netpreserve.org
international internet preservation consortium

Test Bed Taxonomy for Crawler

Andrew Boyko

Version 1.0

Martha Anderson
Gina Jones

The Library of Congress
101 Independence Ave, SE
Washington, DC 20540
USA

Andrew Boyko

Test Bed Taxonomy for Crawler

Background

In defining a test bed for web archiving, we first constrain the scope of the discussion to archiving by means of automated crawling, rather than considering the more general problem of harvest by means such as information deposited by the content owners. We wish to characterize challenges that a crawler may encounter on the Web at large, with the goal of creating permanent demonstrations of each condition. The challenges a crawler faces may be categorized in any number of varying ways, but we choose here to separate them into what we believe are three mutually exclusive categories, which are distinguished by the phase of the crawling process in which they occur.

Content Retrieval Issues

The first category of challenge is in the acquisition of a given piece of content – formulating the request, retrieving it completely and without modification, ensuring that the content retrieved represents the same content a site's user would normally expect to see, and retrieving and storing all relevant metadata describing the context for the retrieved content. The problems that block success in this category stem from variety – in retrieval protocols, in browser environments, in language and encoding, and in implementation of standards. Our test bed must identify or synthesize content reflecting this variety.

Link Detection Issues

A successful crawl must of course not only retrieve content, but also detect and follow links within that content, and so the second category of challenge is in interpreting the meaning of an already-retrieved piece of content, in order to crawl outward from it. Again, variety is the primary obstacle to complete success – consider the many possibilities of content type, each potentially requiring customized understanding. In particular, note the ubiquitous nature of client-side JavaScript, whose flexibility results in myriad ways to concoct features like dynamic menus, each requiring special-case handling in a crawler in order to detect links. In addition to finding or creating content with challenging links, we must also synthesize sites with large and difficult structure, to ensure that the tool completely crawls the entire site.[1]

Presentation Issues

The scope of the test bed is limited to issues in the actual harvesting of content. The current presentation tools (Wayback Machine, NWA) inject, unavoidably, issues of their own, and since the tools are necessary to inspect the archived content, identifying the separating harvesting from presentation is often challenging. Nevertheless, as much as possible we endeavor to focus solely on validating that the content makes it to local storage intact, with all possible accompanying metadata, and defer issues of presentation to another discussion.

Test Bed Topics

With the goal of constructing a live test bed for a web crawling system, we enumerate here a taxonomy of challenges, each of which should straightforwardly indicate one or more test cases to validate a crawler's correctness with regard to the challenge. These test cases will be described at a level of detail that permits implementation in a test bed under our control, comprising pages, scripts, and programs. Entries in this list are not necessarily either unsolved problems, nor are they guaranteed to be completely understood; rather, this collection aims to completely collect all cases we might be able to specify. Existing Internet sites demonstrating each case should also be collected, but by creating our own content for a definitive test bed, we avoid the event of an identified site changing or disappearing.

It must be noted that this categorization intentionally does not address the distinction between "deep" and "surface" web, for these attributes are by-products of the use of the test bed, rather than defining it. If there is a reasonable definition of "deep web" in this context, it is "that which cannot be successfully crawled". The purpose of defining test cases to illuminate where the tools fail, and to identify, for a given crawling tool at a point in time, where the boundary between "deep" and "surface" lies. Our definition here ignores the general problem of deep *information*, such as a relational database, for which it is not meaningful to describe links, and which thus cannot be crawled. While deep information is of significant interest to this community, it requires entirely different processes and tools and is out of the scope of this test bed.

It is regrettably easier to find a problem area for crawling than to implement crawler code to correctly handle it, and with this in mind, pragmatism requires that we define certain test cases without expecting their successful completion using the current generation of tools. Portions of the test bed serve as suggestions for future tool development.

A test case will take the form:

- A technical description of the case

[1] Note that we cannot rely on external sites to test coverage, since there is no objective measurement of a remote site's size and scope available without crawling it and no tool can be assumed correct; thus, we must maintain our own coverage site.

