

# Tracking the Changes of Dynamic Web Pages in the Existence of URL Rewriting

Ping-Jer Yeh

Jie-Tsung Li

Shyan-Ming Yuan

Department of Computer Science  
National Chiao Tung University  
1001 Ta Hsueh Road, Hsinchu 300, Taiwan  
Email: {pjyeh, jtli, smyuan}@cis.nctu.edu.tw

## Abstract

Crawlers in a knowledge management system need to collect and archive documents from websites, and also track the change status of these documents. However, the existence of URL rewriting mechanism raises a page tracking problem since the URLs of a pair of dynamic page instances obtained during different sessions will no longer be the same. This paper proposes a series of algorithms in a bottom-up manner to find the corresponding pairs of dynamic page instances, and then to judge the change status of them. Experiments showed that the performance was very good and the outcome was 100% accurate.

*Keywords:* URL rewriting, crawler, HTTP session, string matching.

## 1 Introduction

A knowledge management system (KMS) needs to collect and archive numerical and textual data from a variety of sources. It also tracks the status of the data so that users can be notified whenever any update has occurred since last time. Nowadays a growing number of data is exposed to the Internet, particularly the Web, e.g., newspapers, technical reports, business intelligence, patent databases, discussion forums, and stock prices. Consequently, the data acquisition module in the KMS (often called the crawler) has to consider the unique characteristics of Web pages, particularly dynamic pages.

This section introduces the challenges faced by crawlers when they try to track dynamic Web pages.

### 1.1 HTTP Session

Since HTTP is a stateless protocol (Fielding, Gettys, Mogul, Frystyk, Masinter, Leach & Berners-Lee 1999), there are some workaround mechanisms trying to simulate a stateful connection (or more formally, *session*) on top of HTTP. Among them, the *cookie* mechanism (Kristol & Montulli 2000, Kristol 2001) is a popular way to enable a series of stateful request-response interactions between Web clients and servers. It is very easy to use, but it is also very prone to abuse, snoop, and attack (Sit & Fu 2001). Therefore, a safer way to use cookies is proposed. That is, to encode and transmit only the session identifier (abbreviated as *session-id* or *sid*) during the lifetime of the session (Hallam-Baker 1996).

Copyright ©2006, Australian Computer Society, Inc. This paper appeared at Australasian Data Mining Conference (AusDM 2006), Sydney, December 2006. Conferences in Research and Practice in Information Technology (CRPIT), Vol. 61. Peter Christen, Paul Kennedy, Jiuyong Li, Simeon Simoff and Graham Williams, Ed. Reproduction for academic, not-for profit purposes permitted provided this text is included.

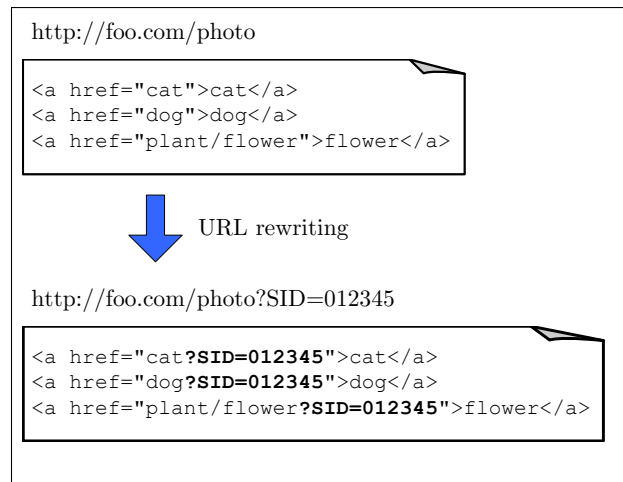


Figure 1: An example illustrating the URL rewriting effect

The use of cookie + session-id combination is now a de facto or default session mechanism in major Web servers and server-side script engines. For example, Java servlets/JSP generates a JSESSIONID cookie to carry the session-id (Coward & Yoshida 2004). By default the cookie is named ASP.NET.SessionId in ASP.NET, PHPSESSID in PHP, and `_session_id` in Ruby on Rails.

Despite the simplicity and popularity, the bad reputation of cookies has made many power users disable the cookie function of Web browsers entirely, at the risk of disabling the subsequent session tracking at the same time, though. To solve this anti-cookie dilemma, another transparent way to place and transmit the session-id is devised. When a new session starts, the server generates a corresponding session-id and then, before sending the dynamic page back to the client, inserts the session-id as a part of relevant URLs rather than places it in the cookies. This so-called *URL rewriting* step is applied on the fly to all subsequent URL links in the dynamic pages without the page author's awareness. Take Figure 1 for example. In the beginning a user requests the dynamic page `http://foo.com/photo`. If the session-id is 012345, the server will rewrite all the relevant links by inserting into them some formatted string like `SID=012345`. From the user's point of view, all links in the dynamic page have such session-id string embedded, and therefore all subsequent navigation through the page will naturally carry the same session-id information. In this way the server knows which session to track for.

