Availability and Preservation of Scholarly Digital Resources

Jason Hennessey
South Dakota State University

AVAILABILITY AND PRESERVATION OF SCHOLARLY DIGITAL RESOURCES

BY

JASON HENNESSEY

A dissertation submitted in partial fulfillment of the requirements for the

Doctor of Philosophy

Major in Computational Science and Statistics

South Dakota State University

2014

AVAILABILITY AND PRESERVATION OF SCHOLARLY DIGITAL RESOURCES

This dissertation is approved as a creditable and independent investigation by a candidate for the Doctor of Philosophy in Computational Science and Statistics degree and is acceptable for meeting the dissertation requirements for this degree Acceptance of this does not imply that the conclusions reached by the candidates are necessarily the conclusions of the major department

Xijin Ge, Ph D Date
Thesis Advisor

Kurt Cogswell, Ph D Date
Head, Department of Mathematics & Statistics

Dean, Graduate School Date

ACKNOWLEDGEMENTS

We do not exist in a vacuum; there are many, some named, some unnamed and some unknown, to whom I owe a great debt of gratitude for helping me during this stage of life. I would like to thank:

Most of all, my God for saving me from death and giving me life; with him all things are possible.

Especially my advisor, mentor and committee chair, Dr. Xijin Ge, for his wisdom, patience and guidance as well as the other members of my committee for the same: Drs. Michael Hildreth, George Hamer, Matthew Biesecker and Donald Auger.

Dr. Steve Strassmann for his idea on using VMs to help the scientific community.

The wonderful Math & Statistics department faculty who shared their time and wisdom at various points along the way: Drs. Kurt Cogswell (department chair), Thomas Brandenburger, Gemechis Djira and Christopher Saunders.

The many faculty at South Dakota State in other departments who spent time beyond that required to convey knowledge, wisdom and friendship, among whom were: Drs. Volker Brözel, Buyung Hadi and CY Wang.

My fellow students, who offered their camaraderie and help in various ways: James Ban, Adam Schmitz, Brian Vachta and Matthew Whipple.

Last and far from least, my mother, who taught me about sacrificial love despite enduring my teenage years.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

DOI	Digital Object Identifier
IA	Internet Archive (http://archive.org)
IP	Intellectual Property
PURL	Persistent Uniform Resource Locator
URL	Uniform Resource Locator
VM	Virtual Machine
VMDK	Virtual Machine DisK
WC	WebCite (http://www.webcitation.org/)
WOS	Thomson Reuters' Web of Science (http://webofknowledge.com/WOS)
WWW	World Wide Web

ABSTRACT

AVAILABILITY AND PRESERVATION OF SCHOLARLY DIGITAL RESOURCES

JASON HENNESSEY

2014

The dynamic, decentralized world-wide-web has become an essential part of scientific research and communication, representing a relatively new medium for the conveyance of scientific thought and discovery. Researchers create thousands of web sites every year to share software, data and services. Unlike books and journals, however, the preservation systems are not yet mature. This carries implications that go to the core of science: the ability to examine another's sources to understand and reproduce their work. These valuable resources have been documented as disappearing over time in several subject areas. This dissertation examines the problem by performing a cross-disciplinary investigation, testing the effectiveness of existing remedies and introducing new ones.

As part of the investigation, 14,489 unique web pages found in the abstracts within Thomson Reuters' Web of Science citation index were accessed. The median lifespan of these web pages was found to be 9.3 years with 62% of them being archived. Survival analysis and logistic regression identified significant predictors of URL lifespan and included the year a URL was published, the number of times it was cited, its depth as well as its domain. Statistical analysis revealed biases in current static web-page solutions.

A prototype has been created to submit static web pages to the archives. It was quite successful, increasing coverage of the scientific webpages in the Internet Archive and WebCite by 22% and 255%, respectively.

Another prototype, Logic Capsule, was created to facilitate the combination of both data and logic into a preserved and searchable archive of Virtual Machines. Were this to be widely adopted, it could represent a dramatic step forward in preserving these tools in their original habitat, reproducing the statistical analyses, interactive web applications and other computer-based work of others. It would also permit scientists to make use of complex software stacks without expertise in the underlying technologies.

Disappearing digital resources continue to be a problem, though existing remedies for static web pages are addressing these problems well. Using an automated submission tool can markedly improve the archival engines' coverage of scholarly URLs. Logic Capsule represents an improved solution for sites with server-based logic and covers a gap in the currently deployed archival methods.

CHAPTER 1 - INTRODUCTION AND OVERVIEW

Reproducing the work of others is a time-honored tradition that forms a basic pillar of science. So also is the passing of knowledge and the sources relied upon to gain that knowledge, allowing many generations to see further by "standing on the shoulders of giants". For the previous few millennia, this recorded knowledge has been passed and widely disseminated using written media such as books and journals, and society has generally learned how to archive and make available this knowledge. Over the past two decades, a new medium for the conveyance of information has become popular, the World Wide Web (WWW), a subset of the Internet's functionality. It is so popular in fact that 2 of the 5 most cited papers from the previous decade[1] included Internet resources.

This new medium has vastly increased the speed, efficiency and efficacy of the propagation of knowledge. But being relatively new, there are immaturities that raise certain questions. What is the prevalence in modern science of producing WWW-based resources? Do they disappear? What are the current archiving mechanisms? Are there gaps in their coverage? If so, what are some solutions? This dissertation attempts to respond to these questions.

In order to produce a comprehensive cross-disciplinary study of scientific online resources and their current preservation, in Chapter 2 an analysis of the problem is undertaken where URLs were extracted from the corpus of a very large bibliographic database, Thomson Reuters' Web of Science (WOS). It is in this document that the

[1] "MEGA4: Molecular evolutionary genetics analysis (MEGA) software version 4.0" and "The Protein Data Bank". Citation count based on Web of Science

current prevalence of the disappearance problem is addressed, as well as an analysis of its pattern and an assessment of the effectiveness of the current solutions. An enhancement is then proposed to compliment the current archival mechanisms.

In 0, a gap in the current solutions, the packaging and archiving of scholarly resources with data and complex logic, is addressed with a novel proposal. This proposal also addresses needs that have been called for by many in the scientific community, reproducible research. Finally, in **Error! Reference source not found.**, the findings are summarized , conclusions are put forward and future work is discussed.

Importance of online resources

How important are online resources? Quantitative evaluation can be viewed from a couple of perspectives. On the one hand, one can look at the steadily rising number and proportion of peer-reviewed, scholarly publications containing an Internet-based resource (see [2] and Figure 2.1). On the other, one can look at the citations received by the publishing papers. Among the papers publishing a URL that were surveyed in Chapter 2, the average number of citations was 29, with the median being 6. The maximum number of citations for a single paper was 9,076[2].

Qualitatively speaking, Dimitrova and Bugeja note that "The goal of rhetoric is to persuade. The goal of the footnote is to prove."[3], indicating that the veracity of a work of scholarship can be examined by viewing its citations. After discussing how Bacon and Locke's works provoked Enlightenment thinkers to reconsider the source, they concluded that changing or disappearing Internet resources attack the heart of scholarship and

[2]"The Protein Data Bank". Nucleic Acids Research, 2000.

research by destabilizing our fixed language and original source material. Because websites are in some cases used as a modern equivalent to the classic footnote, they can embody the basis upon which further scholarship and ideas are based.

Disappearing research

The problem of disappearing online resources has been documented in many specific subject areas, with Table 2.1 containing a large list of these subject-specific studies. In terms of wide, cross-disciplinary analyses, the closest thus far are those of the biological and medical MEDLINE and PubMed databases by Ducut [2] and Wren [4, 5], in addition to Yang’s study of the Social Sciences within the Chinese Social Sciences Citation Index[6].

Comparing the Internet to H.G. Wells' "World Brain" in the conclusion to his 1999 study, Koehler commented that if the Internet were the world brain, it could be seen to have a short memory and change its mind a lot[7]. In his study and witty commentary, Koehler drew attention to the fact that not only do Internet-based resources disappear, but they also change. This phenomenon is quite eloquently demonstrated by a study showing that at the time of publication, 12% of Internet-based citations had already ceased to function[8].

Many reasons exist why online resources could disappear. Some of it may be scholars leaving institutions (especially likely with graduate students), losing their account. Similarly, a project's funding could be lost. A site with widely-used scientific resources could be shut down in the process of an effort to consolidate servers. Though not as likely as in the early days of the WWW, some people could be serving websites from their desktop workstation which they then shut down when they go home on the weekend. Are missing resources important?

Evaluating the importance and impact of missing resources is an ongoing area of research. According to Wren, link decay has only been documented as a general trend; whether "important" URLs are affected more than others is unknown[4]. Quantitatively speaking, in response to a survey sent to corresponding authors of missing online resources in the field of Dermatology, the majority (55%) indicated that the missing information was important to their publication[9]. Examining the data in Chapter 2, out of the ten most cited papers containing a URL in the abstract, three link to websites that are no longer available[3].

Many scientists find it difficult to reproduce the work of others, especially the complex statistical analyses that are common in modern research[10]. This led to the growing Reproducible Research movement, where scientists are encouraged to package their findings in a way that allows another to readily replicate their work. Recognizing the widespread impact of the problem, a 2011 issue of the cross-disciplinary journal Science published a special section calling attention to the problem[11]. In [12], Wicherts relates reproducibility to aviation: a co-pilot can check every action of the captain and there is a black box that records each action. He also demonstrated that the unwillingness of authors to share data in his field (psychology) is associated with weaker statistical results as well as more errors[13]. Several helpful suggestions have come out of this widespread push for reproducible research, some of which are discussed in 0.

[3] These are http://www.stats.ox.ac.uk/~pritch/home.html (4349 citations; published in 2000), http://www.lirmm.fr/w3ifa/MAAS/ (3680 citations; 2003) and http://www.cbs.dtu.dk/services/SignaIP/ (2703 citations; 2004)

Existing methods of preservation

There are several ways that an online resource can disappear. Correspondingly, there are several methods to address it which are not exclusive. For websites that are still on the Internet but have simply changed locations (for example, http://www.sdstate.edu moving to http://www2.sdstate.edu), redirection services exist. The simplest form of this is when one has access to the former web server, where a URL redirection can be used to seamlessly send the browser to the new location using a variety of methods. Some of these methods consist of using the HTTP protocol itself, specifically the 301 and 302 statuses as well as the "Location" header, as well as using browser-implemented methods such as the "Refresh" metatag or JavaScript. Dedicated redirecting services, such as the Digital Object Identifier (DOI) System [14] and Persistent Uniform Resource Locator (PURL) [15], provide a mechanism where one can pre-register a fixed URL that then redirects the user to the destination site using the same methods outlined above. These methods, combined with other mechanisms that can help locate resources (such as using a web search engine) could help address 30-60% of the cases of link rot [9, 16].

Other techniques exist to save static web pages which might help roughly 40% of missing resources[4]. Static pages are those which do not depend heavily on server logic, which would easily be represented by a printout. Examples include methods pages, tutorials and data sets. While some researchers preserve these resources using manual methods, such as saving them to hard drives or printing them out, two tools exist to solve this problem in a centralized fashion. The first one, the Internet Archive (IA) [17], employs an algorithm that crawls the Internet at large, storing snapshots of pages it encounters along the way and has been operating since the mid-1990s. The second, WebCite (WC) [18], stores pages upon request and targets the scientific community and seeks to partner with publishers. The study in Chapter 2 showed that these two tools rescued 49% of published URLs from WOS that were missing.

Several recommendations for addressing reproducible research concerns have been made. The first (and likely easiest) is to simply include the code used for computation as part of a document's supplement[19], though even with this there may exist complex configurations which require expertise. In [20], Peng outlines a spectrum of reproducibility. At the low end is the classical publication, with its textual description, followed by the progression of: including the source code, the code & data, a ready-to-go executable form of the code and data and finally, what Peng refers to as the "Gold standard", complete replication of the experiment.

Methods proposed in this dissertation

Two new methods are proposed and prototyped in this dissertation for enhancing the availability of online scholarly research. In Chapter 2, a new mechanism is proposed and tested that augments the current static-page archival mechanisms (IA and WC) by proactively submitting published URLs which have not been archived. This method showed great success, increasing IA's coverage of the URL list by 22% and WebCite's by 255%!

Bibliography

2. Ducut E, Liu F, Fontelo P: **An update on Uniform Resource Locator (URL) decay in MEDLINE abstracts and measures for its mitigation.** *BMC Med Inform Decis Mak* 2008, **8:**-.
3. Dimitrova DV, Bugeja M: **Consider the source: Predictors of online citation permanence in communication journals.** *Portal-Libraries and the Academy* 2006, **6:**269-283.
4. Wren JD: **URL decay in MEDLINE - a 4-year follow-up study.** *Bioinformatics* 2008, **24:**1381-1385.
5. Wren JD: **404 not found: the stability and persistence of URLs published in MEDLINE.** *Bioinformatics* 2004, **20:**668-U208.
6. Yang SL, Qiu JP, Xiong ZY: **An empirical study on the utilization of web academic resources in humanities and social sciences based on web citations.** *Scientometrics* 2010, **84:**1-19.
7. Koehler W: **An analysis of Web page and Web site constancy and permanence.** *J Am Soc Inf Sci* 1999, **50:**162-180.
8. Aronsky D, Madani S, Carnevale RJ, Duda S, Feyder MT: **The prevalence and inaccessibility of Internet references in the biomedical literature at the time of publication.** *J Am Med Inform Assn* 2007, **14:**232-234.
9. Wren JD, Johnson KR, Crockett DM, Heilig LF, Schilling LM, Dellavalle RP: **Uniform resource locator decay in dermatology journals - Author attitudes and preservation practices.** *Arch Dermatol* 2006, **142:**1147-1152.
10. Ioannidis JPA, Allison DB, Ball CA, Coulibaly I, Cui XQ, Culhane AC, Falchi M, Furlanello C, Game L, Jurman G, et al: **Repeatability of published microarray gene expression analyses.** *Nature Genetics* 2009, **41:**149-155.
11. Jasny BR, Chin G, Chong L, Vignieri S: **Again, and Again, and Again ….** *Science* 2011, **334:**1225.
12. Wicherts JM: **Psychology must learn a lesson from fraud case.** *Nature* 2011, **480:**7-7.
13. Wicherts JM, Bakker M, Molenaar D: **Willingness to Share Research Data Is Related to the Strength of the Evidence and the Quality of Reporting of Statistical Results.** *Plos One* 2011, **6:**e26828.
14. **The DOI System** [http://www.doi.org/]
15. **PURL Home Page** [http://purl.org]
16. Casserly MF, Bird JE: **Web citation availability: Analysis and implications for scholarship.** *College & Research Libraries* 2003, **64:**300-317.
17. **The Internet Archive** [http://www.archive.org/web/web.php]
18. Eysenbach G, Trudell M: **Going, going, still there: Using the WebCite service to permanently archive cited web pages.** *Journal of Medical Internet Research* 2005, **7:**2-6.
19. Barnes N: **Publish your computer code: it is good enough.** *Nature* 2010, **467:**753-753.
20. Peng RD: **Reproducible Research in Computational Science.** *Science* 2011, **334:**1226-1227.

