

# A Discussion of Click-Through Algorithms for Web Page Ranking

May 2005  
Brad Null

## 1 Introduction

The aim of this paper is to develop a reasonable model of web page ranking for a web search engine incorporating information derived from actual engine usage. Such a model could be used to check, replace, or complement the rankings delivered by existing algorithms such as PageRank.

The model we propose assumes users examine the pages search engines feed them in order, and for any page  $i$ , every user that examines that page has a probability of clicking-through to that page of  $p_i$ .

In our analysis we discuss the value to the search engine of knowing this  $k$ -dimensional vector  $p$  of click-through probabilities of the  $k$  pages, specifically how knowing the  $p$  vector enables us to quite easily maximize the probability of a click-through for any search engine query and minimize the expected number of page views for any user before a click-through. Further, we evaluate how many user queries must be observed before  $p$  is known within  $\epsilon$  at 95% confidence, and how pages should be ordered for these queries in order to converge to  $p$  as quickly as possible.

Finally, we discuss the performance of this algorithm under usage models that vary from our assumptions, and we highlight a number of questions to address in continuing research.

## 2 Motivation for a click-through algorithm

The primary benefit of a click-through algorithm for web page ranking is that it incorporates actual user click-through behavior to rank web pages. This contrasts to the popular link-analysis algorithms, such as PageRank, which do not build their rankings off of any actual usage but instead off of the underlying linked structure of the network of web pages on the Internet.

The click-through model we propose herein specifically incorporates click-throughs from the search engine that is incorporating the rankings. Our motivation here is based upon the following idea. The primary users of web page ranking systems are search engines. The objective of a search engine is to help users of that search engine find the pages they are looking for. Thus, given this objective, we would like to identify those pages that visitors to the search engine find most valuable. It would make sense to assume that these are the pages that those visitors click-through to when they visit the engine.

Thus, we bypass the somewhat indirect logic of link analysis and the reputation system it is based on, which values pages based upon an apparent reputation in the world of web pages. Here we get directly to the point, attempting to answer the question, what pages do the *people* that visit our search engine value?

Once we have answered this question satisfactorily, it might then be interesting to go back and validate, critique, or complement the link analysis rankings with our results. We will examine that alternative in the sequel to this analysis.

### 3 A model of web pages and web engine usage

We begin by modelling the system. We assume:

1. There are  $k$  web pages.
2. The search engine cannot distinguish between pages by topic, thus every user query is equally relevant to all pages.
3. Each page  $i$  has an inherent value parameter  $p_i$  which represents the probability that any given user, upon examining page  $i$ 's listing on the search engine, will click-through to page  $i$ .
4. The search engine produces for each user an ordered list of pages. Users examine these pages in order until they decide to click-through to a page. Once the user clicks-through to that page, the user is done.
5. Users will continue examining pages until they have either clicked-through to a page, or rejected all pages.

### 4 Determining our objective function

Assume the pages are ordered by the search engine in order of their indices: 1,2,3,... Then the following two values represent the expected probability that users eventually click-through and the expected number of pages views per user until a click-through.

$$E[\text{probability of success}] = p_1 + (1 - p_1)[p_2 + (1 - p_2)[p_3 + \dots$$

$$E[\text{search time}] = 1 + (1 - p_1)[1 + (1 - p_2)[1 + \dots$$

It is reasonable to assume that maximizing the first of these values and minimizing the second are both primary objectives for a search engine.

Now, obviously with respect to our model and assumption 5) above, the probability of a click-through will be the same every time. However, if we relax assumption 5), this quantity does not remain constant as we reorder pages. Likewise, the expected number of page views changes as we reorder pages, regardless of assumption 5).

In any case, if we can simply identify the  $p_i$  values then we can optimize with respect to both of these objectives by simply ordering the  $p_i$  values in decreasing order. The justification for this can be seen quite clearly in the two simplified equations below.

Notice, the probability of a click-through in  $m$  steps can be rewritten

$$E[\text{probability of success}] = 1 - (1 - p_1)(1 - p_2)\dots(1 - p_m)$$

This value is decreasing in  $p_i$ , so we want the highest  $p_i$ 's included for all sets of  $m$  steps. Thus, we want  $p_i$  ranked in decreasing order to maximize this probability for all  $m$ .

Expected number of page examinations can also be rewritten as

$$E[\text{search time}] = 1 + (1 - p_1) + (1 - p_1)(1 - p_2) + \dots + (1 - p_1)\dots(1 - p_k)$$

so for any ordering of the pages, if you swap pages  $i$  and  $j$  where  $i$  was originally placed before  $j$ , the only terms in this sum that change are those that include a  $p_i$  term and no  $p_j$  term. These terms all decrease if  $p_j > p_i$  and increase if  $p_j < p_i$ . Thus, to minimize expected number of page examinations, we must order by largest  $p_i$ .