Different server-side script engines rewrite URLs in quite different ways. Take again the "cat" link in Figure 1 for example. In Java servlets/JSP the link would be rewritten as

`http://foo.com/photo/cat;jsessionid=sid` (Coward & Yoshida 2004). By default the link would become `http://foo.com/photo/(sid)/cat` in ASP.NET's cookieless mode, `http://foo.com/photo/cat?PHPSESSID=sid` in PHP's transparent sid mode.

## 1.2 Challenges of Changing URLs

The use of URL rewriting raises a problem for crawlers. For a crawler to determine whether a page has been changed since last time, it has to be sure that it can access and identify the same page source, albeit some parts of the page content may not be identical. For example, if the content of page  $p$  was first  $p_{v1}$  and later changed to  $p_{v2}$ , the crawler has to identify that both  $p_{v1}$  and  $p_{v2}$  are two instances/versions of the same  $p$ . With the existence of URL rewriting, however, instances  $p_{v1}$  and  $p_{v2}$  are retrieved through different session-ids, and so do their URLs  $l(p_{v1})$  and  $l(p_{v2})$ . The crawler would, therefore, have difficulty in identifying that  $l(p_{v1})$  and  $l(p_{v2})$  eventually correspond to the same  $p$  at different time.

The purpose of this paper is to propose a set of algorithms in a bottom-up manner to find the corresponding pairs of dynamic Web page instances obtained during different sessions, and then to judge which pages remain the same, and which pages have been updated or removed entirely since last time.

## 2 Problem Formulation

To put it more formally, the highest level algorithm in this paper is as follows. Given two pools of rewritten URLs  $L$  and  $L'$  collected recursively from the same starting point (base URL  $b$ ) during two different sessions, we try to find

1. every  $l_i$  and  $l'_j$  pair that corresponds to the same page source  $p_i$ , where  $l_i \in L$  and  $l'_j \in L'$ ;
2. the list of page links  $L'_{\text{upd}}$  whose page contents have been changed since last time;
3. the list of page links  $L'_{\text{sam}}$  whose page contents remain unchanged;
4. the list of page links  $L'_{\text{new}}$  whose pages never appear last time; and
5. the list of page links  $L'_{\text{err}}$  whose page contents cannot be accessed this time.

In order to achieve the first goal, we also need to find in advance the session-ids  $s$  embedded in  $L$  and  $s'$  in  $L'$ .

For brevity, the following symbols are used throughout the whole paper.

- $b$ : base URL.
- $l$ : URL link; converted to the absolute form for simplicity.
- $L$ : the set of links  $l_i$  where  $i \in \{1..n\}$ .
- $L$  and  $L'$ : two set of links obtained from the same  $b$  in distinct sessions, which can happen simultaneously or at different time.
- $p$ : Web page.
- $p_{v1}, p_{v2}$ : instances/versions of  $p$  obtained during two distinct sessions.
- $s$ : session-id.
- $s$  and  $s'$ : two instances of session-ids obtained from the same  $b$  in distinct sessions, which can happen simultaneously or at different time.

## 3 Finding the Session-Id in the URL Set

This section discusses how to find the session-id string in the URL set  $L$  collected recursively from the starting point  $b$  during one session. First we need to identify some important properties of session-ids that are useful in designing the algorithms. After that we introduce a series of subroutines to enumerate the session-id candidates, and finally design an algorithm to determine the exact session-id among these candidates.

### 3.1 Properties of Session-Ids

A dynamic Web page usually has not only static links that just point to some locations with no parameters but also dynamic links. Every dynamic link may consist of the location, parameters, and the session-id if URL rewriting is enabled. It follows that the average length of static links is usually shorter than that of dynamic links within the same website.

When a session-id is encoded and embedded within the URL, its valid character set is inevitably constrained by that of the URL. According to the URL specifications (Berners-Lee, Masinter & McCahill 1994, Berners-Lee, Fielding & Masinter 2005), some special characters (e.g., ';', '/', '?', '=', '&', '%', and '#') cannot be used directly within the *<scheme-specific-part>*; the same restriction applies to embedded session-ids as well. Consequently we can say that the next character following the session-id must be either a special character or an end-of-string (EOS) mark.

Although Web servers have much freedom to implement the session-ids, for security reasons most of them use some cryptographically secure one-way functions to construct the session-ids (Hallam-Baker 1996, Gutterman & Malkhi 2005). It follows that all session-ids on the same website normally have a fixed string length (e.g., 128 bits), that they seldom collide, and that their string contents are usually unlikely to appear elsewhere.

The URL rewriting mechanism also has another feature by nature. Since the session-id  $s$  remains identical during the same session, all dynamic pages' links in  $L$  should carry the  $s$  with themselves. It means that the  $s$  must be a substring of all dynamic links in  $L$ .