CHAPTER 2 – LINK DECAY

A cross disciplinary study of link decay and the effectiveness of mitigation techniques[4]

Abstract

Background

The dynamic, decentralized world-wide-web has become an essential part of scientific research and communication. Researchers create thousands of web sites every year to share software, data and services. These valuable resources tend to disappear over time. The problem has been documented in many subject areas. Our goal is to conduct a cross-disciplinary investigation of the problem and test the effectiveness of existing remedies.

Results

We accessed 14,489 unique web pages found in the abstracts within Thomson Reuters' Web of Science citation index that were published between 1996 and 2010 and

[4] This chapter is based on a paper by Jason Hennessey & Xijin Ge published in the conference proceedings of the 2013 Mid-South Computational Bioinformatics Society, a peer-reviewed supplement published through BMC Bioinformatics. It may be accessed at: http://www.biomedcentral.com/1471-2105/14/S14/S5/

found that the median lifespan of these web pages was 9.3 years with 62% of them being archived. Survival analysis and logistic regression were used to find significant predictors of Universal Resource Locator (URL) lifespan. The availability of a web page is most dependent on the time it is published and the top-level domain names. Similar statistical analysis revealed biases in current solutions: the Internet Archive favors web pages with fewer layers in the URL while WebCite is significantly influenced by the source of publication. We also created a prototype for a process to submit web pages to the archives and increased coverage of our list of scientific webpages in the Internet Archive and WebCite by 22% and 255%, respectively.

Conclusion

Our results show that link decay continues to be a problem across different disciplines and that current solutions for static web pages are helping and can be improved.

Background

Scholarly Internet resources play an increasingly important role in modern research. We can see this by the increasing number of URLs published in a paper’s title or abstract [2](also see Figure 2.1). As the Internet is a relatively new medium for communicating scientific thought, the community is still figuring out how best to use it in a way that preserves contributions for years to come. One problem is that continued availability of these online resources is at the mercy of the organizations or individuals that host them. Many disappear after publication (and some even disappear before[8]), leading to a well-documented phenomenon referred to as link rot or link decay.

The problem has been documented in several subject areas, with Table 2.1 containing a large list of these subject-specific studies. The URLs accounting for these losses come from both peer-reviewed titles/abstracts as well as direct references from individual publications. In terms of wide, cross-disciplinary analyses, the closest thus far are those of the biological and medical MEDLINE and PubMed databases by Ducut [2] and Wren [4, 5], in addition to Yang’s study of the Social Sciences within the Chinese Social Sciences Citation Index[6].

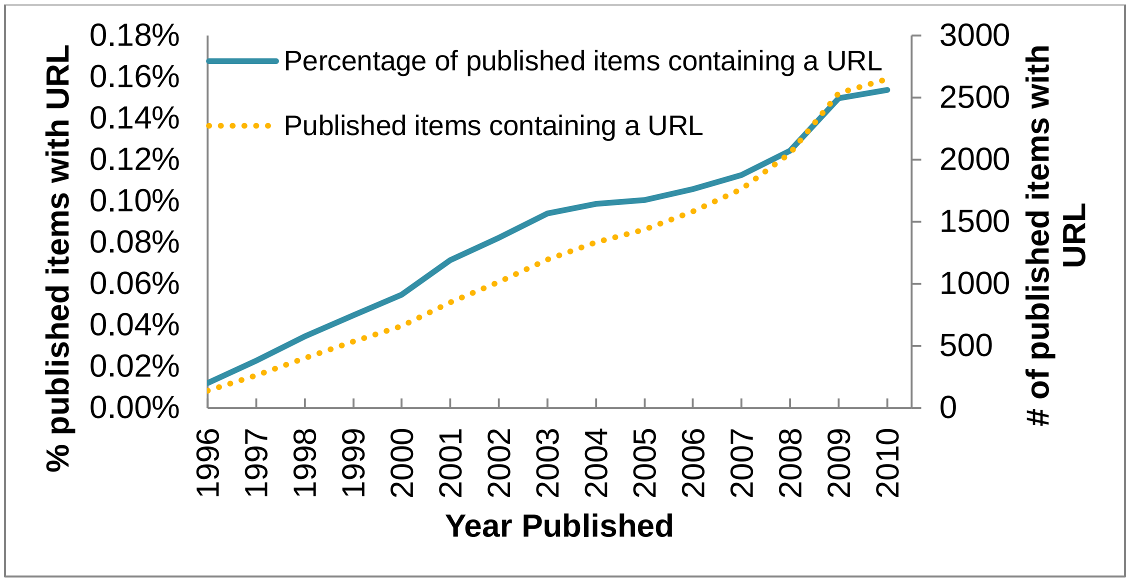


Figure 2.1 - Growth of scholarly online resources. Not only are the number of URL-containing articles (those with "http" in the title or abstract) published per year increasing (dotted line), but also the percentage of published items containing URLs (solid line). The annual increase in articles according to a linear fit was 174 with R2 0.97. The linear trend for the percentage was an increase of 0.010% per year with R2 0.98.
Source: Thomas Reuter's Web of Science

Some solutions have been proposed which attack the problem from different angles. The Internet Archive (IA) [17] and WebCite (WC) [18] address the issue by archiving web pages, though their mechanisms for acquiring those pages differ. The IA, beginning from a partnership with the Alexa search engine, employs an algorithm that crawls the Internet at large, storing snapshots of pages it encounters along the way. In contrast, WebCite archives only those pages which are submitted to it, and it is geared toward the scientific community. These two methods, however, can only capture information that is visible from the client. Logic and data housed on the server are not frequently available.

Other tools, like the Digital Object Identifier (DOI) System [14] and Persistent Uniform Resource Locator (PURL) [15], provide solutions for when a web resource is moved to a different URL but is still available. The DOI System was created by an international consortium of organizations wishing to assign unique identifiers to items such as movies, television shows, books, journal articles, web sites and data sets. It encompasses several thousand "Naming Authorities" organized under a few "Registration Agencies" that have a lot of flexibility in their business models[21]. Perhaps 30-60% of link rot could be solved using DOIs and PURLs[9, 16]. However they are not without pitfalls. One is that a researcher or company could stop caring about a particular tool for various reasons and thus not be interested in updating its permanent identifier. Another is that the one wanting the permanent URL (the publishing author) is frequently not the same as the person administering the site itself over the long term, thus we have an imbalance of desire vs. responsibilities between the two parties. A third in the case of the DOI System is that there may be a cost in terms of money and time associated with registering their organization that could be prohibitive to authors that don't already have access to a Naming Authority[2]. One example of a DOI System business model would be that of the California Digital Library's EZID service, which charges a flat rate (currently $2,500 for a research institution) for up to 1 million DOIs per year[22].

In this study, we ask two questions: what are the problem’s characteristics in scientific literature as a whole and how is it being addressed? To assess progress in combating the problem, we evaluate the effectiveness of the two most prevalent preservation engines: and examine the effectiveness of one prototyped solution. If a URL is published in the abstract, it is assumed that the URL plays a prominent role within that paper, similar to the rationale proposed by Wren [5].

Table 2.1 - Link decay has been studied for several years in specific areas. * denotes studies most similar to the current.

Field	Links Source/Type	Year(s) of URLs	N	Citation(s)
Biology & Medicine	Science curriculum web links	2000	515	[23]
	Full text of 3 dermatology journals	1999-2004	1113	[9]
	Sample of bibliographies being published on PubMed	2006	840	[8]
	References made in the *Annals of Emergency Medicine*	2000, 2003, 2005	586	[24]
	References in 5 biomedical informatics journals.	1999-2004	1049	[25]
	MEDLINE titles & abstracts	1994-2006	10208	[2]*
	Internet citations in 5 health care management journals from 2002-2004	2009-2010	2011	[26]
	MEDLINE abstracts	1995-2007	7462	[4]*
Communications	Citations appearing in research articles in 6 leading communications journals	2000-2003	1600	[3]
Ecology	URLs appearing in the full text of 4 Ecological Society of America journals	1997-2005	2100	[27]
Law	Samples from a collection of born-digital law- and policy-related reports and documents	2007-2010	2372	[28]
Library / Information Science	Citations appearing in 3 leading Information Science journals	1997-2003	2516	[29]
	Sample of citations appearing in library and information science journals	1999-2000	500	[30]
Social Sciences	URLs appearing in the full text of 2 well-respected historical journals	1999-2006	510	[31]
	Citations from articles in the Chinese Social Sciences Index	1998-2007	44973	[6]*
Various	Random Collection of web URLs	1996	371	[7, 32]
Various	Citations in 3 highly circulated journals	2002-2003	672	[33]
Various	Supplementary information published in 6 top-cited journals	2000, 2003	585	[34]
Various	Citations from conference articles	1995-2003	1068	[35]
Various Collections				[36-39]

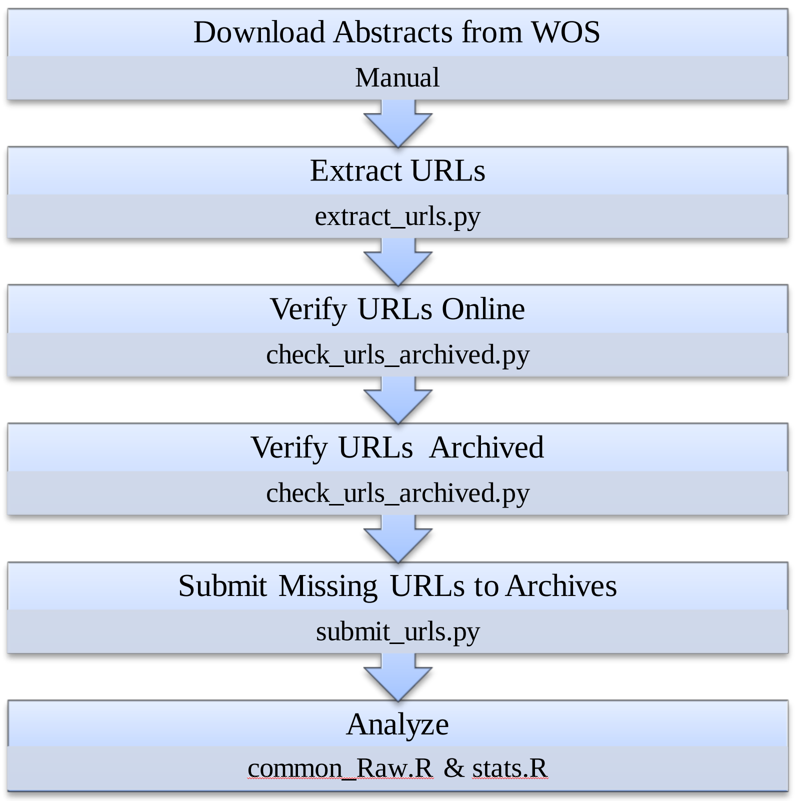


Figure 2.2 - Flowchart of the study procedures. Beneath each step is the method primarily used to execute it. Those ending in .py are Python programs whereas those ending in .R use the R language.

Results

Our goals are to provide some metrics that are useful in understanding the problem of link decay in a cross-disciplinary fashion and to examine the effectiveness of the existing archival methods while proposing some incremental improvements. To accomplish these tasks, we downloaded 18,231 Web of Science (WOS) abstracts containing "http" in the title or abstract from the years under study (1996-2010), out of which 17,110 URLs (14,489 unique) were extracted and used. We developed Python scripts to access these URLs over a 30-day period. For the period studied, 69% of the published URLs (67% of the unique) were available on the live Internet, the Internet Archive's Wayback Machine had archived 62% (59% unique) of the total and WebCite had 21% (16% unique). Overall, 65% of all URLs (62% unique) were available from one of the two surveyed archival engines. Figure 2.3 contains a breakdown by year for availability on the live web as well as through the combined archives, and illustrates each archival engine's coverage. The median lifetime for published URLs was found to be 9.3 years (95% CI [9.3,10.0]), with the median lifetime amongst unique URLs also being 9.3 years (95% CI [9.3,9.3]). Subject-specific lifetimes may be found in Table 2.2. Using a simple linear model, the chances that a URL published in a particular year is still available goes down by 3.7% for each year added to its age with an R^2 of 0.96. Its chances of being archived go up after an initial period of flux (see Figure 2.3). Submitting our list of unarchived but living URLs to the archival engines showed dramatic promise, increasing the Internet Archive's coverage of the dataset by 2080 URLs, an increase of 22%, and WebCite's by 6348, an increase of 255%.

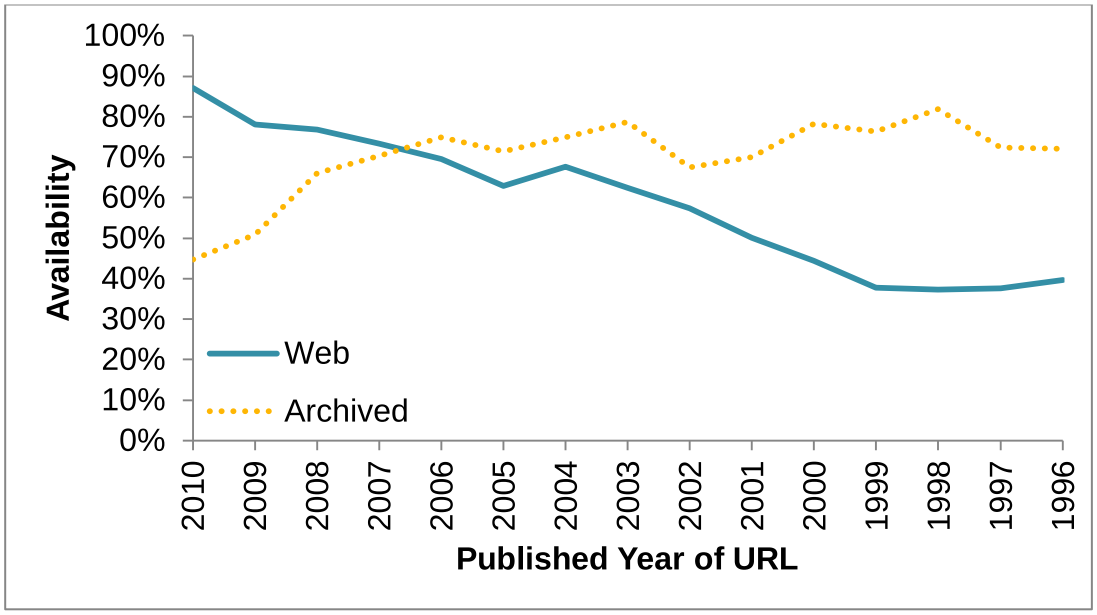


Figure 2.3 - Accessibility of URLs highly correlated with publishing year. The probability of being available (solid line) declines by 3.7% every year based on a linear model with R^2 0.96. The surveyed archival engines have about a 70-80% archival rate (dotted line) following an initial ramp time.

