers come from nodes already visited and from nodes in the designated search engine subset. Unless stated otherwise the experiments in this section used 50% of the 1996 web graph as the search engine set.

The experimental results shown in this section also show the effect of changing r, the number of incoming links we take from the search engine for each visited page (see Section 3.1). WebWalker was run using $r = 0$, $r = 3$, and $r = 10$.

Figure 4 shows the distribution of nodes from two walks accumulated into bins by degree. The nodes in the 1996 web graph are sorted by degree and then separated into 10 deciles, each containing 10% of the nodes. For each decile the bar chart shows the percentage of nodes in a walk that had a degree in the range specified. More or fewer than 10% of the nodes in the walk in a particular range shows a bias in the walk. From this graph we see that 24% of the walk with $r = 0$ was spent in the top 10%, an overestimate of 14%. By taking $r = 3$ we see that the bias towards the top 10% of nodes is decreased to an overestimate of 9%. We also notice for both walks that the bias is concentrated in the top 10% of nodes by degree. There is relatively little bias in the other deciles, and even the nodes in the lowest decile are estimated well.

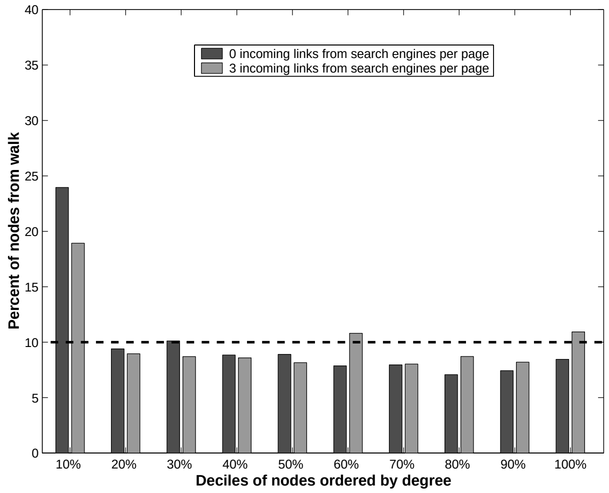

Figure 4: Percent of nodes in two walks on the 1996 web graph by decile of the nodes ordered by degree. For example, the leftmost bar indicates that 24% of the pages in the walk using $r = 0$ were among nodes ranked in the top 10% by degree in the 1996 web graph.

Figure 5 shows the nodes of two walks divided into neighborhoods of WebWalker's starting point. The two walks shown use parameters $r = 0$ and $r = 3$, respectively. All the nodes in the 1996 web graph are ordered by a BFS starting at WebWalker's initial node. They are then divided into 10 equal sized, consecutive sets. The small variations from a uniform distribution indicate that WebWalker has little bias toward its starting neighborhood.

Figure 6 shows the percent of nodes from each of six walks that were also in the search engine for each walk. The first three use 30% of the 1996 web graph as a search engine, and various values for r. The last three use 50% of the 1996 web graph as a search engine. We see from the results that there is a small bias toward

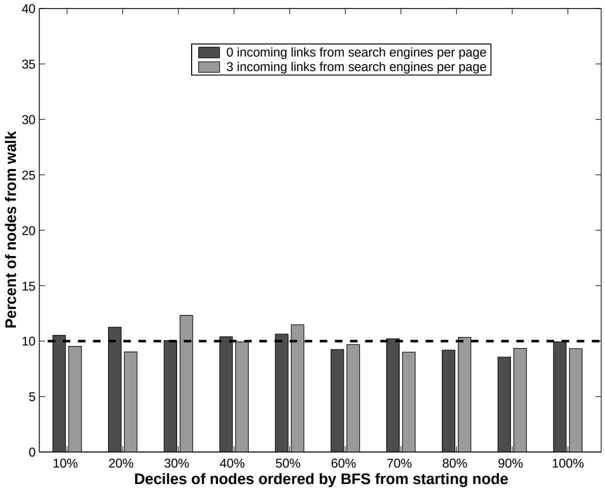

Figure 5: Percent of nodes in the walk on the 1996 web graph by decile of nodes ordered by breadth first search distance from the walk's starting node. For example, the leftmost bar indicates that 10.5% of the nodes in the walk using $r = 0$ were among the first 10% of all nodes when ordered by a BFS starting with WebWalker's initial node.

the search engine, and that the bias increases as more incoming links are taken from the search engine.

We have observed in our experiments that increasing the number of links taken from a search engine *increases* the bias toward that search engine, an undesirable effect. At the same time increasing the number of incoming links taken from search engines *decreases* the bias toward high degree nodes. Based on the experiments on the 1996 web graph we found $r = 3$ to be a good compromise.

We conclude that from the biases we examined that only the bias towards the highest degree nodes is significant.