To sum up, session-ids have the following properties which are crucial to designing algorithms:

**Property 1.** *The session-id remains unchanged during the same session.*

**Property 2.** *Session-ids on the same website normally have a fixed length, even in distinct sessions.*

**Property 3.** *Session-ids on the same website seldom collide. It is also very unlikely to see more than one occurrence of the same session-id string literal within the body of the URL when URL rewriting mechanism is activated.*

**Property 4.** *When URL rewriting mechanism is activated, the session-id is embedded in the URL, and the next character following the session-id must be either a special character or an EOS mark.*

**Property 5.** *When URL rewriting mechanism is activated, the session-id must be one of the common substrings among all dynamically-generated URLs.*

**Property 6.** *When URL rewriting mechanism is activated, dynamic links are usually longer than the static ones within the same website.*

```

1. http://www.amazon.com/exec/obidos/subst/home/home.html/ref=three_tab_gw/002-9355727-0611208
2. http://www.amazon.com/exec/obidos/tg/browse/-/229220/ref=gw_subnav_gft/002-9355727-0611208?%5Fencoding=UTF8
3. http://www.amazon.com/exec/obidos/tg/stores/static/-/gateway/international-gateway/ref=gw_subnav_in/002-9355727-0611208?%5Fencoding=UTF8
4. http://www.amazon.com/exec/obidos/tg/new-for-you/new-releases/-/main/ref=gw_subnav_nr/002-9355727-0611208?%5Fencoding=UTF8

```

Figure 2: An excerpt of URL rewriting results taken from Amazon.com; full data is shown in Figure 6

### 3.2 Determining the Fixed Length of Session-Ids

The session-id must be one of the common substrings among all dynamically-generated URLs (see Property 5), but not all common substrings are the session-id. Take Figure 2 for example. At first glance many common substrings can be found, even if special characters (considering Property 4) are excluded:

- `exec`
- `obidos`
- `ref`
- `002-9355727-0611208`

But which one is the correct session-id? It seems that they all satisfy the properties listed in Section 3.1. It also seems difficult to generalize a set of syntactic rules to extract the correct session-id out of them.

Let's look at the session-id problem from yet another angle. The syntactic rules that synthesize the rewritten URLs may be complex and subject to changes, but the mechanism that generates the session-id itself is seldom subject to changes. Therefore, it seems easier to base our problem-solving strategy on the algorithmic nature of the session-id.

To find the right common substring as the session-id, first we need to determine the length of the valid session-id. Since the length on the same website normally remains the same (see Property 2), its value can be determined either by human inspection or by an easy-to-program yet effective heuristic. We can write a program to make two distinct but concurrent connections to the same website. The pages fetched in the two nearly-concurrent sessions would have almost identical layouts, topology, and contents; except for the links  $L$  and  $L'$  (or more precisely, session-ids  $s$  and  $s'$ ). A simple diff-like comparison (Myers 1986) between  $L$  and  $L'$  would reveal the fixed length of session-ids on this website.

The probe process needs to be done only once for each website unless the website changes its configuration, for example, server version, script engine, or session-key generation algorithm.

### 3.3 Finding the Common Substrings of the Given Length Between Two Strings

Before digging into the details of discovering the fixed-length common substrings among a set of links  $L$ , let's first discuss the basic subroutine of finding them between just two links.

The main idea of FIND-CS( $a, b, k$ ) algorithm in Figure 3 is discussed as follows. Since the session-id is a common substring in both strings  $a$  and  $b$ ,

```

Algorithm: FIND-CS( $a, b, k$ )
Input:   $a, b$ : two strings to be scanned
         $k$ : length of substring(s) to be found
Output:  $R$ : the set of all  $k$ -length common
        substrings between  $a$  and  $b$ 

Begin:
1   $R \leftarrow \{\}$ 
2   $C \leftarrow$  the set of URL special characters
3  ▷ for each substring of length  $k$  in  $b$ 
4  for  $i \leftarrow 1$  to  $\text{length}(b) - k + 1$  do
5       $t \leftarrow$  substring  $b[i..i + k - 1]$ 
6      if  $t$  contains no  $c \in C$ 
          and  $b[i + 1] \in C \cup \{\text{EOS}\}$  then
7          ▷ try to find  $t$  in  $a$ 
8          STRING-MATCHING( $a, t$ )
9          if found then
10              $R \leftarrow R \cup \{t\}$ 
11             end if
12         end if
13     end for
14     return  $R$ 

```

Figure 3: Algorithm for finding the common substrings of the given length between two strings

we can find all session-id candidates  $R$  by extracting every well-formed substring  $t_i$  from  $b$  and then trying to see if  $t_i$  is also in  $a$ . Every  $t_i$  has to, of course, be selected according to the constraint mentioned in Property 4. Its length  $k$  is obtained in the way stated previously in Section 3.2, and is specified as input to the FIND-CS algorithm.