How common are published, scholarly online resources? According to WOS, both the percentage of published items which contain a URL as well as their absolute number has increased steadily from 1996 until 2010 as seen in Figure 2.1. A simple linear fits show the URL proportion's annual increase to be a conservative 0.010 % per year with an R^2 of 0.98, while the absolute number increases by 174 papers with an R^2 of 0.97.

A total of 189 (167 unique) DOI URLs were identified, consisting of 1% of the total, while 9 PURLs (8 unique) were identified. Due to cost[26], it is likely that DOIs will remain useful for tracking commercially published content though not the scholarly online items independent of those publishers.

Table 2.2 - Comparison of certain statistics based on a URL's subject. Subjects are assigned to journals and not specific papers. Note that in these models, a given URL could contribute to multiple subjects due to appearing in multiple journals which could also have multiple subject areas. Where possible, specific subjects were generalized (for example, "Computer Science, Interdisciplinary Applications" became "Computer Science"). Median survival estimated using R's survfit(). "NA" indicates that an upper 95% limit was unable to be computed.

Subject	**Total**	**# Alive (%)**	**Median Survival with 95% CI in years**
Biochemistry & Molecular Biology	4585	3231 (70%)	10.8 (9.0,11.0)
Biotechnology & Applied Microbiology	2225	1586 (71%)	9.0 (8.8,9.0)
Computer Science	2073	1225 (59%)	8.3 (7.0,9.0)
Biochemical Research Methods	2023	1463 (72%)	8.5 (8.5,8.6)
Mathematical & Computational Biology	1661	1200 (72%)	7.5 (7.5,9.0)
Genetics & Heredity	1302	914 (70%)	8.8 (8.8,10.0)
Physics	809	458 (57%)	8.0 (7.6,9.0)
Engineering	703	419 (60%)	7.2 (7.1,10.5)
Statistics & Probability	699	440 (63%)	7.6 (7.0,9.0)
Chemistry	591	397 (67%)	11.4 (9.0,11.9)
Biophysics	432	270 (63%)	10.1 (10.1,10.1)
Astronomy & Astrophysics	416	268 (64%)	11.3 (11.1,NA)
Mathematics	406	254 (63%)	10.7 (4.5,NA)
Zoology	357	319 (89%)	11.2 (9.6,NA)
Cell Biology	353	242 (69%)	8.0 (8.0,10.8)
Biology	346	242 (70%)	9.8 (7.3,NA)
Oncology	342	239 (70%)	6.9 (6.9,7.0)
Plant Sciences	315	235 (75%)	9.8 (8.2,NA)
Environmental Sciences	304	190 (63%)	8.0 (7.6,9.5)
Medicine	293	219 (75%)	13.3 (10.0,NA)

$$\mu = e^{x_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_3 + \ldots + \beta_n x_n}$$

$$S(t{:}\,\mu, s) = 1 - \frac{1}{1 + e^{-\frac{t-\mu}{s}}}$$

Equation 2.1 - Calculations for approximating the median (μ) survival time as well as the survival function *S()* using a logistic parametric model. The survival function is the probability of an individual surviving beyond time t [40]. x_0 is the intercept, with $x_1, x_2, \ldots$ being predictors. s is a scale parameter, in this case found to be 6.79.

URL survival

In order to shed some light on the underlying phenomena of link rot, a survival regression model was fitted with data from the unique URLs. This model, shown in

Table 2.3, identified 17 top-level domains, the number of times a URL has been published, a URL's directory structure depth (hereafter referred to as "depth", using the same definition as [7]), the number of times the publishing article(s) has been cited, whether articles contain funding text as well as 4 journals as having a significant impact on a URL's lifetime at the $P < 0.001$ level. This survival regression used the logistic distribution and is interpreted similarly to logistic models as shown in Equation 2.1. To determine the predicted outcome for a particular URL, one takes the intercept (5.2) and adds to it the coefficients for the individual predictors if those predictors are different from the base level; coefficients here are given in years. If numeric, one first multiplies before adding. The result is then interpreted as the location of the peak of a bell curve for the expected lifetime, instead of a log odds ratio as a regular logistic model would give. Among URL domains, org and dk hadn't the largest positive influence by adding about 8 years while kr had the largest negative effect, subtracting 3, though with a relatively smaller p value of .02. Between journals, Zoological Studies had the largest positive impact on lifetime, adding 16 years, whereas *Computer Physics Communications* had the largest negative impact, subtracting 4 years.

Table 2.3 - Results of fitting a survival regression to the unique URLs. Positive numbers indicate longer median lifetimes. Much like a logistic model, coefficients can be added to the intercept value (after multiplying in the case of numeric predictors) to obtain a median lifetime. For example, the median expected lifetime for a URL published once, with depth 0, whose publishing article had 1 citation, no funding text, domain au and published in a Journal not listed (ie- in the default) would be: (Intercept) 5.22 + Log2(1)*3.57 + 0*-1.46 + Log2(1+1)*0.25 + 0*3.43 + 4.53 = 10 years

Variable	Value	p	5%	95%
(Intercept)	5.22	3.3E-30	4.46	5.97
Log2(URL published)	3.57	1.4E-17	2.88	4.25
depth	-1.46	7.0E-32	-1.66	-1.25
Log2(TimesCited + 1)	0.25	2.8E-04	0.13	0.36
Funding text present	3.43	2.8E-11	2.59	4.28
Domain				
au	4.53	1.5E-04	2.56	6.49
be	3.31	1.9E-02	0.99	5.64
ca	4.88	1.7E-06	3.20	6.56
ch	6.45	7.2E-08	4.48	8.42
cn	1.50	1.3E-01	-0.13	3.13
com	6.02	2.2E-18	4.89	7.16
de	5.74	6.1E-16	4.57	6.91
dk	7.66	5.7E-07	5.14	10.18
edu	3.77	1.6E-13	2.93	4.61
es	3.05	5.4E-03	1.25	4.85
fr	3.65	6.6E-07	2.44	4.85
gov	5.51	1.2E-15	4.38	6.64
il	5.92	3.6E-04	3.19	8.65
in	4.78	2.2E-04	2.65	6.91
it	5.51	1.4E-08	3.91	7.11
jp	5.07	8.0E-09	3.62	6.51
kr	-3.35	2.0E-02	-5.73	-0.97
net	7.01	4.2E-11	5.26	8.76
nl	6.78	1.1E-06	4.49	9.07
org	8.10	2.4E-36	7.04	9.16
ru	3.90	2.3E-03	1.80	6.01

se	1.71	2.4E-01	-0.69	4.12
tw	1.64	1.7E-01	-0.33	3.61
uk	4.49	4.2E-12	3.42	5.56
Source				
Bioinformatics	-2.04	5.7E-03	-3.25	-0.83
BMC Bioinformatics	2.69	3.9E-05	1.62	3.77
BMC Genomics	0.88	4.7E-01	-1.13	2.89
Comp. Physics Comm.	-4.00	3.0E-05	-5.57	-2.42
Genome Research	0.56	7.1E-01	-1.92	3.04
Nucleic Acids Research	1.28	8.6E-04	0.65	1.91
PLoS ONE	-0.39	8.0E-01	-2.95	2.18
Zoological Studies	16.42	2.2E-15	13.01	19.83

Predictors of availability

While examining URL survival and archival, it is not only interesting to ask which factors significantly correlate with a URL lasting but also which account for most of the differences. To that end, we fit logistic models for each of the measured outcomes (live web, Internet Archive and Web Citation availabilities) to help tease out that information. To enhance comparability, a similar list of predictors (differing only in whether the first or last year a URL was published was used) without interaction terms was employed for all 3 methods and unique deviance calculated by dropping each term from the model and measuring the change in residual deviance. Results were then expressed as a percentage of the total uniquely explained deviance and are graphically shown in Figure 2.4.

For live web availability, the most deviance was explained by the last year a URL was published (42%) followed by the domain (26%). That these two predictors are very important agrees with much of the published literature thus far. For the Internet Archive, by far the most important predictor was the URL depth at 45%. Based on this, it stands to reason that the Internet Archive either prefers more popular URLs which happen to be at lower depths or employs an algorithm that prioritizes breadth over depth. Similar to the IA, WC had a single predictor that accounted for much of the explained deviance, with the publishing journal representing 49% of the explained deviance. This may reflect WC's efforts to work with publishers as the model shows one of the announced early adopters, BioMed Central [18], as having the two measured journals (BMC Bioinformatics and BMC Genomics) with the highest retention rates. Therefore, WC is biased towards a publication's source (journals).

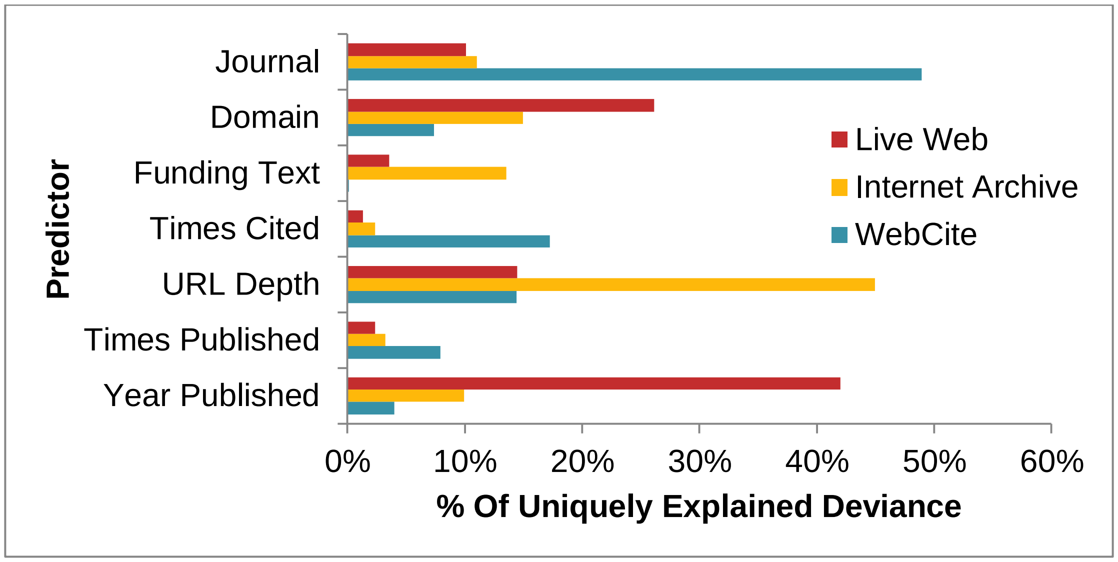


Figure 2.4 – Predictor importance for URL availability. This graph compares what portion of the overall deviance is explained uniquely by each predictor for each of the measured outcomes. A similar list of predictors (differing only in whether the first or last year a URL was published) without interaction terms was employed to construct 3 logistic regression models. The dependent variable for each of the outcomes under study (Live Web, Internet Archive and WebCite) was availability at the time of measurement. Unique deviance was calculated by dropping each term and measuring the change in explained deviance in the logistic model. Results were then expressed as a percentage of the total uniquely explained deviance for each of the 3 methods.

Archive site performance

Another way to measure the effectiveness of the current solutions to link decay is to look at the number of "saved" URLs, or those missing ones that are available through archival engines. Out of the 31% of URLs (33% of the unique) which were not accessible on the live web, 49% of them (47% of the unique) were available in one of the two engines, with IA having 47% (46% unique) and WC having 7% (6% unique). WC's comparatively lower performance can likely be attributed to a combination of its requirement for human interaction and its still-growing adoption.

In order to address the discrepancy, all sites that were still active but not archived were submitted to the engines from which they were missing. Using the information gleaned from probing the sites as well as the archives, URLs missing from one or both of the archives, yet still alive, were submitted programmatically. This included submitting 2,662 to the Wayback Machine as well as 7,477 to WebCite, of which 2,080 and 6,348 were successful, respectively.

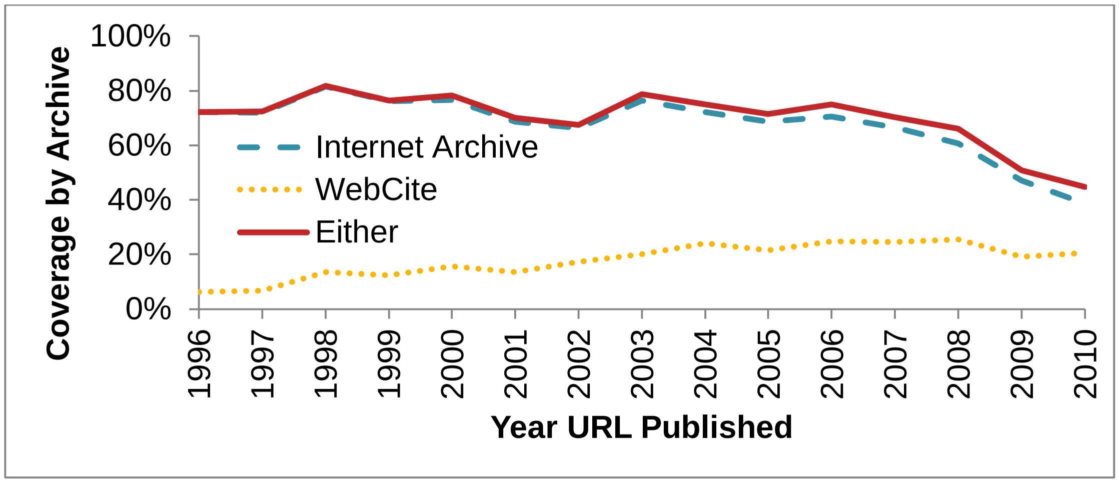


Figure 2.5 - URL presence in the archives. Percentage of URLs found in the archives of the Internet Archive (dashed line), WebCite (dotted line) or in any group (solid line). IA is older, and thus accounts for the lion's share of earlier published URLs, though as time goes on WebCite is offering more and more.

Discussion

Submission of missing URLs to archives

Archiving missing URLs in each of the archival engines had their own special nuances. For the Internet Archive, the lack of a practical documented way of submitting URLs (see http://faq.web.archive.org/my-sites-not-archived-how-can-i-add-it/) necessitated trusting a message shown by the Wayback Machine when one finds a URL that isn't archived and clicks the "Latest" button. In this instance, the user is sent to the URL "http://liveweb.archive.org/<url>" which has a banner proclaiming that the page "will become part of the permanent archive in the next few months". Interestingly, as witnessed by requests for a web page hosted on a server for which the authors could monitor the logs, only those items requested by the client were downloaded. This meant that if only a page's text were fetched, supporting items such as images and CSS files would not be archived. To archive the supporting items and avoid duplicating work, wget's "—page-requisites" option was used instead of a custom parser.

WebCite has an easy-to-use API for submitting URLs, though limitations during the submission of our dataset presented some issues. The biggest issue was WebCite's abuse detection process, which would flag the robot after it had made a certain number of requests. To account for this and be generally nice users, we added logic to ensure a minimum delay between archival requests submitted to both the IA and WC. Exponential delay logic was implemented for WC when encountering general timeouts, other failures (like mysql error messages) or the abuse logic. Eventually, we learned that certain URLs would cause WC's crawler to timeout indefinitely, requiring the implementation of a maximum retry count (and a failure status) if the error wasn't caused by the abuse logic.