## 5 Determination of the $p$ vector

Having established the value inherent in knowing  $p$ , we must now estimate how much work is involved in finding  $p$ . More specifically, for our purposes, we will say it is good enough to determine each  $p_i$  within  $\pm\epsilon$  at 95% confidence.

### 5.1 Examinations of page $i$ needed to determine $p_i$ within $\pm\epsilon$

First let us consider a single page. How many examinations of that page do we need to get an estimate of  $p_i$  within  $\epsilon$  at the 95% confidence level?

By modelling each yes/no click-through decision,  $x_i$ , as an independent Bernoulli trial, (where we say  $x_i = 1$  if clicked-through and  $x_i = 0$  if passed over) the variance of one page examination is  $p_i(1 - p_i) \leq .25$ . Then by the Central Limit Theorem the standard deviation of a sample of  $n$  page examinations of page  $i$  will be

$$\sqrt{\frac{p_i(1 - p_i)}{n}}$$

For a 95% confidence interval with epsilon width, we need

$$z_{.95} \sqrt{\frac{p_i(1 - p_i)}{n}} \leq \epsilon$$
$$\Rightarrow n \geq \frac{z_{.95}^2 p_i(1 - p_i)}{\epsilon^2}$$

where  $z$  is the two-tail  $z$ -test value for 95% or about 2, so we need approximately  $\frac{4p_i(1-p_i)}{\epsilon^2}$  page examinations. This value is maximized where  $p_i = .5$ , so in the worst case we would need  $\frac{1}{\epsilon^2}$  examinations of page  $i$  to estimate  $p_i$  within  $\pm\epsilon$ .

### 5.2 Queries to the search engine required to estimate $p$ within $\pm\epsilon$

Given the minimum number of page examinations needed for a single page calculated above, we can proceed to calculate the minimum number of click-throughs for a given page as just  $p_i$  times this quantity, or

$$\frac{4p_i^2(1 - p_i)}{\epsilon^2}$$

So to estimate the entire  $p$  vector, we would need at least

$$\sum_{i=1}^k \frac{4p_i^2(1 - p_i)}{\epsilon^2}$$

click-throughs. This implies that we need at least this many search engine queries.

This quantity is maximized where  $p_i = \frac{2}{3} \forall i$ . So in the worst case, we would need at least  $\frac{4k}{27\epsilon^2}$  search engine queries to determine  $p \pm \epsilon$ .

We leave as an open question the issue of how to order pages so that we could find  $p \pm \epsilon$  within the minimum number of steps, but a couple of obvious ideas present themselves.

1. One possible approach would be to always order the pages in descending order of the expected number of trials needed  $r_i$ , where  $r_i = \frac{4\hat{p}_i(1-\hat{p}_i)}{\epsilon^2} - n_i^{obs}$ ,  $\hat{p}_i$  is our current estimate of  $p_i$  and  $n_i^{obs}$  is the total number of examinations of page  $i$  at present.
2. A second approach that takes advantage of the fact that we expect to get more page examinations out of a query if we have pages with lower  $p_i$  ranked early, suggests ordering all pages that still need a significant number of observations in increasing order of  $\hat{p}_i$ . Interestingly, this order is the opposite of the order we intend to use in the ultimate operation of the search engine. We will discuss the potential ramifications of this fact later.

### 5.3 Finding the highest $p_i$ value

Additionally, one might ask, instead of determining the entire  $p$  vector, how long does it take to just find the highest  $p_i$  (or at least one within  $\epsilon$ )?

Well, in the worst case, where all  $p_i$  values are very close together, we can't do this any quicker than finding the whole vector.

However, let us consider a situation where the highest  $p_i = p_1$  and all other  $p_i = p_x$  where  $p_1$  is significantly higher than  $p_x$ . To determine  $p_1 > p_i$  for any  $i$  we need 95% confidence that  $p_1 - p_i \geq 0$  (or  $-\epsilon$ ).