- Specific steps required to implement a test case (harvestable page or site) that demonstrates the case

- A description of successful completion of the given test case

- An assessment of the ability of current crawling tools to successfully complete the test case

- Extant Internet sites demonstrating the case, if known

- An assessment of the case's relative prevalence on the Internet, and typical habitats

- An assessment of the importance of this test case

What follows is an annotated taxonomy. It should be reasonable to create test cases for each outermost (leaf) level of the tree. No particular information is conveyed by the ordering of theentries contained herein; it is not a linear ranking of difficulty. Any tool will certainly succeed and fail at a different subset of the cases in the test bed.

Taxonomy Outline

1 Test Bed: Retrieval Issues

These issues involve the process of formulating and performing a request for content in a web crawl, and receiving, validating, and storing the response. Issues relating to the parsing of the content, particularly in the identification of links contained within the content, are addressed in section 2; for the purposes of this section, the actual response can be considered an opaque sequence of bytes.

1.1 Difficult URIs

We assume that the crawler is driven by URIs, supplied by a list of initial seeds and extended by extraction from retrieved documents.

1.1.1 URIs with invalid characters

The URI specification (RFC 2396) and URL specification (RFC 1738) are clear on permissible characters and quotings. Nevertheless, plenty of real content includes technically illegal URIs, for which a reasonable interpretation can be made. In addition, a crawler should correctly log URIs that cannot be handled by the tool, for later collection and identification.

Invalid URI Schemes

Invalid HTTP URIs

1.1.2 URIs with common prefix, varying form parameters (following '?')

A common pattern in URIs found on sites with dynamic content is URIs with a matching base prefix (preceding the '?' delimiter) and varying suffixes, most commonly as form parameters. These URIs are logically distinct and should be handled as separate items. Though the URI suffix (following the '?') may be interpreted as form parameters, the crawler should not attempt to dissect the parameters and make its own interpretation (e.g. treating http://site/?a=b&c=d as identical to http://site/?c=d&a=b)

1.1.3 Ensure no duplication with varying but synonymous character quoting

The URI specification describes how a character may be represented as "escaped" by prefixing the character's 2-digit hex representation with '%'. URIs should be converted to a single canonical form before comparison.

1.1.4 No duplication with varying forms of hostname or IP address

In general, host names contained in URIs cannot be compared or normalized, except in certain degenerate cases.

URIs containing hostnames

Host names are case-insensitive, and should be compared as such. A DNS resolver will ignore a trailing period, which may also confound URI comparison. Hostnames containing technically invalid characters (per RFC 1034), such as “_”, may exist and should be tolerated and sent to the DNS resolver. Hostnames should not generally be considered identical if they differ in components, regardless of typical approach (e.g. www.host.tld is not guaranteed to be the same host as host.tld).

URIs with IP addresses

IP addresses may appear in URIs (though they are discouraged). A given IP may be printed in the typical dotted-quad form (N.N.N.N), but may equivalently be supplied as a single 32-bit number or other degenerate but legal form.

1.2 HTTP Typical Cases

1.2.1 Static content retrieval

Basic cases of static files of commonly found MIME content types will be served.

1.2.2 Script-based content creation

A variety of scripts (PHP, JSP, Perl, as necessary) will create content of varying types (HTML, XML, images, PDF), including variations in the attributes of the transmission (buffering, absence/presence of Content-length header vs. chunking, …)

1.2.3 Redirection

Test cases should include temporary and permanent redirections, of varying nestings (including potentially infinite recursion), both local to the current host and external, with varying levels of spec compliance/correctness.

1.3 HTTP Edge Cases

The HTTP protocol is complex, and many atypical anomalies might confound a crawler.

1.3.1 Extreme size

A crawler will support a maximum retrievable size, which may be unlimited. A test case should validate that this size is honored, and if unlimited, that arbitrarily large content can be retrieved (including obvious boundaries such as >2GiB, >4GiB, larger…)

1.3.2 Communication slowdowns/timeouts

A server might have an unreliable connection or route, or be otherwise overloaded, in which cases it may send content slowly, with pauses or at a trickled rate. A crawler’s configured timeouts should be validated, but otherwise a slow trickle of data should be completely collected.