To estimate what impact we had on the archives' coverage of the study URLs, we compared a URL survey done directly prior to our submission process to one done afterwards; a period of about 3.5 months. It was assumed that the contribution due to unrelated processes would not be very large given that there was only a modest increase in coverage, 5% for IA and 1% for WC, over the previous period of just under a year and a half.

Each of the two archival engines had interesting behaviors which required gauging successful submission of a URL by whether it was archived as of a subsequent survey rather than using the statuses returned by the engines. For the Internet Archive, it was discovered that an error didn't always indicate failure, as there were 872 URLs for which wget returned an error but which were successfully archived. Conversely, WebCite returned an asynchronous status, such that even in the case of a successful return the URL might fail archival; the case in 955 out of a total of 7,285.

Submitting the 2662 URLs to IA took a little less than a day, whereas submitting 7285 to WC took over 2 months. This likely reflects IA's large server capacity, funding and platform maturity due to its age.

Generating the list of unique URLs

Converting some of the potential predictors from the list of published URLs to the list of unique URLs presented some unique issues. In particular, while converting those based on the URL itself (domain, depth, whether alive or in an archive) were straightforward, those which depended upon a publishing article (number of times URL was published, the number of times an article was cited, publishing journal, whether there was funding text) were estimated by collating the data from each publishing. Only a small amount, 8%, of the unique URLs, appeared more than once, and among the measured variables that pertained to the publishing there was not a large amount of variety. Amongst repeatedly-published URLs, 43% appeared in only one journal and the presence of funding text was the same 76% of the time. For calculating the number of times a paper was published, multiple appearances of a URL within a given title/abstract were counted as one. Thus, while efforts were made to provide a representative collated value where appropriate, it’s expected that different methods would not have produced significantly different results.

Pitfalls and drawbacks

Even though WOS’s index appears to have better quality OCR than Pub Med, it still has OCR artifacts. To compensate for this, the URL extraction script tried to use some heuristics to detect the most common sources of error and correct them. Some of the biggest sources of error were: randomly inserted spaces in URLs, “similar to” being substituted for the tilde character, periods being replaced with commas and extra punctuation being appended to the URL (sometimes due to the logic added to address the first issue).

Likely the largest contributors to false negatives are errors in OCR and the attempts to compensate for them. In assessing the effectiveness of our submissions to IA, it is possible that the estimate could be understated due to URLs that had been submitted but not yet made available within the Wayback Machine.

Dynamic websites with interactive content, if only present via an archiving engine, would be a source of false positives, as the person accessing the resource would presumably want to use it as opposed to viewing the design work of its landing page. If a published web site goes away and another installed in its place (especially true if a .com or .net domain is allowed to expire), then the program will not be able to tell the difference since it will see a valid (though impertinent) web site. In addition, though page contents can change and lose relevance from their original use[41], dates of archival were not compared to the publication date.

Another source of false positive error would be uncaught OCR artefacts that insert spaces within URLs if it truncated the path but left the correct host intact. The result would be a higher probability that the URL would appear as a higher level index page, which are generally more likely to function than pages at lower levels [9, 16].

Bibliographic database

Web of Science was chosen because, compared to Pub Med, it was more cross-sectional and had better OCR quality based on a small sampling. Many of the other evaluation criteria were similar between Pub Med and WOS, as both contain scholarly work and have an interface to download bibliographic data. Interestingly, due to the continued presence of OCR issues in newer articles, it appears that bibliographic information for some journals is not yet passed electronically.

Conclusions

Based on the data gathered in this and other studies, it is apparent that there is still a problem with irretrievable scholarly research on the Internet. We found that roughly 50% of URLs published 11 years prior to the survey (in 2000) are still left standing. Interesting is that the rate of decay for late-published URLs (within the past 11 years) appears to be higher than that for the older ones, lending credence to what Koehler suggested about eventual decay rate stabilization[32]. Survival rates for living URLs published between 1996 and 1999, inclusive, only vary by 2.4% (1.5% for unique) and have poor linear fits (R^2 of .51 and .18 for unique), whereas years [2000, 2010] have linear slope 0.031 and R^2 .90 (.036 and R^2 .95 for unique URLs using the first published year) indicating that the availability between years for older URLs is much more stable whereas the availability for more recent online resources follow a linear trend with a predictable loss rate. Overall, 84% of URLs (82% of the unique) were available in some manner: either via the web, IA or WC.

Several remedies are available to address different aspects of the link decay problem. For data-based sites that can be archived properly with an engine such as the Internet Archive or WebCite, one remedy prototyped is to submit the missing sites which are still alive to the archiving engines. Based on our results (illustrated in Figure 2.6), this method was wildly successful, increasing IA's coverage of the study's URLs by 22% and WC's by 255%. Journals could require authors to submit URLs to both the Internet Archive and WebCite, or alternatively programs similar to those employed in this study could be used to do it automatically. Another way to increase archival would be for the owners of published sites to ease restrictions for archiving engines since 507 (352 unique) of the published URLs had archiving disabled via robots.txt according to the Internet Archive. Amongst these, 16% (22% of the unique) have already ceased being valid. While some sites may have good reason for blocking automated archivers (such as dynamic content or licensing issues), there may be others that could remove their restrictions entirely or provide an exception for preservation engines.

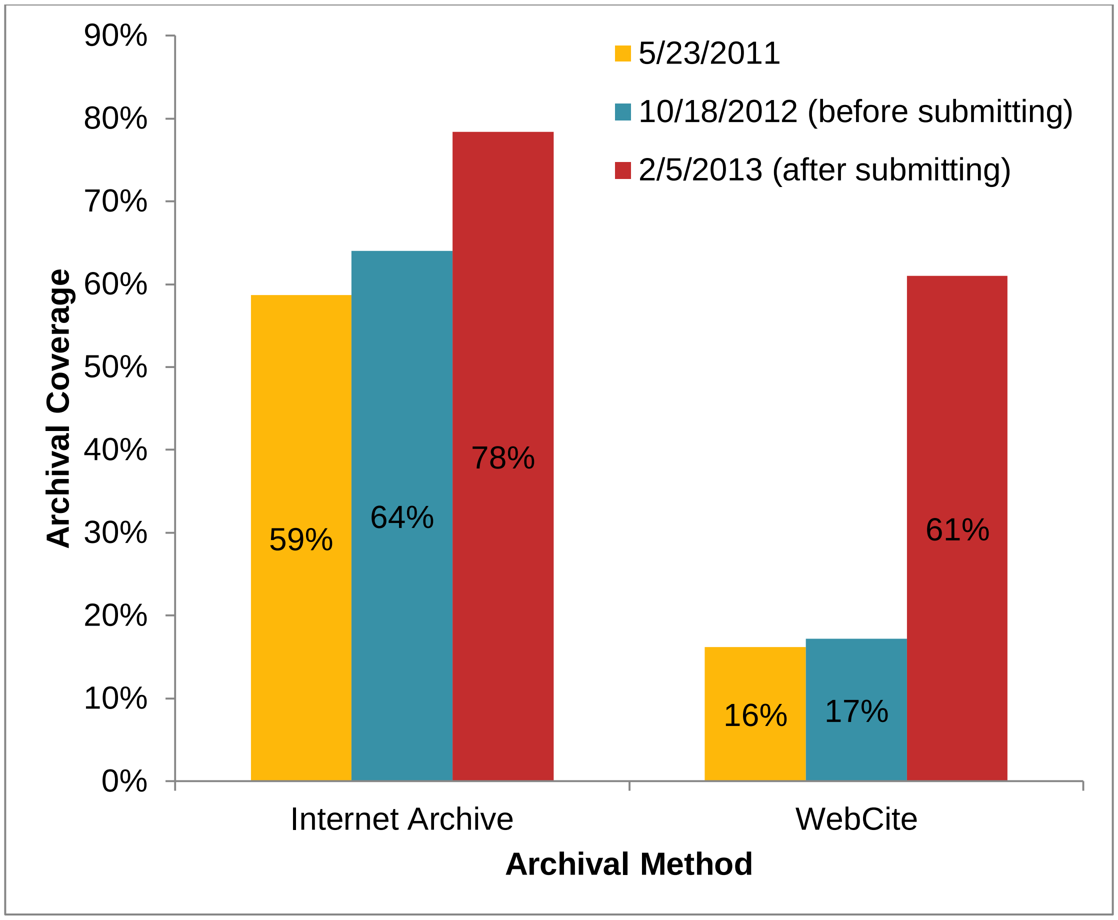


Figure 2.6 - Archival engine coverage of the URL list at different times. All URLs marked as alive in 2011 but missing from an archive were submitted between the 2012 and 2013 surveys. The effect of submitting the URLs is most evident in the WebCite case though the Internet Archive also showed substantial improvement. Implementing an automated process to do this could vastly improve the retention of scholarly static web pages.

To address the control issue for redirection solutions (DOI, PURL) mentioned in the introduction, those who administer cited tools could begin to maintain and publish a permanent URL on the web site itself. Perhaps an even more radical step would be for either these existing tools or some new tool to take a Wikipedia approach and allow end-users to update and search a database of permanent URLs. Considering the studies that have shown around at least 30% of dead URLs to be locatable using web search engines [4, 30], such a peer-maintained system could be effective and efficient, though spam could be an issue if not properly addressed.

For dynamic websites, the current solutions are more technically involved, potentially expensive and less feasible. These include mirroring (hosting a website on another server, possibly at another institution) and providing access to the source code, both of which require time and effort. Once the source is acquired, it can sometimes take considerable expertise to make use of it as there may be complex libraries or framework configuration, local assumptions hard-coded into the software or it could be written for a different platform (Graphics Processing Unit, Unix, Windows, etc.). The efforts to have reproducible research, where the underlying logic and data behind the results of a publication are made available to the greater community, have stated many of the same requirements as preserving dynamic websites [42, 43]. Innovation in this area could thus have multiple benefits beyond just the archival.

Methods

Data preparation

The then-current year (2011) was excluded to eliminate bias from certain journals being indexed sooner than others. For analysis and statistical modeling, the R program [44] and its "survival" library [45] were used (scripts included in supplement).

Wherever possible, statistics are presented in 2 forms: one representing the raw list of URLs extracted from abstracts and titles and the other representing a deduplicated set of those URLs. The former is most appropriate when thinking about what a researcher would encounter when trying to use a published URL in an article of interest and also serves as a way to give weight to multiply-published URLs. The latter is more appropriate when contemplating scholarly URLs as a whole or when using statistical models that assume independence between samples.

URLs not the goal of this study such as journal promotions and invalid URLs were excluded using computational methods as much as possible in order to minimize subjective bias. The first method, removing 943 (26 unique), looked for identical URLs which comprised a large percentage of a journal's published collection within a given year. Upon manual examination, a decision was then made whether to eliminate them. The second method, which identified 18 invalid URLs (all unique), consisted of checking for WebCitation's "UnexpectedXML" error. These URLs were corrupted to the point that they interfered with XML interpretation of the request due either to an error in our parsing or the OCR.

DOI sites were identified by virtue of containing "http://dx.doi.org". PURL sites were identified by virtue of containing "http://purl." in the URL. Interestingly, 3 PURL servers were identified through this mechanism: purl.oclc.org, purl.org and purl.access.gpo.gov.

To make for results more comparable to prior work as well as easier to interpret analysis, a URL was considered available if it successfully responded to at least 90% of the requests and unavailable if less than that. This method is similar to the method used by Wren[5], and differs from Ducut's[2] by not using a "variable availability" category defined as being available > 0% and < 90% of the time. Our results show that 466 unique URLs (3.2%) would have been in this middle category, a number quite similar to what Wren's and Ducut's would have been (3.4% and 3.2%, respectively). Being such a small percentage of the total, their treatment is not likely to affect analysis much regardless of how they are interpreted. Having binary data also eases interpretation of the statistical models. In addition, due to the low URL counts for 1994 (3) and 1995 (22), these years were excluded from analysis.

Survival model

Survival analysis was chosen to analyze living URLs due to its natural fit; like people, URLs have lifetimes and we are interested in discussing them, what causes them to be longer or shorter and by how much. Lifetimes were calculated by assuming URLs were alive each time they were published, which is a potential source of error [8]. Data was coded as either right or left-censored; right-censored since living URLs presumably would die at an unknown time in the future and left-censored because it was unknown when a non-responding URL had died. Ages were coded in months rather than years in order to increase accuracy and precision.

Parametric survival regression models were constructed using R's *survreg()*. In selecting the distribution to use, all of those available were tried, with the logistical showing the best overall fit based on Akaike Information Criterion score. Better fits for two of the numeric predictors (number of citations to a publishing paper and number of times a URL was published) were obtained by taking the base 2 logarithm. Collinearity was checked by calculating the variance inflation factor against a logistic regression fit to the web outcome variable. Overall lifetime estimates were made using the *survfit()* function from R's survival library.

Extracting & testing URLs

To prepare a list of URLs (and their associated data), a collection of bibliographic data was compiled by searching WOS for "http" in the title or abstract, downloading the results (500 at a time), then finally collating them into a single file. A custom program (extract_urls.py in Appendix) was then used to extract the URLs and associated metadata from these, after which 5 positive and 2 negative controls were added. A particular URL was only included once per paper.

With the extracted URLs in hand, another custom program (check_urls_web.py in Appendix) was used to test the availability of the URLs 3 times a day over the course of 30 days, starting April 16, 2011. These times were generated randomly by scheduler.py (included in Appendix), the algorithm guaranteeing that no consecutive runs were closer than 2 hours. A given URL was only visited once per run even if it was published multiple times, saving load on the server and speeding up the total runtime (which averaged about 25 minutes due to use of parallelism). Failure was viewed as anything that caused an exception in python’s “urllib2” package (which includes error statuses, like 404), with the exception reason being recorded for later analysis.

While investigating some of the failed fetches, a curious thing was noted: there were URLs that would consistently work with a web browser but not with the Python program or other command line downloaders like wget. After some investigation, it was realized that the web server was denying access to unrecognized User Agent strings. In response, the Python program adopted the User Agent of a regular browser and subsequently reduced the number of failed URLs.

At the end of the live web testing period, a custom program (check_urls_archived.py in Appendix) was used to programmatically query the archive engines on May 23, 2011. For the Internet Archive’s Wayback Machine, this was done using an HTTP HEAD request (which saves resources vs. GET) on the URL formed by “http://web.archive.org/web/*/” + <the url>. Status was judged by the resulting HTTP status code with 200 meaning success, 404 meaning not archived, 403 signifying a page blocked due to robots.txt and 503 meaning that the server was too busy. Because there were a number of these 503 codes, the script would make up to 4 attempts to access the URL, with increasing back off delays to keep from overloading IA’s servers. The end result still contained 18, which were counted as not archived for analysis. For WebCite, the documented API was used. This supports returning XML, a format very suitable to automated parsing [46]. For sites containing multiple statuses, any successful archiving was taken as a success.