Line 8 invokes some string-matching algorithm to see if  $t_i$  is also in  $a$ . Well-known algorithms such as the Knuth-Morris-Pratt (KMP) algorithm and the Boyer-Moore algorithm can be used here (Cormen, Leiserson, Rivest & Stein 2001).

### 3.4 Finding the Exact Session-Id

Just by knowing the exact length of session-ids (see Section 3.2) will not eliminate entirely the problem of finding the exact session-id strings. Take Figure 4 for example. There are two very long common substrings: the length of "1PS2V6KKYBZ34F3RK1PJ" is 20 and "002-9355727-0611208" is 19. Even if we are sure that the length of session-ids to be found is 19, and therefore "1PS2V6KKYBZ34F3RK1P" (notice that the trailing character 'J' is excluded) is ruled out by Property 4, FIND-CS( $a, b, 19$ ) still discovers more than one candidate:

- `002-9355727-0611208`
- `PS2V6KKYBZ34F3RK1PJ` (notice that the beginning character '1' is excluded)

Just by looking at the two links  $a$  and  $b$  is not enough; it gives us little information about which candidate is the session-id. All we need is to look at the whole set of links. The complete algorithm for finding session-ids in the whole set is listed in Figure 5. Let's explain the rationale behind the algorithm.

A dynamic Web page typically has both static and dynamic-generated links in it. Obviously session-ids can exist only in the dynamic ones. Therefore we need to separate the dynamic links from the static ones and focus on the former.

However, there is no general rules to distinguish both types of links literally. Two properties can help us solve this problem. Dynamic links tend to be longer than the static ones (according to Property 6),

```

a. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww
.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F
002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%
2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazo
n%2526cid%253Damazon%255Fgateway%255Ffpbox%26token%3D21513C
47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd
_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER
&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ

b. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww
.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_6494
02_2%2F002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.am
azon.com%2Fgp%2Fsearch.html%2F%253Fplatform%253Dgurupa%2526
url%253Dnode%253D11055981%2526keywords%253DT3%26token%3DD2E
7DF9A889DD105A02CC33159540A6D52DE4D8B&pf_rd_p=160346101&pf
_rd_s=right-4&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0D
ER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ

```

Figure 4: Another excerpt of URL rewriting results taken from Amazon.com; full data is shown in Figure 6

**Algorithm:** FIND-SID( $L, k$ )  
Input:  $L$ : pool of links  
 $k$ : length of session-id string to be found  
Output: session-id string of length  $k$   
Begin:

```

1  ▷ find session-id candidates  $V$ 
2   $M \leftarrow$  clone of  $L$ 
3  do
4       $t \leftarrow$  longest string in  $M$ 
5       $u \leftarrow$  second longest string in  $M$ 
6      remove  $t$  and  $u$  from  $M$ 
7       $V \leftarrow$  FIND-CS( $t, u, k$ )
8  while  $V = \{\}$ 
9  ▷ calculate the number of occurrences
   for each  $v \in V$  in  $M$ 
10  $D$ : associative array mapping from string
    to integer
11 for each  $v \in V$  do
12    $D[v] \leftarrow 0$ 
13   for each  $m \in M$  do
14     if  $v$  is a substring of  $m$  then
15        $D[v] \leftarrow D[v] + 1$ 
16     end if
17   end for
18 end for
19 ▷ identify the right session-id among
   candidates  $V$ 
20  $s \leftarrow \arg \max_v (D[v])$ 
21 return  $s$ 

```

Figure 5: Algorithm for finding the exact session-id

and therefore Lines 4–5 pick the longest links  $t$  and  $u$  as a guess. Moreover, the session-id is one of the common substrings of these dynamic links (according to Property 5), and therefore Line 7 calls the FIND-CS algorithm to enumerate the candidates  $V$ . If no candidate is found, implying that our previous guess was wrong, the surrounding loop in Lines 3–8 is responsible for making a second guess (or more, if necessary).

At present  $V$  may also contain some non-session-id strings by chance, as Figure 4 has showed. A property can help us distinguish the real session-id from the others. According to Property 5, the session-id string should appear in *all* dynamic links, implying that the most frequent candidate is more likely to be the real session-id. Therefore Lines 10–18 try to find the frequency of each candidate in the whole links (albeit not in the *dynamic* links), and Line 20 picks the most frequent one as the session-id.

### 3.5 A Complete Example

Let's use a complete example to demonstrate the whole scenario of finding the session-id from the start. The example was taken from Amazon.com in the middle of March, 2006. The base URL  $b$  was <http://www.amazon.com> and the traversal depth was 0. Seventeen links were obtained and shown in Figure 6.