1.3.3 TCP connection anomalies

Any permutation of unexpected bizarre TCP-level responses should be tolerated, and should not cause permanent failure of the process, nor should a long sequence of such responses (from a malicious or malfunctioning server) cause resource losses.

1.3.4 Invalid server responses

A malicious or broken HTTP server sending back invalid or mangled HTTP responses should not cause any crawler difficulties.

1.3.5 HTTP/1.0 server intolerant of 1.1 requests

The crawler should be entirely tolerant of naïve/clumsy/homegrown HTTP servers, particularly those offering a minimal HTTP/1.0-compliant implementation.

1.4 Cookies

1.4.1 Scripts requiring previously-set cookies before sending content

In the case where a dynamic site sets a cookie and then presents certain subsequent content only when presented with that cookie in future requests, the crawler should receive such content.

1.4.2 Site providing different content for same URIs with different cookies set

A given page may present different content depending on the value, presence, or absence of a cookie, which may be differently set based on crawl path through the site. (Example: a choice of language, which sets a cookie that affects the presentation of future pages)

1.5 Implicit Content

There exists a category of content made available on web servers but not linked to from the site itself; rather, the content is understood to be available at well-known URIs. In all known cases, such content is essentially metadata for the entire site, and should probably be retrieved along with other content from the site.

1.5.1 Robots Exclusion Rules (robots.txt)

The most well known implicit content, a resource named http://hostname/robots.txt is widely understood to provide rules for automated crawlers regarding content that should be ignored. It should be retrieved and stored along with site content; whether its contents are honored by the crawler is addressed in section 0.

1.5.2 Favicon.ico

A de facto standard exists for a Windows-format icon file to be placed at http://hostname/favicon.ico, which many browsers (on multiple platforms) now use as a visual identifier for the site. As such, it should be retrieved as part of the context for a site.

1.5.3 P3P

The Platform for Privacy Preferences specification (http://www.w3.org/TR/P3P/) provides an XML format for site privacy statements, which are available either at a well-known location or explicitly named by a LINK tag. In either case, any available P3P content should be retrieved as part of the site’s context.

1.6 Negotiated Content

A site may provide entirely different content for a given URI depending on information provided by the client in HTTP headers; often, these headers will be set in every request, and announce a browser's capabilities or local context.

1.6.1 Charset

The Accept-Charset header allows a client to suggest a more preferable character encoding than the default ISO-8859-1.

1.6.2 Content-Type

The Accept header allows a client to suggest its preference of MIME content type for a given resource.

1.6.3 Encoding

The Accept-Encoding header allows a client to suggest its preference of content coding, such as various forms of compression (none, UNIX compress, or gzip, most commonly).

1.6.4 Language

The Accept-Language header allows a client to suggest its preference of natural language, should a server have multiple versions of the resource available.

1.6.5 User-Agent

The User-Agent header, which identifies the software implementation of the client as well as its platform, is sometimes used by server code to select content for a URI based on the expected capabilities of the software. While it is an error-prone practice, and the header is easily falsified, its frequency means it should be taken into account.

1.7 HTTP Authentication

Various implementations exist for protecting resources from anonymous access by requiring a username and password. Assuming the authorization is available to the crawler client, these conditions can be implemented straightforwardly.

1.7.1 Header-based (Basic, Digest)

The HTTP protocol supplies a header-based authentication scheme, in which a request for a protected resource returns HTTP status 401 (Unauthorized), along with a WWW-Authenticate header describing the protection scheme. A client would resubmit the request with an Authorization header with the appropriate username and password, encoded in a form according to the scheme. Completing this test case requires configuration by URL and/or the "realm" named in the WWW-Authenticate header's contents.

1.7.2 Form-based

More common than header-based authentication is the embedding of a username and password form in HTML content. While this approach has typical patterns, the particular naming of the HTML form fields for the username and password, and the URL to submit the form to, will vary by site. Handling this case correctly requires tolerance of this variance, and thus separate configuration, by URL.