References

2. Ducut E, Liu F, Fontelo P: **An update on Uniform Resource Locator (URL) decay in MEDLINE abstracts and measures for its mitigation.** *BMC Med Inform Decis Mak* 2008, **8:**-.
3. Dimitrova DV, Bugeja M: **Consider the source: Predictors of online citation permanence in communication journals.** *Portal-Libraries and the Academy* 2006, **6:**269-283.
4. Wren JD: **URL decay in MEDLINE - a 4-year follow-up study.** *Bioinformatics* 2008, **24:**1381-1385.
5. Wren JD: **404 not found: the stability and persistence of URLs published in MEDLINE.** *Bioinformatics* 2004, **20:**668-U208.
6. Yang SL, Qiu JP, Xiong ZY: **An empirical study on the utilization of web academic resources in humanities and social sciences based on web citations.** *Scientometrics* 2010, **84:**1-19.
7. Koehler W: **An analysis of Web page and Web site constancy and permanence.** *J Am Soc Inf Sci* 1999, **50:**162-180.
8. Aronsky D, Madani S, Carnevale RJ, Duda S, Feyder MT: **The prevalence and inaccessibility of Internet references in the biomedical literature at the time of publication.** *J Am Med Inform Assn* 2007, **14:**232-234.
9. Wren JD, Johnson KR, Crockett DM, Heilig LF, Schilling LM, Dellavalle RP: **Uniform resource locator decay in dermatology journals - Author attitudes and preservation practices.** *Arch Dermatol* 2006, **142:**1147-1152.
14. **The DOI System** [http://www.doi.org/]
15. **PURL Home Page** [http://purl.org]
16. Casserly MF, Bird JE: **Web citation availability: Analysis and implications for scholarship.** *College & Research Libraries* 2003, **64:**300-317.
17. **The Internet Archive** [http://www.archive.org/web/web.php]
18. Eysenbach G, Trudell M: **Going, going, still there: Using the WebCite service to permanently archive cited web pages.** *Journal of Medical Internet Research* 2005, **7:**2-6.
21. **Key Facts on Digital Object identifier System** [http://www.doi.org/factsheets/DOIKeyFacts.html]
22. **EZID: Pricing** [http://n2t.net/ezid/home/pricing]
23. Markwell J, Brooks DW: **"Link rot" limits the usefulness of web-based educational materials in biochemistry and molecular biology.** *Biochemistry and Molecular Biology Education* 2003, **31:**69-72.
24. Thorp AW, Brown L: **Accessibility of internet references in Annals of Emergency Medicine: Is it time to require archiving?** *Ann Emerg Med* 2007, **50:**188-192.
25. Carnevale RJ, Aronsky D: **The life and death of URLs in five biomedical informatics journals.** *International Journal of Medical Informatics* 2007, **76:**269-273.
26. Wagner C, Gebremichael MD, Taylor MK, Soltys MJ: **Disappearing act: decay of uniform resource locators in health care management journals.** *J Med Libr Assoc* 2009, **97:**122-130.

27. Duda JJ, Camp RJ: **Ecology in the information age: patterns of use and attrition rates of internet-based citations in ESA journals, 1997-2005.** *Frontiers in Ecology and the Environment* 2008, **6:**145-151.
28. Rhodes S: **Breaking Down Link Rot: The Chesapeake Project Legal Information Archive's Examination of URL Stability.** *Law Library Journal* 2010, **102:**581-597.
29. Goh DHL, Ng PK: **Link decay in leading information science journals.** *Journal of the American Society for Information Science and Technology* 2007, **58:**15-24.
30. Casserly MF, Bird JE: **Web citation availability - A follow-up study.** *Libr Resour Tech Ser* 2008, **52:**42-53.
31. Russell E, Kane J: **The missing link - Assessing the reliability of Internet citations in history journals.** *Technology and Culture* 2008, **49:**420-429.
32. Koehler W: **A longitudinal study of Web pages continued: a consideration of document persistence.** *Information Research-an International Electronic Journal* 2004, **9:**-.
33. Dellavalle RP, Hester EJ, Heilig LF, Drake AL, Kuntzman JW, Graber M, Schilling LM: **Information science - Going, going, gone: Lost Internet references.** *Science* 2003, **302:**787-788.
34. Evangelou E, Trikalinos TA, Ioannidis JPA: **Unavailability of online supplementary scientific information from articles published in major journals.** *Faseb Journal* 2005, **19:**1943-1944.
35. Sellitto C: **The impact of impermanent web-located citations: A study of 123 scholarly conference publications.** *Journal of the American Society for Information Science and Technology* 2005, **56:**695-703.
36. Bar-Ilan J, Peritz B: **The lifespan of "informetrics" on the Web: An eight year study (1998-2006).** *Scientometrics* 2009, **79:**7-25.
37. Gomes D, Silva MJ: **Modelling Information Persistence on the Web.** In *Book Modelling Information Persistence on the Web* (Editor ed.^eds.). City; 2006.
38. Markwell J, Brooks DW: **Evaluating web-based information: Access and accuracy.** *Journal of Chemical Education* 2008, **85:**458-459.
39. Wu ZQ: **An empirical study of the accessibility of web references in two Chinese academic journals.** *Scientometrics* 2009, **78:**481-503.
40. Klein JP, Moeschberger ML: *Survival analysis : techniques for censored and truncated data.* 2nd edn. New York: Springer; 2003.
41. Bar-Ilan J, Peritz BC: **Evolution, continuity, and disappearance of documents on a specific topic on the web: A longitudinal study of "informetrics".** *Journal of the American Society for Information Science and Technology* 2004, **55:**980-990.
42. Peng RD: **Reproducible research and Biostatistics.** *Biostatistics* 2009, **10:**405-408.
43. Ince DC, Hatton L, Graham-Cumming J: **The case for open computer programs.** *Nature* 2012, **482:**485-488.
44. R Development Core Team: **R: A Language and Environment for Statistical Computing.** In *Book R: A Language and Environment for Statistical Computing* (Editor ed.^eds.). City: R Foundation for Statistical Computing; 2011.

45. Therneau T: **A Package for Survival Analysis in S.** In *Book A Package for Survival Analysis in S* (Editor ed.^eds.), 2.36-12 edition. City; 2012.

46. **WebCite Technical Background and Best Practices Guide** [http://www.webcitation.org/doc/WebCiteBestPracticesGuide.pdf]

CHAPTER 3 - LOGIC CAPSULE

Using virtual machines to enhance digital scientific resources

Abstract

Background

Internet-based scientific resources have had a huge impact on science and are widely used to facilitate the sharing of information and tools. Unfortunately, they vanish with regularity, having a median lifetime of about 9 years across the sciences, in a phenomenon known as link decay or link rot. This vanishing can limit the ability of future researchers to reproduce the original work. Solutions have been proposed for resources based on simple webpages, however these solutions cannot in many cases properly archive those with complex server logic which compose about 45-62% of those which are missing. With the rate of publication for these online resources continuing to grow, so too will the number that vanish.

Additionally, concerns about the difficulty to replicate complex, computer-based statistical analyses have spawned a growing movement around the need for reproducible research. To answer these needs, technology that addresses the long term availability, ease of use and compatibility requirements of these interactive computing environments is required.

Results

The needs for addressing the problems of archiving and utilizing complex interactive websites and software are examined and one possible solution involving virtualization technology is discussed. This solution combines virtualization with best practices to address archival and other issues that inhibit the use of scientific digital resources. Finally, a prototype that implements this solution is put forward and made available at http://logiccapsule.net.

Conclusions

Virtualization coupled with standardizing practices provides a practical technology ideal for archiving complex scientific applications and their data, allowing scientists to precisely reproduce the complex analyses of others. It has the potential to not only improve the quality and availability of science across several fields but can also improve its productivity.

Logic Capsule provides an environment for the long-term archiving of scholarly applications that responds to the archival, reproducible research, format obsolescence, and ease of use concerns expressed by the scientific community. By making research more accessible, if adopted Logic Capsule or something similar could not only improve the quality and availability of science across several fields but also improve its productivity by making that science easier to use and replicate.

Introduction

Reproducing and scrutinizing the research of others is a time-honored tradition that forms a basic pillar of science. So too is the passing along of the knowledge, methods and sources relied upon to gain that knowledge, allowing many generations to see further by "standing on the shoulders of giants". Until recently, the primary media for the dissemination and review of this information have been physical books and journals. Though it has taken time, society has learned and refined techniques to archive and make this paper media-based knowledge available to future generations.

Over the last 20 years, the Internet has arisen as the new primary medium for scholarly communication. Scholarly Internet-based resources play an increasingly important role in modern research, demonstrated by the increasing number of Uniform Resource Locators (URLs) published in titles and abstracts [2, 47]. These resources disappear steadily, however, affecting a wide variety of subject areas [2, 4, 47]; a phenomenon referred to as link decay or link rot. Recognizing this problem, solutions for archiving static web pages arose including the Internet Archive[17] and WebCite[18]. These methods can help, but only when the resources are static, having the ability to be represented as if printed out.

Not all resources are static, however, which can prevent these archives from creating an accurate reproduction. Some URLs point to interactive web applications that rely upon comprehensive server-side logic and data. Others have programs, code or documents that must be processed on the downloader's system. [4] estimated the prevalence of non-static resources (classified as software programs or databases) among missing biomedical URLs in MEDLINE to be 62%. [9] estimated the prevalence to be 45% within Dermatology journals. Thus, given the fairly consistent disappearance of Internet-based resources in general, it is clear that a significant number of missing scholarly URLs are not fully archivable by current solutions focused on static pages.

Even when a program is available, using it to reproduce original research could be hampered due to age and complexity. Format obsolescence, the lack of ability to use a particular file due to not being able to interpret its encoding, can render a resource unusable even though it can be accessed. This is especially pertinent to files created more than a decade or two in the past. For example, what is the likelihood that in 20 years a program written in version 2 of the Python programming language will still be executable when development efforts for the language have already been on version 3 for several years? In 1995, Rothenberg brought attention to this potential lack of ability to read media, both physically and logically[48]. 15 years later, however, Rosenthal argued that obsolescence is not as pressing an issue due to a combination of more mature market dynamics, open source as well as the ability to use virtualization in a manner similar to what is presented in this paper[49]. In addition, the software configuration complexity of many modern analytical tools can, for many scientists, be a hindrance to their reproduction or even use, and are beyond the expertise of many[50]. Even for those with the technical skills, it can be a time-consuming chore to configure the prerequisite packages. A single application could conceivably require configuring several components, for example a web server, database and statistical software.

Other concerns that involve reproducing the work of others center on being able to translate an article’s description of its statistical analysis. **Reproducible research** in this context encompasses being able to reproduce and scrutinize such an analysis. This can be especially important when the underlying data is difficult to reproduce due to time, expense, or other factors[20] and can many times be difficult to do solely using the steps described in an article[10]. Wicherts relates reproducibility to aviation: a co-pilot can check every action of the captain and there is a black box that records each action[12].

Addressing these concerns, virtualization technology packages the data and logic needed to capture a complete computing environment into a single, executable and portable package that meets needs of reproducible research ideally[51]. This package is referred to as a Virtual Machine (VM), and may be run on any system possessing an emulator or hypervisor supporting x86-based VMs, which includes most desktop and server systems. That computation would then be available to future scholars for understanding, reproduction and criticism.

Presented in the next section is a discussion of the requirements for an archival solution with a focus on reproducible research, their challenges and how a virtualized system ideally meets them. Next, a prototype implementing the ideas from the prior two sections called Logic Capsule is presented. After that, areas for future work are discussed, related work and finally conclusions.

Requirements

The ultimate goal for any archival system supporting reproducible research as a primary function is to facilitate a future researcher making use of someone else's work. To that end, a few aspects of such a system make themselves clear in that the package and any dependencies must be: findable by researchers looking for them, available when needed; easily executable in the environments future users might employ and able to faithfully reproduce the original service or analysis. A bonus would be that it's easy to use, capturing the expertise of the author as much as possible to avoid needlessly complex configuration.

There are primarily three classes of threats to the success of this type of project: natural, political and technological. Natural threats result from catastrophic events which can destroy either the data or the ability to access that data and include earthquake, flood, hail and tornado. Political threats emerge from man-made organizations and include funding, copyright, legal concerns and an institution’s will to continue the effort. Finally, technological threats represent those arising from technical implementations like having a VM in a format that is no longer executable. Other threats include archive corruption (whether accidental or purposeful) and physical storage failures.

Directory

A directory will likely be one of the most important aspects of a VM-based research archival service as it connects researchers with the resources they are searching for. To do its job faithfully a directory must possess two key properties. First, it must facilitate making its contents easy to find through a local or 3rd party search engine. Second, entries in this directory must both be unique and immutable.

Resources that can’t be found are not useful. Thus, a directory for a reproducible research VM-based service must make its contents easy to locate, both by users looking for resources using Internet search engines or users who specifically came to the directory to find the resource. To facilitate locating entries in the directory from external search engines, all entries should 1) be enumerated using methods like sitemaps[52] to ease their indexing and 2) make their metadata available in a programmatic fashion using markup such as RDF[53] so that third parties can integrate the directory. Required metadata would include the resource's title and any associated URLs along with relevant citation information such as a publishing article's title and author list. Using this information, one can easily foresee sites such as journal websites and citation indexes like Web of Science and Pubmed integrating these VMs into their interfaces. It's also possible and hopeful that searches based on an article's title would return the directory entry within the first set of results. Browsing the directory locally would greatly be facilitated by including keywords or tags, so that categories could automatically be generated. Other information, such as the cryptographic checksums (discussed in the below section, "Available"), format and usage instructions, are helpful as well, but more so after the VM has been located.

Once a directory entry for a particular resource is made available, it's important that prior versions are immutable (e.g. never removed or altered). A large part of reproducible research is being able to produce exactly the same analysis as an original author. Thus, it would stand to reason that once a VM has been introduced into the system and cited, it should remain that way for future researchers. Though this might sound wasteful, examples of its utility could include understanding why different results were seen in a later version of a particular software package or how an earlier dataset differed from a later, more mature one. In a similar vein, each entry should carry a unique identifier so that particular versions and packages can be identified, shared, downloaded and discussed.

Self-Contained

The key to reproducible research is reproducing research. For an analysis component, this would mean that all of the logic and data a part of the original analysis should be included in a self-contained VM. A corollary to this requirement is that the VM should not depend on network resources. This is not always feasible, however.

For most Virtual Machines, it would be expected that the logic portion (the Operating System (OS), analysis software and any dependencies) would take up the majority of space. There are cases, though, where the size of the dataset could easily eclipse the logic, causing very large VMs which would be difficult to transfer. These could include those containing Next Generation Sequencing data as well as other “big data” projects with large files. For example, a collection of Ensembl Annotated Human Genome data (usable in several bioinformatics contexts) hosted on Amazon.com is 310GB[54]. Such large file sizes, especially if those files were shared across multiple projects, might inhibit the use of the VMs as the downloaders would have to wait for the transfers complete and have sufficient disk space available to run them.