Let  $L$  denote the whole set of 17 links  $l_1..l_{17}$ . By the probe process mentioned in Section 3.2, the length of session-id is found to be 19 on this website. Lines 4–5 in FIND-SID( $L, 19$ ) would first pick the longest two links:  $l_9$  and  $l_{10}$ .  $l_9$  and  $l_{10}$  equal to  $a$  and  $b$  in Figure 2, and therefore FIND-CS( $l_9, l_{10}, 19$ ) in Line 7 outputs the same results mentioned in the first paragraph of Section 3.4: 002-9355727-0611208 and PS2V6KKYBZ34F3RK1PJ. Afterwards Lines 10–18 calculate the frequency of them as 14 and 5, respectively. Consequently the most frequent one, 002-9355727-0611208, is decided as the session-id.

### 3.6 Discussion

With these algorithms, we can easily find the session-id among a collection of URLs. But the mechanism bases on some assumptions and therefore has some limitations.

Many aspects of these algorithms rely on the probability that Property 6 holds. If dynamic links are shorter than the static ones on some websites, Lines 4–5 of the FIND-SID algorithm would be in the wrong direction. In addition, if the number of dynamic links is too small, Line 13 would cover too

```

1. http://www.amazon.com/exec/obidos/subst/home/home.html/ref=three_tab_gw/002-9355727-0611208
2. http://www.amazon.com/exec/obidos/tg/browse/-/229220/ref=gw_subnav_gft/002-9355727-0611208?%5Fencoding=UTF8
3. http://www.amazon.com/exec/obidos/tg/stores/static/-/gateway/international-gateway/ref=gw_subnav_in/002-9355727-0611208?%5Fencoding=UTF8
4. http://www.amazon.com/exec/obidos/tg/new-for-you/new-releases/-/main/ref=gw_subnav_nr/002-9355727-0611208?%5Fencoding=UTF8
5. http://www.amazon.com/exec/obidos/tg/new-for-you/top-sellers/-/main/ref=gw_subnav_ts/002-9355727-0611208?%5Fencoding=UTF8
6. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_el_feat%2F002-9355727-0611208%3F%255Fencoding%3DUTF8%26node%3D172282&pf_rd_p=163187901&pf_rd_s=leftnav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
7. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_gf%2F002-9355727-0611208%3F%255Fencoding%3DUTF8%26node%3D3370831&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
8. http://www.amazon.com/gp/amabot/?pf_rd_url=%2Fgp%2Fbrowse.html%2Fref%3Dgw_br_jw%2F002-9355727-0611208%3F%255Fencoding%3DUTF8%26node%3D3367581&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
9. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Dgf_gw_wine%2F002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.wine.com%2Fpromos%2Famazonwine.asp%253Fan%253Damazon%2526s%253Damazon%2526cid%253Damazon%255Fgateway%255Ffbbox%26token%3D21513C47B07C95040C8597937CAB64718E97425C&pf_rd_p=163187901&pf_rd_s=left-nav&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
10. http://www.amazon.com/gp/amabot/?pf_rd_url=http%3A%2F%2Fwww.amazon.com%3A80%2Fgp%2Fredirect.html%2Fref%3Damb_link_649402_2%2F002-9355727-0611208%3Flocation%3Dhttp%3A%2F%2Fwww.amazon.com%2Fgp%2Fsearch.html%2F%253Fplatform%253Dgurupa%2526url%253Dnode%253D11055981%2526keywords%253D%253D%26token%3DD2E7DF9A889DD105A02CC33159540A6D52DE4D8B&pf_rd_p=160346101&pf_rd_s=right-4&pf_rd_t=101&pf_rd_i=507846&pf_rd_m=ATVPDKIKX0DER&pf_rd_r=1PS2V6KKYBZ34F3RK1PJ
11. http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.uk%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-ukhome-21%26site%3Damazon
12. http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.de%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-dehome-21%26site%3Dhome
13. http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.co.jp%2Fexec%2Fobidos%2Fredirect-home%3Ftag%3Dintl-usstoreh-jphome-22%26site%3Damazon
14. http://www.amazon.com/exec/obidos/redirect-to-external-url/002-9355727-0611208?%5Fencoding=UTF8&path=http%3A%2F%2Fwww.amazon.fr%2Fexec%2Fobidos%2Fredirect-home%3Fsite%3Damazon%26tag%3Dusfr-storeh-footer-21
15. http://ec1.images-amazon.com/images/G/01/marketing/visa/amzn_visa-save-30-today.gif
16. http://ec1.images-amazon.com/images/G/01/gateway/partner-logos/target_logo_V58636053.gif
17. http://images.amazon.com/images/P/6305428050.01._PE30_SCTZZZZZZZ.jpg

```

Figure 6: A full example of URL rewriting results taken from Amazon.com with the traversal depth of 0

many irrelevant links and consequently contribute too many counts to the wrong candidates.

A possible workaround to these limitations is to increase the breadth and depth of traversal in the beginning. In this way, every single wrong candidate may have less chance to have higher frequency than the real session-id does. More evaluations are to be done in Section 5.1.