So we need a number of examinations of page 1,  $n_1$ , and page  $i$ ,  $n_i$ , such that

$$\frac{p_1(1-p_1)}{n_1} + \frac{p_i(1-p_i)}{n_i} \leq \left(\frac{p_1-p_i}{2}\right)^2$$

As we must perform  $k$  such comparisons, the minimum number of page examinations needed is the solution to the following optimization problem.

$$\begin{aligned} \min N &= n_1 + (k-1)n_i \\ \text{s.t. } \frac{p_1(1-p_1)}{n_1} + \frac{p_i(1-p_i)}{n_i} &\leq \frac{(p_1-p_i)^2}{4} \\ n_1, n_i &> 0 \end{aligned}$$

The constraints show that we need

$$n_i \geq \frac{4p_i(1-p_i)}{(p_1-p_i)^2}$$

and

$$n_1 \geq \frac{p_1(1-p_1)}{\frac{(p_1-p_i)^2}{4} - \frac{p_i(1-p_i)}{n_i}}$$

We will leave further analysis of how this impacts the size of  $N$  for the sequel.

## 6 Considering other usage models

It is always important to check the assumptions of a model. And an important assumption of this model is the assumption about how users behave. So let's take a moment to consider departures from this model and examine its performance under different usage assumptions.

## 6.1 The Random Surfer

The simplest alternate model we might consider is that of the Random Surfer. What if people really are Random Surfers? What will our model converge to?

Given that the Random Surfer is not well-defined with respect to a search engine interface, let's consider three models of the Random Surfer and see how he impacts our web page parameter  $p$ :

1. The Random Surfer always clicks through on the first page he is given.
2. The Random Surfer clicks through on each page he examines with some fixed probability  $\psi$  (e.g.  $\psi = .5$ ).
3. The Random Surfer clicks onto any of the pages on the first page with equal probability.

It is easy to see that in the first two scenarios we would respectively converge to  $\hat{p}_i = 1$  and  $\hat{p}_i = \psi$  for all pages.

In the third model however, the  $\hat{p}_i$  values might get skewed depending upon our order rule. For example, if we deterministically place the same set of pages in the same spots over and over again, then our results would be biased against pages that were more frequently placed near the top of the page. These pages would end up with lower  $\hat{p}_i$  values than pages placed near the bottom even though intrinsically this Random Surfer is indifferent amongst all pages.

This result arises from the fact that this selection process for the Random Surfer violates our assumption about how users select pages (i.e. sequentially), and it leads us towards an idea of ensuring, for robustness, that there be some randomization among ranks as well. While our model claims ranks do not matter, it is quite possible that in some application (such as this one) they do, and this should be dealt with.

On a side note, it is important to realize that regardless of what  $\hat{p}_i$  values come out of this evaluation, the search engine will still always have the same performance with respect to the objective of minimizing the expected time until a click-through. So even though our algorithm does not necessarily converge to a "fair"  $\hat{p}_i$ , it is still optimal with respect to our ultimate objective of minimizing the expected time to a click-through.

## 6.2 A two type model

Now let's look at a slightly more complicated model of user behavior and see how our model deals with it.

We assume there are two types of users, connoisseurs and lemmings. Lemmings constitute the fraction  $a$  of all users and lemmings always click-through to the first page they are given (thus corresponding to one of our models of the Random Surfer). Connoisseurs, the other  $1 - a$  of all users, have a  $k$ -dimensional vector  $p^c$  that determines the probability they will click-through to any page they examine (i.e. they click-through to page  $i$  upon examining it with probability  $p_i^c$ ). This sort of model also easily generalizes to a model with many different kinds of users with different click-through probabilities for different pages.

Let's look at a very basic example under two different ordering rules. We assume  $k = 3$ ,  $a = \frac{1}{2}$ , and  $p^c = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ . In the first choice rule we randomly assign the pages to ranks in the search engine. Our model predicts under this scenario that when placed first, each page has a  $a + (1 - a)p_i^c$  probability of getting clicked-through and a  $(1 - a)(1 - p_i^c)$  probability of getting passed. The probabilities of getting clicked and passed when placed second and third can be found similarly, and the overall  $\hat{p}$  vector can be determined by a weighted average of these results. Under our specifications here,  $\hat{p}_i \rightarrow .622$  for all  $i$ .

Now let's look at a simple biased ranking scheme where pages are always ranked in the order 1,2,3. In this case, the  $\hat{p}$  vector converges to  $(\frac{2}{3}, \frac{1}{3}, \frac{1}{3})$ . So with this usage model we can see that ranking matters, and in this case, there is bias towards the top of the list.

Let's also consider briefly a model where every user has a set of acceptable pages and waits until he sees one of them to click. Let's take a simple example with  $k = 3$  and 4 types of users. Lemmings occur with probability  $\frac{1}{2}$ , and connoisseurs that prefer only page  $i$  occur with probability  $\frac{1}{6}$  for all  $i$ . When we randomize rankings, this model converges to  $\hat{p} = (\frac{2}{3}, \frac{2}{3}, \frac{2}{3})$ , which seems like a reasonable result. When we present the same biased ordering as in our last example, we converge to  $\hat{p} = (\frac{2}{3}, \frac{1}{2}, 1)$ . Surprisingly, in this case there is bias towards the top and bottom of the list!