One solution to this problem would be to allow VMs to use the network to access just the portions needed for a particular computation. It may have to be enough to acknowledge that someone wishing to download and reproduce a computation with a large amount of data will have to enable access to Internet resources. As long those resources remain available, then reproducibility will have been preserved. VMs could also foreseeably make use of valuable online service such as TogoWS[55] as part of its workflow.

Since interfaces and file locations can change, it may make sense for a VM preservation service to employ a policy by which external dependencies would only be referenced using a method that has long term access and preservation in mind. For example, such a requirement could be that references must only be done through Digital Object Identifiers[21], or to consortiums which endeavor to maintain long-term access and backwards compatibility. The DataCite consortium provides exactly this type of unique, permanent identifier for research datasets[56].

Available

For a tool to be useful, it must be accessible. This implies that it must have been stored in such a way that it continues to be retrievable, that it is reachable over a network and that the package's integrity has been maintained. To avoid natural and political risks to the storage of VMs, they must be backed up and stored redundantly in geographically and politically diverse locations. There are a number of ways to satisfy this requirement. One is to have the VM storage redundantly shared and served from multiple organizations. By distributing the work among multiple systems and groups of people, the impact of a failure, whether caused by a system crash, network trouble, physical disaster or organizational apathy is mitigated by having other sites that can mirror and serve the contents. Another way is to use cloud-based object storage services such as Amazon's S3 and Google's Cloud Storage. Cloud services many times have geographical redundancy built in while incurring less operational effort from the implementer, less up-front investment in hardware and in many cases an already-built Content Delivery Network. These benefits come at a cost, however, in that if the cloud is completely relied upon for redundancy and durability it can itself become a single point of failure unless multiple providers are used at even greater recurring expense. To mitigate cloud concerns, a hybrid model is possible, with an organization hosting the contents both individually and using public cloud services to spread the load. This would have the advantage of simplifying the manually-managed infrastructure (perhaps only hosting at a single site) while gaining the redundancy and bandwidth advantages of a cloud provider while not being completely reliant on that provider.

Given the long-term preservation requirement for this system, steps must be taken in order to facilitate recovery of the data in the face a byzantine failure. Ideally, the VMs and their metadata should be backed up to multiple geographies (for protection from natural threats) and using multiple types of media such as optical disks, hard disks and tape to protect against bit rot and format obsolescence. Cloud backup could be one of the methods used for backing up, though shouldn't be the only one. Something as simple as an expired credit card could then end up jeopardizing the contents of the entire system when they were most needed!

Properties of a collision resistant cryptographic hash function, h()

Weak collision resistance

$Given\ input\ x, it\ is\ difficult\ to\ find\ a\ distinct\ x'such\ that\ h(x) = h(x')$

Strong collision resistance

$such\ that\ h(x) = h(x')$

Figure 3.1 – Properties of a collision resistant cryptographic hash function. "Difficult" in this context may be substituted with "computationally infeasible". Such functions can be used by a VM archiving service to make it extremely difficult for a modified VM to go undetected. Source: [1]

While being available is an important aspect of a VM service, steps need to be taken to ensure to ensure that what is found is what was uploaded. To that end, the output of at least one collision resistant hash function (used here interchangeably with "hash function") should be included in a directory entry. A hash function takes an arbitrarily sized file and quickly computes a short, fixed-size (typically 256-512 bits) summary that has some advantageous properties, which include 1) weak collision resistance and 2) strong collision resistance, for which definitions may be found in Figure 3.1 and [1]. These properties, if present in a hash function, ensure that it would be very difficult a stored VM archive to change without being detected, whether the change were intentional or not. Candidate functions would include any of the several which have been studied and are generally accepted by the cryptographic community, such as Whirlpool and those in the SHA-2 and SHA-3 families. Initial calculation of the hash(es) should be done as early as possible, preferably before the VM has left its author's computer, in order to detect corruption of the archive while it is in transit. A hash should also be calculated and checked periodically for each VM on the server as well as after downloading in order to ensure that it has remained unchanged since its generation. Multiple hashes using different functions could also be taken to avoid a breach in security of any one particular one such as has been seen with the broken MD5 function[57]. For improved security, public-key cryptography could be used to sign and later authenticate these hashes or even the directory entry itself.

Storage / Transfer Optimization

Even with a fast connection, downloading a VM can take a long time given that VM sizes of 20 or 30 GB are increasingly common. Long transfer times and excessive storage requirements, if not addressed, could cause user dissatisfaction and thus present an impediment to adoption. Several optimizations are possible. One is deduplication, which has shown significant promise in reducing VM sizes by 40%-80%[58, 59]. With it, redundancies within and across VMs can be detected and removed, such that only one copy of particular blocks need be stored or transferred. Deduplication takes advantage of the fact that much of the standard OS (such as a Linux distribution) is the same across VMs, and thus the largest amount of unique data comes from newly introduced functionality. Similarly, delta disks[60], otherwise known as difference disks or linked clones, would take this a step further if multiple VMs were dependent on the same popular base disk image (like the Ubuntu Cloud Image[61]). In those cases, as long as the same base image had previously been transferred or stored, only the changed parts needed to create a particular implementation would be needed. These technologies allow the transfer and storage price to be paid only once. Common direct compression programs such as zip, gzip, bzip2 and 7-zip, as well as the less common rzip[62], lrzip[63] and zpaq[64] can achieve sizable space savings for single VMs[65]. Such techniques have the potential to not only reduce storage space and costs, but also decrease the transfer times of inter-site replication and end-user downloads.

In addition to these optimizations, there are others that could be made to reduce the latency to when a VM is usable. The ability to lazily load portions of the virtual disk over the network as necessary would allow a VM to start running without being fully downloaded. This technique has been seen in Moka5’s cache priming[66], VMTorrent[67] and Snowflock[68]. In addition, a "Run in the Cloud" option, where the VM would automatically be deployed to some cloud, would gain the advantages of cloud deployment without being dependent on it. A hosted cloud solution such as SHARE[69] has the potential to offer instant access to the VM by streaming only the user interface. Peer to peer technology such as BitTorrent could also help in keeping storage and transfer costs low while increasing transfer speeds as demand rose.

Format

Given that the primary goal for this set of requirements is the long-term archival and ability to use research, documentation on a few aspects the VM itself will need to be written, tested and standardized upon. Supported file formats for the VM will need to be evaluated, with a goal of ensuring the ability to execute it potentially in the distant future. Desirable also is compatibility with a number of hypervisors. Some of the currently popular hypervisors and their native formats consist of: VMware’s VMDK, Virtualbox’s VDI, Microsoft’s VHD and KVM’s QCOW2. Each of the hypervisors mentioned support importing VMDK-formatted VMs, making VMDK a good candidate as the standard. VMDK and a large number of other formats are also supported by the qemu-img program, which could be used to translate between formats. This program could also perform server-side conversion as needed. Separate author's instructions would need to be written and tested with each popular hypervisor. In addition, compatibility between the hypervisors would need to be tested and documented periodically beyond just the format. For example, the virtual devices (such as network cards, processors, chipsets, etc) emulated by the hypervisors can differ and potentially be a cause of incompatibility.

Similar to aspects of a VM's format, the x86 instruction set itself could be a source of long term incompatibility. Current market trends, heavily driven by mobile computing and the use of open source systems that are processor agnostic, are reducing the x86's hegemony, so it is valid to ask whether x86 VMs will be runnable in 20 or 30 years. Two technologies will likely ensure that they are. The first, processor emulation, has been around for quite a long time and it is reasonable to believe that it will continue to be. Popular open source emulators such as qemu and Bochs have existed for some time. Indeed, some can currently run MS-DOS versions from close to 20 years ago[70, 71]. Native processors and the emulation technology will likely continue to become faster, meaning that even emulated performance should not be an issue. The second trend, cloud computing, should likewise become more widely standardized and mature. This means that even if local devices are not able to natively make use of these archived scholarly works, users should still be able to use them in a cloud environment using either virtualization or emulation.

Copyright

Concerns surrounding copyright and Intellectual Property (IP) violations are important and could be detrimental to the goals of an archival system if not addressed. They primarily fall into to two areas: 1) the uploading of software or other items to which the contributor is not authorized due to copyright, patent or trademark rights and 2) the release of the aforementioned rights belonging to the uploader. An archival system is in a similar situation to any Internet-based, user-generated content hosting service such as Drop Box, Youtube, or Amazon. Similar to those services, an archival system would employ measures to protect IP, though legal responsibility would rest with the uploader. The system may also need to conform to the Digital Millennium Copyright Act's (DMCA) requirements to address copyright complaints.

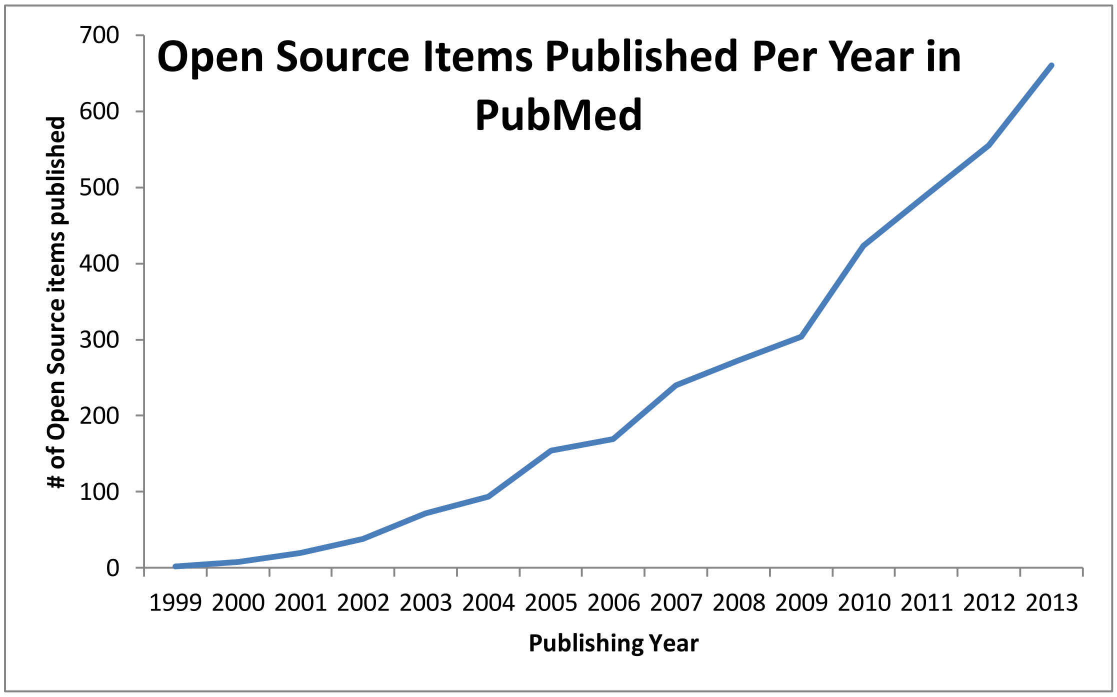


Figure 3.2 - Open Source use in PubMed. The adoption of open source in the biomedical field, quantified by having "open source" appearing in the title or abstract, has been steadily growing for over a decade. This indicates that a system only supporting open source software is only growing in relevance. Source: PubMed.gov

Unauthorized uploads could be discouraged using documentation, legal and review based mechanisms. Documentation on the site would discuss IP concerns and encourage the use of open source-licensed software stacks. These software stacks are very popular in the Bioinformatics area and are employed widely. They include Linux and its many distributions; statistical environments like R (with Bioconductor), Weka and Mr. Bayes; many task-specific packages like BLAST and EMBOSS and support for almost every popular language like C, C++, Fortran, Java, python, perl and PHP.

Accepting contributions requires that the user agree that to the best of their knowledge they have the legal right to distribute all of the uploaded materials. For those portions to which they own copyright (such as statistical analysis scripts), they would be required to license to the archival system and any future partners an irrevocable , unrestricted and perpetual license to redistribute their contributions. This may include selecting from a popular, standard open source license such as the BSD, GPL or Creative Commons. This would place the archival in the position of accepting in good-faith the attestation of the user that redistribution of the VM does not violate anyone's IP rights.

Archival systems could also implement a cursory review to see if there are any obvious violations before enabling redistribution of the VM. Such items could include using obviously copyrighted software such as the Windows OS or MATLAB. In cases where the previous mitigations failed, the archive's website would publish a contact address and work with its legal team to draft a policy permitting copyright holders to register complaints, similar to what is done at many other Internet-based hosting services[5].

Cloud computing-based remote execution could also address certain copyright concerns, as the running of VMs would take place on a centrally managed data center rather than on a user's computer. "Run in the Cloud" functionality could help broaden the list of acceptable contributions. For example, if two institutions (the one hosting a VM and the end user's) both held licenses to run a particular piece of copyrighted software, the user might be able to instantly run that VM on a remote node in addition to downloading it. There could also be cases where running the VM remotely were allowed, however download was forbidden if the end user's institution did not have the appropriate license. This is an exciting area to explore. SHARE employs some of these ideas[69].

[5] For examples, see https://www.dropbox.com/dmca and https://www.youtube.com/yt/copyright/copyright-complaint.html

While copyright issues constrain the flexibility of downloadable VM-based solutions, it is an addressable legal restriction and not a technical one. While it may seem that only supporting open source software might be an impediment, academic adoption within the biomedical field has been consistently growing (see Figure 3.2).