4.2 Subset Size and Aggregate Query Approximation Experiments

Next, we show a flavor of the approximations of web subsets one can perform using WebWalker. We ran WebWalker on the current web (February 2000) and tried to approximate the relative size of the following subsets:

(1) The pages that can be returned as query results by AltaVista [2]. WebWalker uses AltaVista as a source for incoming links. We wished to check the effect this has on the approximation of AltaVista.

(2) The pages that can be returned as query results by the FAST search engine [3] (formerly called "All the Web") WebWalker does not use FAST as a source for incoming links.

(3) The overlap of AltaVista and FAST (pages that are covered by both).

(4) The pages that contain inline images.

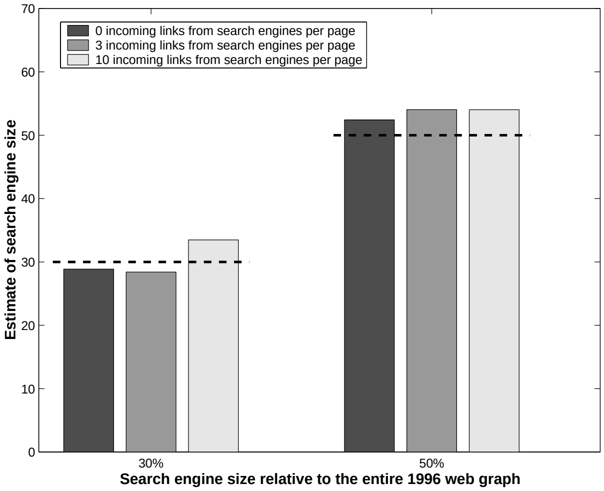

Figure 6: Percent of nodes in six walks that are also contained in the search engine used for incoming links in those walk. For example, the leftmost bar indicates that 28.9% of the nodes in the $r = 0$ walk were actually in the search engine comprised of 30% of all nodes in the 1996 web graph.

In order to check whether a given sample page belongs to a search engine (e.g., FAST or AltaVista) we extract a list of features from the page (its URL and title and the list of phrases it contains) and repeatedly query the search engine for pages that contain one of these features. We determine that the page is covered by the search engine if we find the exact URL (after normalization) of the page in the results of one of these queries. Note that if the search engine keeps only one copy for mirrored sites, this method might determine that a page is not included in the search engine index, although it actually is (it is simply not returned by the search engine as a result of a query). Therefore, the actual coverage of the search engine might be somewhat larger than what is reflected by our approximations.

The approximations we obtained for the relative size of FAST, AltaVista, their intersection, and the image set are presented in Table 1. Reports for the FAST and AltaVista indexes [9] were about 300 million pages and 250 million pages respectively. This means that the ratio between them should be about 1.2. Our approximations come close with a ratio of 1.14. This may indicate that the bias of WebWalker towards AltaVista (compared to FAST) is not significant. These results were obtained from three walks with a total of 41,384 page accesses.

In addition we wanted to evaluate the following aggregate queries about the web:

(1) The distribution of pages according to the last part of their domain name (com, edu, org, and others)

(2) The average size of static HTML pages.

(3) The average number of hyperlinks in static HTML pages.

Set	Size Approximation
FAST	39.97%
AltaVista	35.46%
FAST ∩ AltaVista	21.13%
Images	72.60%

Table 1: Web Subset Approximations

The approximations we obtain for domain name distributions are based on a combination of 21 runs of WebWalker, with a total of 146,133 page accesses. To validate the accuracy of the resulting approximations we compare them against similar tests made in February, 2000 by Inktomi [7], which based their estimations on an extensive 1 billion page index of the web.[2]

The 14 largest domains (and .mil) together with their relative sizes are presented in Table 2. We give the corresponding Inktomi approximations, when available, for reference.

Domain	Our Approximation	Inktomi Approximation
.com	49.15%	54.68%
.edu	8.28%	6.69%
.org	6.55%	4.35%
.net	5.60%	7.82%
.de	3.67%	
.jp	2.87%	
.uk	2.75%	
.gov	2.08%	1.15%
.ca	1.58%	
.au	1.37%	
.us	1.12%	
.fr	1.01%	
.se	0.98%	
.it	0.94%	
...		
.mil	0.19%	0.17%

Table 2: Largest Internet Domains

The average page size and the average number of links in a page are presented in Table 3.

Query	Approximation
Average size (in bytes)	11,655
Average number of hyperlinks	9.56

Table 3: Aggregate Query Approximations for Static HTML Pages

5 Related Work

Several recent attempts have been made to estimate the size and/or quality of various search engines' indexes. All of this previous work requires obtaining a sample of pages in some way. In some cases the idea is to obtain a uniformly distributed sample, while in others it is to find a sample weighted by a particular metric.