All in all, these results indicate that under different usage assumptions, our model may demonstrate bias towards certain rankings. Obviously we would want to neutralize this bias. Our preliminary results indicate some randomization over ranks as a possibly attractive way of accomplishing this.

## 7 Extensions

Below we briefly discuss a few extensions that could help in incorporating a model such as this into a real-world setting.

### 7.1 An ongoing approach framework

Above we have only discussed how to essentially initialize the system by estimating  $p$ . But to do this we have to interact with users, who might not be very eager to participate in such experiments which make no effort to give them good performance in the interim. At one extreme, consider the approach offered above of ordering pages in inverse order of  $\hat{p}_i$  until  $\hat{p}_i$  is well-estimated. This will lead to very poor perceived performance on the part of the user throughout the experiment. The question becomes, are we really going to be able to carry out this whole initialization process if perceived performance throughout the process is so bad? For example, if  $\epsilon = .01$  and  $p_i \approx .5 \forall i$  then we need approximately 10,000 observations for each page, and will get on average 2 observations per user query. If we assume we have 1000 pages, then we will have to have 5 million queries before all probabilities are estimated with confidence to within 1%. The concern is that people will get annoyed with the engine's apparently poor performance, and the search engine will end up losing customers over time.

So how can a search engine get this information over time without sacrificing too much in the way of performance? To model this problem we could set up user utility as a function of the number of pages viewed until a click-through. Our long-run objective is then to maximize the average user utility. Looking at the infinite horizon version of this problem, it makes sense to perform something like the analysis above as a sort of phase one process. However, as the search engine runs the risk of losing business, constraints might have to be added in order to weight the early periods sufficiently. Some ideas for doing this by altering this optimization problem are:

- incorporate a decay factor that weights current users and near-term users more heavily.
- add a constraint that the expected utility of every user must be greater than or equal to some threshold.
- add a constraint that the *realized* utility of users over any or all periods must be greater than or equal to some threshold. (Note: here we are dynamically altering our constraints based upon realized performance.)

## 7.2 Distinguishing among pages by content

One of the assumptions we made above was that all users are looking for the same thing. Of course, in practice this is not true. One idea for dealing with this is to incorporate a similarity measure  $\phi \in [0, 1]$  between a page and a query that would indicate that pages with this level of similarity are  $\phi$  as likely to get selected as pages that perfectly match the query.

To normalize  $\phi$  we would obviously require extensive historical analysis. (Which is to say nothing about deriving a measure of similarity.) But with an accurate  $\phi$ , then for every examination of  $i$  we could weight our sample result  $x_i$  as  $\tilde{x}_i = \frac{x_i}{\phi}$ . Thus the expectation of  $\tilde{x}_i$  would be  $p_i$  and these observations would not introduce bias.

*Note: These observations will have different variance depending upon  $\phi$  (increasing as  $\phi$  decreases), but nonetheless they can be factored into our estimate of  $p$  using a variance minimization technique a la portfolio theory to help derive the best overall estimate of  $p$  for each page.*

## 7.3 What about partial satisfaction?

We might also want to incorporate the concept that a user can derive partial satisfaction from a web page. Assuming that we can measure that level of satisfaction (on a  $[0, 1]$  scale), we might incorporate this partial satisfaction by giving that page a partial click-through with respect to this observation. As for how to determine partial satisfaction, we might simply conduct surveys or analyze time spent on pages and click-backs.

## 7.4 What about Link Analysis and the web structure?

As a final salvo, we ask the questions: could we incorporate link-analysis at all?

The answer to this question is certainly, yes. One idea is to acquire additional data on where users go after they click-through, build estimates of the values of those links in the web graphs, and run something like PageRank using those estimated link-values and a normalization of our  $\hat{p}$  vector as the reset probabilities. It would be very interesting to see how these results compared to the rankings of our model and to those of PageRank.

# 8 Conclusions

We assert that we have established herein a reasonable click-through algorithm for web page ranking that responds well to our intuition about how search engines are used. In addition, there are a number of generalizations that we can and should incorporate into the model to make it suitable to a real world setting.

There are, however, a couple of areas of concern:

- Initialization of our  $p$  vector is not easy. It takes  $O(\frac{k}{\epsilon^2})$  time to estimate the  $p$  vector.
- The model is sensitive to our assumption that all users have a uniform probability of clicking-through to any page. This sensitivity must be taken into account when we construct our order rule. It seems very likely that we will want to enforce a certain level of randomization among rankings by the search engine in order to avoid a sort of self-fulfilling prophecy.