## 4 Tracking the States of Web Pages

### 4.1 Algorithm

Based upon previous subroutines, we are ready to track the change states of Web pages in the existence of URL rewriting. The tracking algorithm is shown in Figure 7. Inside it, two subroutines (CRAWL and GET-NEWEST-DOCUMENT) are left as implementation details.

In the beginning the crawler's repository has a record containing a base URL  $b$  and a set of pages (referred to by  $L$ ) traversed and collected from  $b$  last time. The traversal breadth and depth are determined according to some pre-given values or inclusion/exclusion rules.

Now we would like to know if these pages have any change since last time. At first Line 4 crawls the documents starting from  $b$  in the same way that  $L$  was obtained last time.

To compare the change states between pages referred to by  $L$  and  $L'$ , their session-ids have to be unified to the same ground in advance. It is obvious that if pages from  $L$  still exist in the meantime, their links would remain identical except for the session-id part. Therefore Lines 8–10 replace the old  $s$  in  $L$  with the new  $s'$ . The replacement is safe because session-id strings are meant to be unique; they seldom appear elsewhere in the URL string (see Property 3). Normally we do not need to worry about the risk of corrupting the non-session-id parts.

When the unification is done, the relation between two sets  $L$  and  $L'$  can be illustrated in Figure 8. The case for  $L'_{\text{new}}$  is so trivial that Line 11 sets it as  $L' - L$  accordingly.

The case for  $L - L'$  is a little tricky. It may due to access error (Lines 15–17); it may also due to the orphan phenomenon. A page is called an orphan if it existed before and still exists at present, but for some reasons the links to it disappear so that it is not recursively reachable via  $b$  anymore. The contents of

**Algorithm:** TRACK( $b, L, k$ )

Input:  $b$ : base URL  
 $L$ : pool of links collected last time  
 $k$ : length of session-id string

Output:  $L'_{\text{sam}}$ : current links; content unchanged since last time  
 $L'_{\text{upd}}$ : current links; content updated since last time  
 $L'_{\text{new}}$ : current links; never appeared last time  
 $L'_{\text{err}}$ : current links; fail to access this time

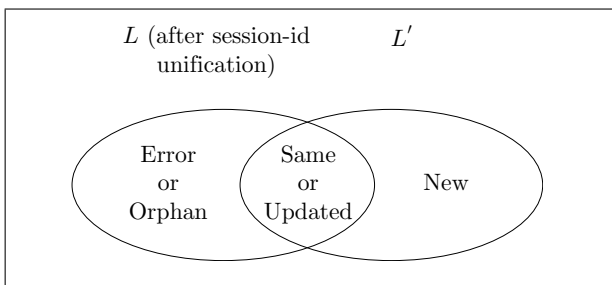
Begin:

```

1   $L'_{\text{sam}} \leftarrow \{\}$ 
2   $L'_{\text{upd}} \leftarrow \{\}$ 
3   $L'_{\text{err}} \leftarrow \{\}$ 
4   $L' \leftarrow \text{CRAWL}(b)$ 
5   $\triangleright$  update the old session-id with the new one
6   $s' \leftarrow \text{FIND-SID}(L', k)$ 
7   $s \leftarrow \text{FIND-SID}(L, k)$ 
8  for each  $l \in L$  do
9      replace the substring  $s$  in  $l$  with  $s'$ , if any
10 end for
11  $L'_{\text{new}} \leftarrow L' - L$ 
12  $\triangleright$  check: orphan or error?
13  $O \leftarrow \{\}$   $\triangleright$  orphan
14 for each  $l \in L - L'$  do
15      $p_{v2} \leftarrow \text{GET-NEWEST-DOCUMENT}(l)$ 
16     if  $p_{v2}$  is null then
17          $L'_{\text{err}} \leftarrow L'_{\text{err}} \cup \{l\}$ 
18     else
19          $O \leftarrow O \cup \{l\}$ 
20     end if
21 end for
22  $\triangleright$  check: same or updated?
23 for each  $l' \in (L \cap L') \cup O$  do
24      $p_{v1} \leftarrow$  previous version of  $l'$  page content in the repository
25      $p_{v2} \leftarrow$  current version of  $l'$  page content in the repository
26     if  $p_{v1} = p_{v2}$  then
27          $L'_{\text{sam}} \leftarrow L'_{\text{sam}} \cup \{l'\}$ 
28     else
29          $L'_{\text{upd}} \leftarrow L'_{\text{upd}} \cup \{l'\}$ 
30         update  $p_{v1}$  with  $p_{v2}$  in the repository
31     end if
32 end for
33 return  $L'_{\text{sam}}, L'_{\text{upd}}, L'_{\text{new}}, L'_{\text{err}}$ 

```

Figure 7: Algorithm for tracking the states of Web pages

Figure 8: Relation between two sets of links  $L$  and  $L'$  (after session-id unification)

such orphans may, of course, remain the same or have been updated since last time. Consequently Lines 13–21 try to identify them for Lines 23–32 to process.