1.8 Non-HTTP protocols

While the majority of the content under our interest is available by HTTP, several other protocols warrant our validation.

1.8.1 HTTPS

The secure variant of HTTP communicates on a different TCP port than the default port 80. Its use requires clients to maintain a valid set of root security certificates.

Bad Certificates

A crawler should tolerate but note the presence of expired certificates.

OCSP

A crawler may wish to verify the validity of a supplied certificate using the OCSP protocol.

1.8.2 DNS

A crawler can be expected to archive the DNS information retrieved as part of the harvest.

1.8.3 FTP

FTP is still a common delivery protocol for file distribution, linked from HTML content.

1.8.4 Streaming Media, non-HTTP client (e.g. Real w/RTSP)

Streaming media (audio or video) is typically not served by the web server process, but by a separate server program, not necessarily on the same machine as the web server providing the content with the initial link. Commonly the streaming server is not linked to by a direct URL in the source page; rather, the link retrieves via HTTP a small intermediate file with a MIME content type that launches the media player client. The media player interprets the contents of the file and connects to the appropriate streaming server referenced in the URL therein.

1.8.5 Streaming Media, custom HTTP client (e.g. Windows Media Player using HTTP)

An alternative delivery protocol for streaming media, used in some cases by Windows Media content, is HTTP, but using a custom client (Windows Media Player) and with non-standard HTTP headers that identify the client as WMP. Because the protocol remains HTTP, it may be possible to crawl this content using a tool that can emulate the WMP behavior.

1.8.6 Custom-protocol data feed

Rich content types such as Flash and Java applets may open a direct connection to a server, using a custom protocol, in order to retrieve a live feed of data (for example, a database query, or some other real-time updates). This conversation cannot be straightforwardly detected without executing the content in its host environment, and even if detected, the dynamic nature of the content renders it difficult or impossible to retrieve, similar to the general problem of database harvest.

1.9 HTML Form crawling

The submission of an HTML form is a potentially complex request, and though it can be sidestepped in some cases by extracting URLs present in the FORM tag's target attribute and possibly also in form fields, it remains that some resources are only accessible by performing a realistic form submission.

Certain forms will never be generally harvestable by a crawler, both those using the simpler GET method or the more complex POST method. The presence of a text entry field in a form, should it be logically required for the form to be handled, renders the form uncrawlable, as the crawler cannot guess what text should fill the form field. Even in the case of forms consisting of input fields with constrained values, such as checkboxes, radio buttons, and selection pop-ups, the combinatorial variety of possible inputs makes any form with more than a few fields impractical to crawl.

1.9.1 GET requests

Forms implemented with the GET method are expected to be idempotent; that is, the result of the form's submission will be the same each time the request is sent, with the implication that the state of the web server is not changed by the request (e.g. database query). With the GET method, the form's fields' contents are stored entirely in the URL, so a crawled GET form can be archived by its URL as can any other synthetic URL.

1.9.2 POST requests

The expectation of idempotency does not apply to POST requests; a common POST example would be a form that updates a database, in which example each submission of the form by a crawler would presumably create a new, different response. This suggests that harvesting POST forms could be generally inappropriate, as the harvest should be documentary in nature rather than affecting the resource being crawled.

The unique identification of a POST request comprises not only the URL but also the body of the request, which is potentially quite large; this will require different strategies for presentation.

1.10 Robot exclusion compliance

It should be possible to constrain the crawler's scope to only the content considered permissible by conventions for expressing robot exclusions. It should also be possible to limit the crawler's request rate to constrain its use of network and computing resources both locally and remotely.

1.10.1 Compliance with robots.txt

We wish to test whether the crawler correctly avoids the URL space defined in a site's robots.txt exclusion when the tool is configured to do so, and whether it ignores the exclusions when so configured. Test cases should include tolerance of some typical malformations of the robots.txt format.