Henzinger et al. [15] used a large number of random walks designed to converge to Google's [5] *page rank* distribution over the nodes walked. This distribution favors popular pages. They then attempt to remove the bias by sampling pages from the walk based on the inverse of an estimate for the page rank of each page. They used a random graph model for the web, and produced data showing a clear bias towards high degree nodes. This bias is difficult to compare with our own since the random graph model used did not exhibit many properties of the 1996 web graph we used. For instance the 1996 web graph has significantly higher maximum degree, which could potentially cause a much larger bias in their walk. Their results on the web also indicate a bias towards Google, possibly the result of a bias toward pages with a high page rank. Their approach seems to require many more page accesses to get approximations than the one presented in this paper. This work was carried out independently to our own, over a similar time-frame, and is an extension of their previous work on approximating page rank using random walks [14].

Bharat and Broder [11] attempted to generate random pages in a search engine index by constructing random queries. These pages were then used to estimate the relative size and overlap of various search engines. Their queries were based on a dictionary of words collected from Yahoo!, and the resulting sample was therefore biased towards English language pages. Also, because they combined several terms in their queries, they note a bias towards content rich pages.

Lawrence and Giles [17, 18] report the sizes of several search engines based on the number of pages returned in response to 1,050 actual queries. They note that the resulting pages are not uniformly distributed and use the results to discuss what might be called the useful size of a search engine, instead of measuring the total number of pages indexed. In addition

[2] Note that in order to generate this index, Inktomi needed powerful computational and network resources and four months of work.

they attempted to approximate the size of the web by estimating the number of web servers (via sampling random IP addresses and querying for a server) and multiplying by an estimate for the number of pages per site. They note, as do others, that the distribution of pages per server has a very high variance, and that their estimates are susceptible to inaccuracy as a result.

6 Conclusions

We have presented an efficient and accurate method for estimating the results of aggregate queries on web pages in the indexable web. We achieve this by performing a carefully designed random walk to produce a close to uniformly distributed sample of web pages. We have validated the walk in extensive experiments on the web today as well as on a large portion of the web from 1996 obtained by an ALEXA crawl. Furthermore, our technique is extremely efficient, and can produce uniformly distributed web pages quickly without significant computation or network resources. In fact, our estimates are reproducible on a single PC with a modest connection to the Internet in as little as one to two days.

One particularly interesting application of our technique is estimating the fraction of web pages covered by a particular search engine. This can be extended to a less biased mechanism for comparing search engine coverage. Furthermore, with accurate knowledge of the absolute size of a search engine, we can estimate the size of the web.

This technique opens up many new interesting questions. We would like to obtain stronger theoretical analysis of WebWalker's mixing time to support our experimental results. In addition, we would like to further reduce the bias towards high degree nodes and pages indexed by the search engine.

Acknowledgements

Sridhar Rajagopalan introduced us to this subject, provided us with the 1996 copy of the web, and gave us his constant support.

Thanks also to Alistair Sinclair and David Gibson for helpful discussions, and to Randy Huang and David Gibson for technical assistance.

References

[1] Alexa. `http://www.alexa.com`.

[2] AltaVista. `http://www.altavista.com`.

[3] fast. `http://www.alltheweb.com`.

[4] Go. `http://www.go.com`.

[5] Google. `http://www.google.com`.

[6] HotBot. `http://www.hotbot.com`.

[7] Inktomi. `http://www.inktomi.com`.

[8] Lycos. `http://www.lycos.com`.

[9] Search Engine Watch. `http://searchenginewatch.com`.

[10] D. Aldous. On the Markov chain simulation method for uniform combinatorial distributed and simulated annealing. *Probability in the Engineering and Informational Sciences*, 1:33–46, 1987.

[11] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public Web search engines. In *Proceedings of the 7^{th} International World Wide Web Conference (WWW7)*, pages 379–388, April 1998.

[12] A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: experiments and models. In *Proceedings of the 9^{th} International World Wide Web Conference (WWW9)*, May 2000.

[13] D. Gilman. A Chernoff bound for random walks on expander graphs. *SIAM J. on Computing*, 27(4):1203–1220, 1998.

[14] M.R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najrok. Measuring index quality using random walks on the Web. In *Proceedings of the 8^{th} International World Wide Web Conference (WWW8)*, pages 213–225, May 1999.

[15] M.R. Henzinger, A. Heydon, M. Mitzenmacher, and M. Najrok. On Near-Uniform URL Sampling. In *Proceedings of the 9^{th} International World Wide Web Conference (WWW9)*, pages 295–308, May 2000.

[16] N. Kahale. Large deviation for Markov chains. *Combinatorics, Probability and Computing*, 6:465–474, 1997.

[17] S. Lawrence and C.L. Giles. Searching the World Wide Web. *Science*, 5360(280):98, 1998.

[18] S. Lawrence and C.L. Giles. Accessibility of information on the web. *Nature*, 400:107–109, 1999.