The main loop in Lines 23–32 is obvious. It tries to validate every link in  $(L \cap L')$  plus orphans to decide whether the page has been updated or not.

#### 4.2 Implementation Hints

There are some other things that are important for implementing the TRACK algorithm.

If there are a large proportion of static pages in the collection, the CRAWL subroutine in Line 4 can be optimized further. When it tries to fetch a document from a remote website, the `If-Modified-Since` request header (Fielding et al. 1999) can be used to avoid retransmission of the unchanged content.

Web servers and server-side script engines have a timeout value for sessions. For example, by default the timeout is 60 minutes in Tomcat (an open-source Java servlets/JSP container), 20 minutes in ASP.NET, and 24 minutes in PHP. But the timeout value is reconfigurable, especially on e-commerce websites. Therefore implementation of TRACK should consider this issue and should not let the session idle

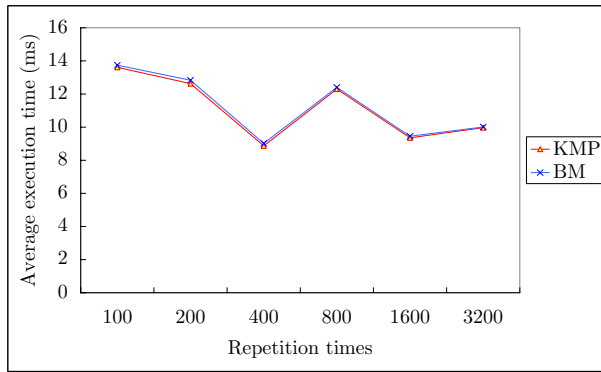


Figure 9: Performance of FIND-SID algorithm on Amazon.com. The experiment was done on Pentium 4 2.8 GHz running Mandriva Linux 2006 and JDK 1.4.2\_11-b06. In the figure, KMP stands for the Knuth-Morris-Pratt algorithm, and BM stands for the Boyer-Moore algorithm.

too long between Line 4 and the first invocation of Line 15.

Roughly speaking, the idle time would be proportional to the execution time for FIND-SID since the time complexity for Line 6 is identical to that for Line 7, and the time complexity for Lines 8–10 is a little lower. It seems a good place to use a real-world example to evaluate the performance of FIND-SID. The experiment was conducted by repeating the one mentioned in Section 3.5 several times. Both Knuth-Morris-Pratt and Boyer-Moore algorithms were employed for comparison. The time spent in network transmission was excluded from the result.

As can be seen in Figure 9, even for such a website with very long dynamic links and session-ids, the FIND-SID algorithm took about 10 milliseconds—much shorter than typical session idle time, we think.

If performance pursuers still worry about the session idle time, it can be reduced to about 2/3 by moving Line 7 to any location before Line 4. Line 11 can also be moved to any location after Lines 13–21 or Lines 23–32 since it has no side effect on the latter two blocks. Finally, the loop in Lines 14–21 can be executed on another concurrent thread to be triggered by Line 9.

## 5 Experimental Results

To validate the effectiveness of these algorithms, we conducted a set of experiments in a bottom-up manner.

### 5.1 Correctness of FIND-SID Algorithm

To test the correctness of FIND-SID algorithm, we first try to investigate the number of candidates found by FIND-CS algorithm, and also the recall of the finding. Finally we investigate the correctness of FIND-SID algorithm.

The experiments were conducted on 4 websites, with the traversal depth of 1. Each was repeated 100 times.

- A: <http://www.amazon.com/> and the length of session-id = 19.
- B: <http://www.amazon.co.jp/> and the length of session-id = 19.
- C: <http://webcaspar.nsf.gov/> and the length of session-id = 32.

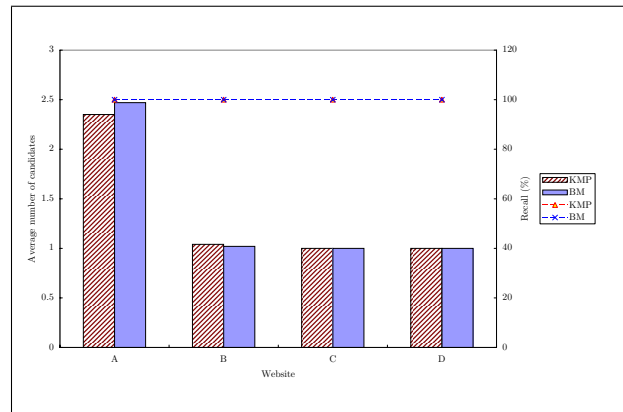


Figure 10: Correctness of FIND-CS algorithm on websites A, B, C, and D, as mentioned in Section 5.1. The bar chart and the  $y$ -axis on the left-hand side show the average number of candidates found, while the line graph and the  $y$ -axis on the right-hand side show the average recall of session-ids.