1.10.2 Compliance with robot META tags

META tags in HTML documents may also express archiving exclusions in a form similar to the robots.txt file, and should be honored if configured.

1.10.3 Ensure robot headers are sent (User-Agent, From)

Related to robots.txt compliance is the definition of standard HTTP headers that should be supplied by a crawler. A crawler configured to hide its robot nature and masquerade as a browser may omit these.

1.10.4 Ensure reasonable request rate and bandwidth limits

We may evaluate a crawler's compliance with configured politeness in several dimensions, including peak and sustained bandwidth usage, number of concurrent threads visiting a single host, rate of requests, and length pausing between requests.

1.11 Change Detection

It may be desirable for a crawler to be able to examine and compare content it has already acquired as it considers its crawl frontier.

1.11.1 Duplicate detection: www.x.com often identical to x.com

A crawl whose scope is an entire domain (or broader) has some chance of repeatedly crawling a site that is available under multiple host names. Perhaps the most common example of this condition is when a site's content is available from http://hostname.tld/ as well as from http://www.hostname.tld/, but in general any two host names may conceivably mirror the same content. Note that no DNS lookups can either identify or preclude this condition; it requires analysis of the acquired content.

1.11.2 Change over time (if-modified-since headers supported)

A long-term crawl process may revisit content already acquired in order to detect whether an updated copy of the content may be added to the acquisition. HTTP provides a standard mechanism for requesting content if it has been updated since a given date. Test cases should include trivial/automated change (e.g. JavaScript-based current-date printing).

2 Test Bed: Link Detection Issues

These issues concern content that has been successfully acquired (against all odds, considering the length of the previous section) and must now be parsed in order to detect links to other content. These issues are thus independent of a networking environment, as the content is already assumed locally accessible.

2.1 Character Encodings

2.1.1 Unicode (UTF-8, UTF-16), Asian, Cyrillic

A crawler should be tolerant of the variety of possible character encodings for text on the Internet.

2.1.2 Line ending issues (pathological cases affecting crawler parser)

A parser that treats a document as a series of lines separated by line end characters may become confused by variant line ending characters, which differ among the common platforms in use on the Web (Windows, UNIX, Mac).

2.2 HTML Parsing Issues

2.2.1 META refresh

HTML content may include a META tag that directs the browser to another page after a period of time.

2.2.2 BASE HREF

A BASE tag alters the sense of all relative links in a page.

2.2.3 LINK tags

LINK tags, which provide contextual linkages whose presentation may vary, should be crawled.

2.2.4 Tricky/absurd relative linkages (backslashes, "../../../")

The variety of legal and illegal relative links should be tested.

2.2.5 Internationalized domain names ("IDN")

A recent specification for encoding native-language host names to DNS-supported ASCII forms is supported by browser plugins. The crawler should be able to mimic this encoding.

2.3 HTML Form interpretation

Form submission is discussed as a retrieval issue in section 0. Other than their target URL, forms do not inherently contain linkage, but providing navigation tools is a common form usages.

2.3.1 Form with SELECT options containing links

A common pattern of navigation is to use an HTML form consisting of a SELECT popup widget, which contains site links as its various values, which are followed when users selects them either through JavaScript or by submitting the form to the server. This pattern should be identified and the links followed.

2.4 JavaScript

The category of JavaScript issues is impossible to completely characterize, because JavaScript, being a complete programming language, offers such broad functionality that its possibilities are too many to enumerate. Many sites with problematic JavaScript will include scripts combining some or all of these issues, which renders them hard to separate. In addition, to date there is no expectation of being able to implement a crawler that is able to execute the JavaScript in the manner of a browser, which would be essential to guaranteeing correctness. Regrettably, this all indicates that a test bed of problematic JavaScript code will need to be exceptionally broad to be able to characterize even a common subset of the scope of JavaScript usage.

2.4.1 Synthetic URLs

Many scripts construct a link to be followed by pasting it together programmatically from textual fragments; without executing the script, the link cannot be detected or reconstructed in the crawler.