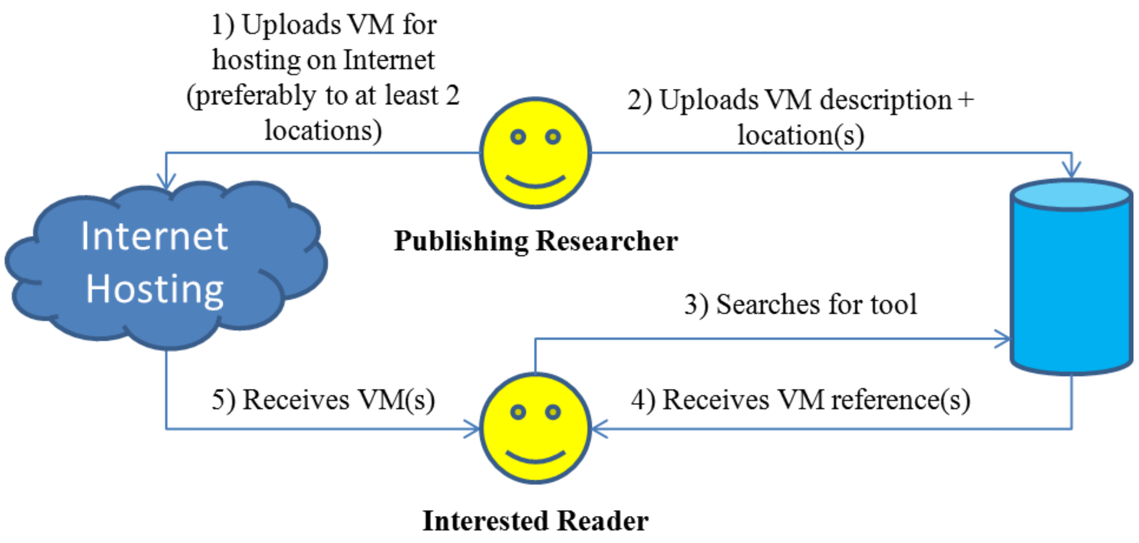
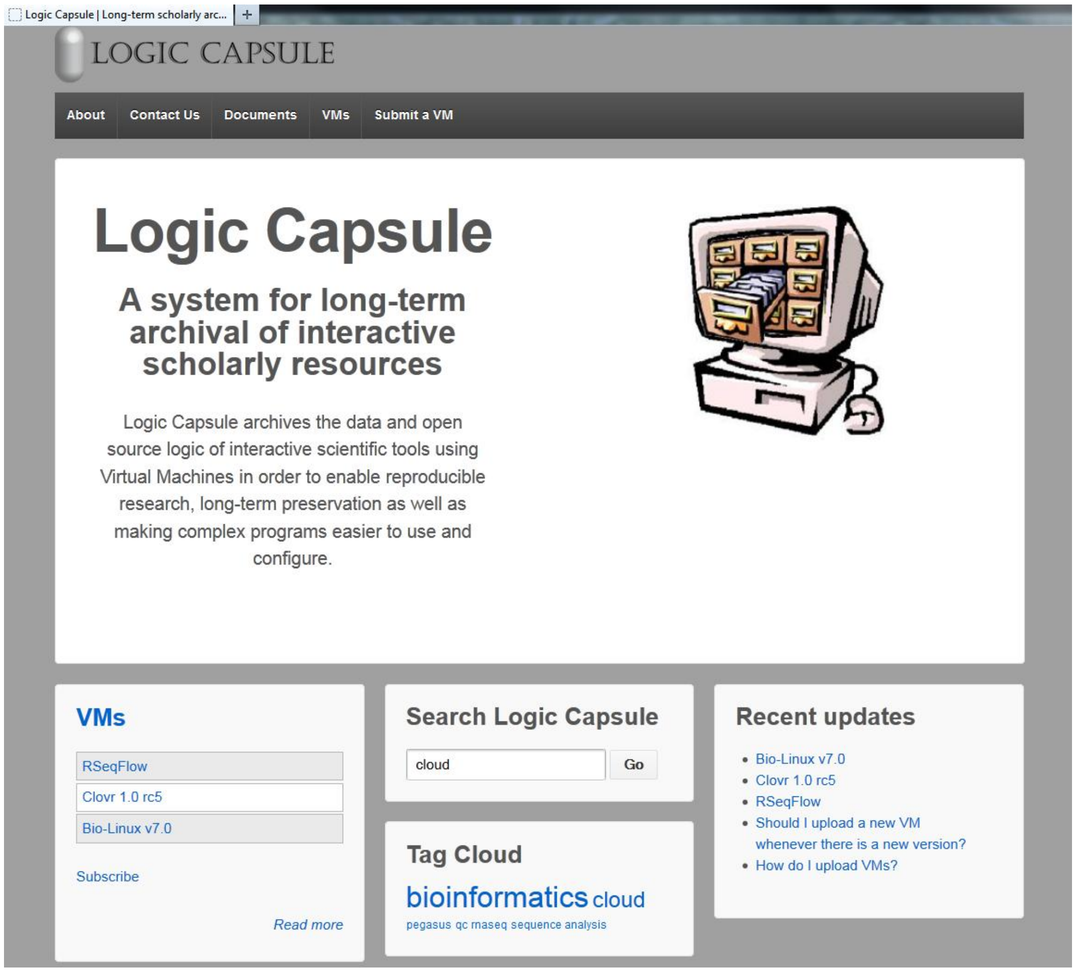

1) Uploads VM for hosting on Internet (preferably to at least 2 locations)
2) Uploads VM description + location(s)
Internet Hosting
Publishing Researcher
3) Searches for tool
5) Receives VM(s)
4) Receives VM reference(s)
Interested Reader

Figure 3.3 - Overview of Logic Capsule workflow. A publishing researcher 1) makes their VM available for download on the Internet, then 2) notifies Logic Capsule about their upload. At some later point, an interested party 3) searches for the functionality either via a general web search or Logic Capsule directly, 4) finds the VM's page on Logic Capsule and 5) downloads it.

Logic Capsule

Logic Capsule is a prototype for a virtual machine-based archival system for reproducible research. It catalogs and makes available scholarly VMs, “capsules”, with the workflow illustrated in Figure 3.3. It endeavors to meet the requirements outlined above by implementing a directory, metadata, tags, checksums, documentation and manual intervention at the key point of VM submission. The Wordpress Content Management System (CMS) system and the Responsive theme were used to facilitate access to and the organization of pertinent data. Directory entries can be browsed manually by selecting tags, searching or browsing all VMs manually. Commenting on VMs is supported and encouraged in order to share experiences gained while using capsules.

In its current implementation, after packaging a VM, a publishing researcher then uploads it to the Internet to be hosted. For this purpose, the user can use any hosting available to them including free cyberlocker services (see Table 3.1). Once uploaded, the researcher can then go to the Logic Capsule website and submit a new entry using the "Submit a VM" link from the homepage (shown in Figure 3.4). Once reviewed by an administrator, the VM will then be publicly available through the website, whether by navigation, searching on the site itself or searches via the Internet. After finding a VM of interest, a user then uses the link(s) to download the VM(s).


Logic Capsule | Long-term scholarly arc...

LOGIC CAPSULE

About | Contact Us | Documents | VMs | Submit a VM

Logic Capsule

A system for long-term archival of interactive scholarly resources

Logic Capsule archives the data and open source logic of interactive scientific tools using Virtual Machines in order to enable reproducible research, long-term preservation as well as making complex programs easier to use and configure.

VMs

RSeqFlow

Clovr 1.0 rc5

Bio-Linux v7.0

Subscribe

Read more

Search Logic Capsule

cloud Go

Tag Cloud

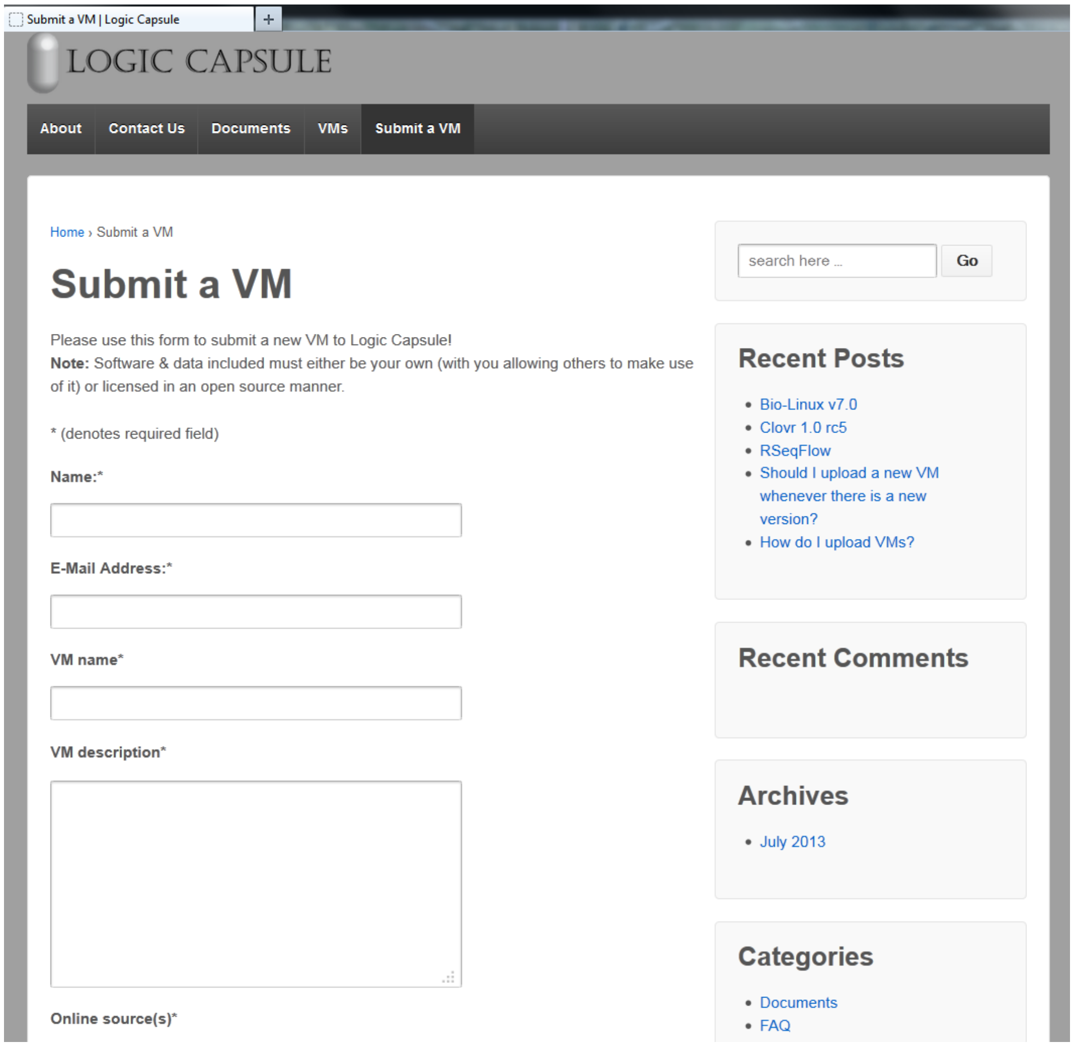
bioinformatics cloud pegasus qc maseq sequence analysis

Recent updates

- Bio-Linux v7.0
- Clovr 1.0 rc5
- RSeqFlow
- Should I upload a new VM whenever there is a new version?
- How do I upload VMs?

Figure 3.4 - The Logic Capsule home page The site aims to be simple and easy to use, presenting the most common functions such as searching, submitting or accessing a VM right away while also providing more sophisticated capabilities such as the ability to subscribe to new VM notifications.

Logic Capsule allows users to submit new VMs by filling out a form that is then reviewed before being included in the database. This form, shown in Figure 3.5, requires certain information needed for all VMs, including: title, description, release date (or some approximation), version, URL for site, contact for the VM, Citation (if applicable), instructions on how to use the VM, suggested tags, file size, SHA256 of the archive in order to verify the capsule’s integrity and links to download source(s) (preferably at least 2). Once an entry has been posted it will not generally be updated except to fix outdated links. New versions and updates will require new posts. A CAPTCHA is used to prevent automated bots from submitting entries. Currently, it is the uploader's responsibility to host the file(s) associated with this VM. Hosted, redundant storage would be a valuable feature that would have required additional funding.



Home › Submit a VM

Submit a VM

Please use this form to submit a new VM to Logic Capsule!
Note: Software & data included must either be your own (with you allowing others to make use of it) or licensed in an open source manner.

* (denotes required field)

Name:*

E-Mail Address:*

VM name*

VM description*

Online source(s)*

Recent Posts
- Bio-Linux v7.0
- Clovr 1.0 rc5
- RSeqFlow
- Should I upload a new VM whenever there is a new version?
- How do I upload VMs?

Recent Comments

Archives
- July 2013

Categories
- Documents
- FAQ

Figure 3.5 - The submission form for a new VM. Several fields are required; to prevent abuse, a CAPTCHA as well as manual intervention is required before the submission is made public.

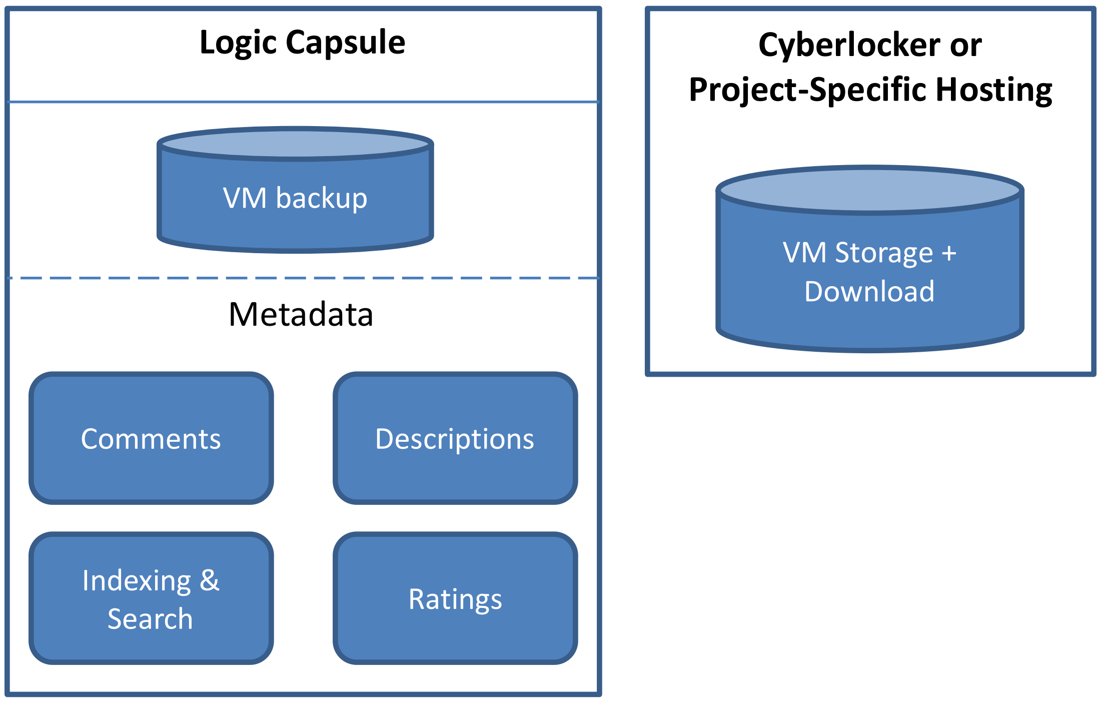


Figure 3.6 - Data-centric architecture of Logic Capsule. Various metadata components which make the information available and add value to it are hosted by Logic Capsule. Storage for VMs is currently provided by third parties.

The catalog makes VMs available through the homepage by either a comprehensive search function or direct browsing. Comments on VMs provide a social venue for users to help each other by providing feedback and tips. Similarly, tags may be applied to VMs to assist future seekers identify relevant software Documentation in the form of dedicated pages and FAQs are available from the home page, with new entries being added as feedback is received.

Table 3.1 – Selected cyber locker services. These services allow the uploading of arbitrary files for free, with most retaining them as long as they are accessed regularly. If other permanent, web-accessible storage is unavailable, these can be used by researchers to host VMs on Logic Capsule.