- D: <http://www.jesishpittsburgh.org/> and the length of session-id = 32.

First, Figure 10 shows that the average numbers of candidates were very small, and the recalls were 100% all the time. The outcome was very accurate.

The next step is to evaluate the correctness of FIND-SID. The outcome was all 100% correct, so the figure is omitted.

### 5.2 Correctness of TRACK Algorithm

The experiments were conducted on 2 portfolios of Web pages. The tracking step was done within a half hour after the first crawling step. It was to avoid too few intersections and too many missing pages since these websites changed their contents very frequently.

- A: <http://www.amazon.com/> with depth = 1.
- B: composed of 3 subtasks:
  - B<sub>1</sub>: <http://tw.news.yahoo.com/finance/> with depth = 2,
  - B<sub>2</sub>: <http://tw.news.yahoo.com/technology/> with depth = 2,
  - B<sub>3</sub>: search <http://tw.news.yahoo.com/> with keyword “google”.

Bitwise comparison between  $L$  and  $L'$  pairs showed that the outcome in Table 1 was 100% accurate. Close inspection of the pairs revealed further that varying advertisements and embedded JavaScript codes contributed to a smaller  $L'_{sam}$ . If we want a more tolerant  $L'_{sam}$ , i.e., not necessarily bitwise identical, the “equality” test in Line 26 of the TRACK algorithm should be relaxed to allow for content-based filtering and customization.

## 6 Conclusion and Future Work

The URL rewriting mechanism is a popular way to enable sessions between Web clients and servers. However, the mechanism and the lack of regularity in the URLs produced have raised problems for knowledge management systems to track the changes of Web pages. This paper has devised a series of algorithms in a bottom-up manner, and also demonstrated that they are very effective and efficient in tracking dynamic Web pages in the existence of URL rewriting.

Table 1: Experimental results of the TRACK algorithm

	Portfolio A	Portfolio B
First crawling $L$ :	2006-06-06 20:31 340	2006-06-07 14:26 139
Tracking	2006-06-06 20:55	2006-06-07 14:40
$L'_{sam}$ :	76	19
$L'_{upd}$ :	264	120
$L'_{new}$ :	54	3
$L'_{err}$ :	0	0

This paper has focused mainly on the issue raised by the URL rewriting mechanism. In the future, we will move on to other practical issues when implementing the crawler module of a KMS. For example, it is important to investigate more proper ways to define and handle the “equality” of dynamic pages from the user’s point of view. To do this, first we plan to incorporate the idea of AJAX-based screen-scraping toolkit so that users can specify the regions of interest in a more intuitive way. Second, we plan to provide content-based filtering so that users can customize their own equality test.

Another important issue is to investigate a sophisticated but also intuitive way for users to specify and refine a series of steps required to navigate into a specific subset of the target website. To do this, the same technique of AJAX-based screen scraping can also be helpful. We think that this approach is more finer-grained and easier-to-use than traditional command-line argument or HTML screen-scraping approaches.

## 7 Acknowledgments

This work was mainly supported by National Science Council grant NSC95-2752-E-009-PAE: advanced technologies and applications for next generation information networks, and partially supported under grant NSC94-2213-E-009-026 and NSC94-2520-S-009-004.

## References

- Berners-Lee, T., Fielding, R. & Masinter, L. (2005), Uniform Resource Identifier (URI): Generic syntax, RFC 3986, Internet Engineering Task Force.
- Berners-Lee, T., Masinter, L. & McCahill, M. (1994), Uniform Resource Locators (URL), RFC 1738, Internet Engineering Task Force.
- Cormen, T. H., Leiserson, C. E., Rivest, R. L. & Stein, C. (2001), *Introduction to Algorithms*, second edn, The MIT Press.
- Coward, D. & Yoshida, Y. (2004), Java servlet specification version 2.4, JSR 154, Sun Microsystems.
- Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. & Berners-Lee, T. (1999), Hypertext Transfer Protocol – HTTP/1.1, RFC 2616, Internet Engineering Task Force.
- Guterman, Z. & Malkhi, D. (2005), ‘Hold your sessions: An attack on Java session-id generation’, *Lecture Notes in Computer Science* **3376**, 44–57.
- Hallam-Baker, P. M. (1996), Session identification URI, W3C working draft, World Wide Web Consortium.
- Kristol, D. M. (2001), ‘HTTP cookies: Standards, privacy, and politics’, *ACM Transactions on Internet Technology* **1**(2), 151–198.
- Kristol, D. & Montulli, L. (2000), HTTP state management mechanism, RFC 2695, Internet Engineering Task Force.
- Myers, E. W. (1986), ‘An  $O(ND)$  difference algorithm and its variations’, *Algorithmica* **1**(1), 251–266.
- Sit, E. & Fu, K. (2001), ‘Inside risks: Web cookies: not just a privacy risk’, *Communications of the ACM* **44**(9), 120.