2.4.2 Dynamic HTML menus

Many different patterns and code libraries exist for implementing dynamic pulldown-style menus. Synthetic URL creation is the underlying cause of most harvesting failures of this category.

2.4.3 Rollover images

Many different patterns and code libraries exist for implementing “rollovers”, which typically change visible images or play sounds in response to a user’s mouse movements. Synthetic URL creation is the underlying cause of most harvesting failures of this category.

2.4.4 Window creation

Many forms of script open a new browser window, in various ways, to present auxiliary information or bifurcate the page flow. Synthetic URL creation is the underlying cause of most harvesting failures of this category.

2.5 Synthesized Content

Sites that create content dynamically add a level of complexity to retrieval (as in case 0, for example), but not necessarily to link detection.

2.5.1 No-404 web sites

In particular, a site that returns a valid (HTTP 200 OK) synthesized page in the context of a “404 Not Found” page will be certain to confound a crawler, if any incorrect links appear on the site. Particularly problematic is the case of that non-404 page incorrectly containing relative links to itself, which might themselves not be found, leading to an infinite progression.

2.5.2 Crawler traps

Sites intentionally designed to trap a crawler by creating an infinite tree of synthetic documents may be addressed either by limited-depth crawling or by detecting characteristics of a trap. This test case is perhaps identical to a case that tests the ability of a crawler to process a site of truly enormous scope.

2.6 Non-HTML content with links

These common content types often contain links to other content, which it is reasonable to crawl.

2.6.1 PDF

The PDF format is reasonably well documented, and links should be detected in unencrypted documents.

2.6.2 XML (RSS/RDF)

In general, XML documents may contain URLs, though any given format may choose to specify them differently. Of particular interest are the increasingly popular syndication formats RSS and RDF, which are specifically designed for publishing sets of links, typically ones particularly relevant or current.

2.5.3 Office document formats

Common office document formats (such as Microsoft Office's tool set) can contain URLs in linked or non-linked forms. It is reasonable to crawl at least the linked URLs. A crawler would require specific knowledge of each document format.

2.7 Embedded content with links

The distinction between this category and category 0 is subtle at best, but we refer here to content that is presented in the context of a containing page in which it is embedded. Links may occur both in the embedding information (such as links to the plug-in for the content) and in the content itself.

2.7.1 Flash/Director

The Macromedia Flash format is proprietary but documented, and tools exist to read and understand it; linking is a standard attribute of this document type.

2.7.2 Java applets

The Java class format is documented and can be decompiled, but identifying links in applet code can be quite challenging, as the number of ways a link could be expressed is uncountable, and links may be constructed in Java code in the same way as in JavaScript (0).

3 Presentation Issues

Though the test bed should defer, for the present, any attempt to validate presentation issues, we choose to document them here, specifically to indicate that they are deferred. Our test cases for the prior categories will most likely serve to illustrate the following issues as well.

The primary function of a presentation tool is to present a self-consistent view of a harvested set of site, which is achieved by manipulating links contained within each piece of content in order to re-orient them within the harvested archive rather than outward to the current Internet.

3.1 JavaScript link rewriting

Links created synthetically (as described in section 0) are not only difficult to detect, but also to correctly rewrite, for the same reasons. The JavaScript must actually be executed by a piece of software in order to find the text of the link, which is beyond the scope of current tools.

3.2 Non-HTML link rewriting

Formats such as Flash or Java applets often contain links, particularly when used to present rich navigation features. A presentation tool must adjust the links in this content as well, which is logically the same task as rewriting HTML links, but more technically challenging, as the file formats are more complex.

3.3 Optional link rewriting

Related to the prior case is the problem of link rewriting in content for which it may be incorrect, such as PDF or word processor documents. Consider an archived report which discusses web sites; it would be confusing to the reader if each URL reference in the document were updated to refer back to the archived versions. The boundary dividing documents which should and should not have links rewritten is unclear.

3.4 Server-connected content

Rich content types such as Java applets or Flash may create connections back to their host server in order to retrieve dynamic information such as database content.