Site	Max file size
box.net	1GB
depositfiles.com	10GB
filedropper.com	5GB
filehosting.org	unlimited
hotfile.com	400MB
mightyupload.com	4GB
putlocker.com	1GB
rapidshare.com	unlimited
slingfile.com	2GB

Methods

WordPress, a popular Content Management System, is used to host the Logic Capsule website. It was chosen because of its ability to organize and present large amounts of information while at the same time facilitating social involvement among users. The Responsive theme was used in order to seamlessly accommodate different form factors, making the website easy to use from mobile devices, tablets and desktop computers. Metadata, as outlined in the data-centric architecture diagram in Figure 3.6, is kept in a mysql database. Figure 3.7 contains an execution-centric architecture diagram of Logic Capsule.

Since storage was not integrated due to cost and bandwidth concerns, capsules are downloaded to offline storage for long term archiving before publishing a submitted VM. This adds redundancy in case a hosted VM later becomes unavailable.

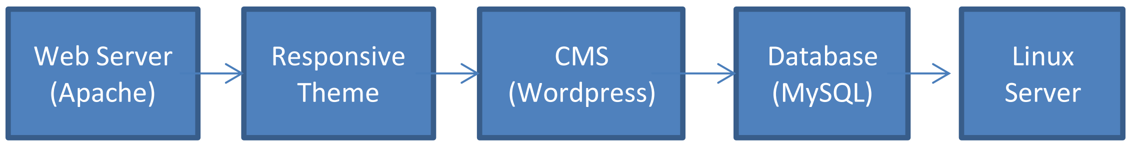


Figure 3.7 – Execution-centric architecture for Logic Capsule. Users utilize their web browser to interact with Wordpress via the Responsive Theme. Wordpress uses the MySQL database to store its information. All of this functionality is implemented on top of a linux-based server.

Future Work

Hosting of VMs in Logic Capsule is currently the publishing researcher's responsibility, though with additional funding this could be hosted centrally. The key to doing so would be ensuring enough bandwidth, replication and space to continue smooth operations. Maintaining consistent access to the hosted VMs would require not only technical measures but also institutional partnerships to prevent a single institution’s failure from denying access to the entire system.

Related work

Several existing projects produce ad-hoc VMs like CloVR [72], Bio-Linux [73] and RSeqFlow[74]. While providing these makes the software easy to use and ready to go (not requiring configuration), their usefulness for archiving is hindered by only have the latest version available; this is likely due to the large file size. In addition, the packages’ continued availability depends on the producing web site. From a reproducible research perspective, while these VMs provide a configured environment ready for analysis, the exact way a particular study used of the VM would not be available; detailed methods sections would bring us back to the current dilemma of relying upon imprecise natural language. These VMs are also not centrally catalogued, making it more difficult to find them.

SHARE[69] is a cloud-based VM solution for packaging reproducible research, and in many ways approaches the ideals of the system introduced in this paper. One distinct advantage of this method is that it can allow the use of copyrighted software due to institutional licenses; that software not being made available for download but being executed within the confines of an institution. Authors wanting to make available preliminary versions of their software might also appreciate the lack of the download ability. Running VMs in a remote datacenter also has benefits, such as permitting the use of relatively low powered clients (including thin clients, tablets and smartphones), better accommodating low-bandwidth connections (which would make downloading difficult) and making available vast computational resources that may not be otherwise easily available. Unfortunately, this comes with the drawback of many cloud-based solutions: reliance on a centralized operator. In the case of a data-center or network outage, instead of only preventing new VM downloads everyone would be prevented from making use of all VMs hosted in that environment. Lacking the ability to make copies could also reduce the chance of successful long-term archival.

Also focusing on cloud as a solution, [75] introduced a process to reproduce computations that involve public cloud computing based resources. It functions by having a programmer write logic that can rebuild the environment that was used for a particular experiment or analysis, separating the logic used for building the base VMs from that used to install and configure the application software and data that reside on top of those VMs. In some ways, [75] is complimentary to the solution presented in this paper as it represents a method for creating and running VM-based computations (continuing to keep the bulk of the configuration expertise needed with the creator as opposed to the end user) while this paper deals more in making those packages available and findable. It does not on its own address some of the other requirements for having a reproducible research system, such as ensuring that all of the software prerequisites continue to be available as well as ensuring the continued availability of the package itself.

Non-virtualization-based solutions also exist, though so far they have tended to been applicable to very specific scenarios or have restrictive environments. Galaxy provides a workbench for genomics studies that allows for direct linking to results that display each step taken to generate them[76]. Sweave allows code using the popular R system to be embedded into papers written using LaTex[77].

Conclusions

A VM-based archive used for reproducible research is practical, possible and provides an ideal environment for the long-term archiving for appropriate scholarly works. It addresses the archival, reproducible research, format obsolescence, and ease of use concerns that have been expressed by the scientific community. By making research more accessible, such a service stands not only to improve the quality of science across many fields but also improves its productivity. At the same time, there are research questions that need to be addressed in order to make such a service more comprehensive. These include referencing large data sets and reducing the overhead for the storage and transfer VMs, though increasing bandwidth and decreasing storage costs might mitigate the latter.

References

1. Menezes AJ, Van Oorschot PC, Vanstone SA: *Handbook of applied cryptography*. Boca Raton: CRC Press; 1997.
2. Ducut E, Liu F, Fontelo P: **An update on Uniform Resource Locator (URL) decay in MEDLINE abstracts and measures for its mitigation.** *BMC Med Inform Decis Mak* 2008, **8:**-.
4. Wren JD: **URL decay in MEDLINE - a 4-year follow-up study.** *Bioinformatics* 2008, **24:**1381-1385.
9. Wren JD, Johnson KR, Crockett DM, Heilig LF, Schilling LM, Dellavalle RP: **Uniform resource locator decay in dermatology journals - Author attitudes and preservation practices.** *Arch Dermatol* 2006, **142:**1147-1152.
10. Ioannidis JPA, Allison DB, Ball CA, Coulibaly I, Cui XQ, Culhane AC, Falchi M, Furlanello C, Game L, Jurman G, et al: **Repeatability of published microarray gene expression analyses.** *Nature Genetics* 2009, **41:**149-155.
12. Wicherts JM: **Psychology must learn a lesson from fraud case.** *Nature* 2011, **480:**7-7.
17. **The Internet Archive** [http://www.archive.org/web/web.php]
18. Eysenbach G, Trudell M: **Going, going, still there: Using the WebCite service to permanently archive cited web pages.** *Journal of Medical Internet Research* 2005, **7:**2-6.
20. Peng RD: **Reproducible Research in Computational Science.** *Science* 2011, **334:**1226-1227.
21. **Key Facts on Digital Object identifier System** [http://www.doi.org/factsheets/DOIKeyFacts.html]
47. Hennessey J, Ge S: **A cross disciplinary study of link decay and the effectiveness of mitigation techniques.** *Bmc Bioinformatics* 2013, **14:**S5.
48. Rothenberg J: **Ensuring the Longevity of Digital Documents.** *Scientific American* 1995, **272:**42-47.
49. Rosenthal DSH: **Format obsolescence: assessing the threat and the defenses.** *Libr Hi Tech* 2010, **28:**195-210.
50. Traynor C, Williams MG: **Why are geographic information systems hard to use?** 1995**:**288-289.
51. Howe B: **Virtual Appliances, Cloud Computing, and Reproducible Research.** *Computing in Science & Engineering* 2012, **14:**36-41.
52. **Sitemaps.org** [http://www.sitemaps.org/]
53. **RDF Primer** [http://www.w3.org/TR/2004/REC-rdf-primer-20040210/]
54. **Ensembl Annotated Human Genome Data (MySQL Release 68)** [https://aws.amazon.com/datasets/2315]
55. Katayama T, Nakao M, Takagi T: **TogoWS: integrated SOAP and REST APIs for interoperable bioinformatics Web services.** *Nucleic Acids Res* 2010, **38:**W706-W711.
56. Brase J: **DataCite - A Global Registration Agency for Research Data.** *2009 Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology* 2009**:**257-261.

57. Wang X, Yu H: **How to Break MD5 and Other Hash Functions.** In *Advances in Cryptology – EUROCRYPT 2005. Volume* 3494. Edited by Cramer R: Springer Berlin Heidelberg; 2005: 19-35: *Lecture Notes in Computer Science*].
58. Jin K, Miller EL: **The effectiveness of deduplication on virtual machine disk images.** In *Book The effectiveness of deduplication on virtual machine disk images* (Editor ed.^eds.). pp. 1-12. City: ACM; 2009:1-12.
59. Ng C-H, Ma M, Wong T-Y, Lee PPC, Lui JCS: **Live deduplication storage of virtual machine images in an open-source cloud.** In *Book Live deduplication storage of virtual machine images in an open-source cloud* (Editor ed.^eds.). pp. 80-99. City: International Federation for Information Processing; 2011:80-99.
60. **Linked Virtual Machines** [http://pubs.vmware.com/vsphere-50/topic/com.vmware.wssdk.pg.doc_50/PG_Ch11_VM_Manage.13.4.html]
61. **Ubuntu Cloud Images** [http://cloud-images.ubuntu.com/]
62. Tridgell A: **Efficient Algorithms for Sorting and Synchronization.** Australian National University, 1999.
63. **lrzip: Long Range ZIP or Lzma RZIP** [http://ck.kolivas.org/apps/lrzip/]
64. **Data compression programs** [http://www.mattmahoney.net/dc/]
65. Smith MA, Pieper J, Gruhl D, Real LV: **IZO: Applications of Large-Window Compression to Virtual Machine Management.** In *LISA*. 2008: 121-132.
66. Lam MSL, Sapuntzakis CP, Chandra RUV, Zeldovich NB, Rosenblum M, Chow JE, Brumley DJ: **Cache-based system management architecture with virtual appliances, network repositories, and virtual appliance transceivers.** In *Book Cache-based system management architecture with virtual appliances, network repositories, and virtual appliance transceivers* (Editor ed.^eds.). City: Google Patents; 2008.
67. Reich J, Laadan O, Brosh E, Sherman A, Misra V, Nieh J, Rubenstein D: **VMTorrent: scalable P2P virtual machine streaming.** In *Book VMTorrent: scalable P2P virtual machine streaming* (Editor ed.^eds.). pp. 289-300. City: ACM; 2012:289-300.
68. Lagar-Cavilla HA, Whitney JA, Scannell AM, Patchin P, Rumble SM, De Lara E, Brudno M, Satyanarayanan M: **SnowFlock: rapid virtual machine cloning for cloud computing.** In *Proceedings of the 4th ACM European conference on Computer systems*. ACM; 2009: 1-12.
69. Van Gorp P, Mazanek S: **SHARE: a web portal for creating and sharing executable research papers.** *Procedia Computer Science* 2011, **4:**589-597.
70. **QEMU Official OS Support List - MS-DOS:** [http://www.claunia.com/qemu/objectManager.php?sClass=application&iId=53]
71. **Creating a MS-DOS Virtual PC under Virtualbox** [https://mylinuxramblings.wordpress.com/2010/12/05/linux-mint-debian-edition-lmde-first-impressions/]
72. Angiuoli S, Matalka M, Gussman A, Galens K, Vangala M, Riley D, Arze C, White J, White O, Fricke WF: **CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing.** *Bmc Bioinformatics* 2011, **12:**356.
73. **Bio-Linux 6.0** [http://nebc.nox.ac.uk/tools/bio-linux]

74. Wang Y, Mehta G, Mayani R, Lu JX, Souaiaia T, Chen YH, Clark A, Yoon HJ, Wan L, Evgrafov OV, et al: **RseqFlow: workflows for RNA-Seq data analysis.** *Bioinformatics* 2011, **27:**2598-2600.
75. Klinginsmith J, Mahoui M, Wu YM: **Towards Reproducible eScience in the Cloud.** 2011**:**582-586.
76. Goecks J, Nekrutenko A, Taylor J, Galaxy T: **Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.** *Genome Biol* 2010, **11**.
77. Leisch F: **Sweave: Dynamic generation of statistical reports using literate data analysis.** In *Compstat*. Springer; 2002: 575-580.

APPENDIX

From Chapter 2 – Link decay

Readme

```
Files in this directory were used for processing URLs as well as
checking their statuses. I hope they are helpful to you!

They are stated roughly in the order of running.

Python programs are written for python version 2.6 or 2.7. Tunables can
be found at the top of each file.
R programs were run with Revolution R, Community 6, which is based on R
2.14.2.

For the programs that take CSV files for input, the most important
columns are url and PY (Published Year). All others will be passed
along untouched.
```

```
If you use these files in your published work, it would be appreciated
if you could cite this paper.

For any questions on these files, you may contact:
jason.hennessey@jacks.sdstate.edu

--- The Files ---
README.txt:
      This very important documentation :)

extract_urls.py:
Output one URL per line from a tab-delimited file (such as that output
by ISI Web of Science) to a CSV.
New columns (in addition to including all others):
      extractText: A little bit of text surrounding the URL for some
additional context
      url_num: URL number in this abstract; starts at 0
      url
      host: hostname part of the URL
      dom: Top level domain name of host

Example:
      $ python extract_urls.py articles.txt urls.csv

Note: short descriptions of additional fields may be found at:
http://images.webofknowledge.com/WOKRS410B4/help/WOS/h_fieldtags.html

scheduler.py:
Meant to be run once a day (perhaps from cron) in order to schedule
check_urls_web.py to be run at random times.
Requires the "at" command from a Unix-like environment.
Example:
      $ python /home/user/scheduler.py

check_urls_web.py:
Scans a list of URLs and, in a parallel fashion, checks to see if they
are still active and valid.
Using the same status for duplicate URLs.
Takes a CSV for input, and outputs a CSV with the additional fields of
"web<date&time>" and "web_reason<date&time>".
For example:
      web2011-04-17-11-26,web_reason2011-04-17-11-26
Can either have separate input and output files or, if just one file is
specified, will overwrite the specified file.
Example:
      $ python check_urls_web.py urls.csv

check_urls_archived.py
Checks whether URLs are included in the Internet Archive or WebCite.
Like check_urls_web.py, takes CSV files as input and output. Creates
new columns for each archival method requested, the names for which
include the date.
Example:
      $ python check_urls_web.py urls.csv

submit_urls.py
```

```
Submits URLs to either the Internet Archive Wayback Machine or WebCite
that have both a successful web status and are not currently present in
that archive.
Lots of delays and other niceness built in so as to not overwhelm the
servers.

Currently depends on the 2011 web statuses, and uses the same "web
present" status as the analysis used (up >= 90% of the time). This can
be changed by modifying testUrls().

Unlike the check* programs, this one depends on finding web and archive
status columns in addition to the url column.

Called as: <program> <-i|-w|-iw> <input file> <output file> [-c
output_column to continue]
If -c passed, uses output_column in the <output file>[-n] to continue
from where the previous run left off. Creates a new filename of <output
file>-n where n is incremented and starts at 1. Use only the basename
for
<output file> and the program will figure out the latest one to use

Example:
$ python submit_urls.py -iw urls.csv urlsSubmitted.csv

For R files:
SHOW_THINKING marks some of the thought processes that went into
decisions that were made, and has primarily been left FALSE for the
whole development period.

Near the top of stats.R, there are two functions: installLibs() and
loadLibs()
The commands in installLibs() will need to be run before things will
work.
If not running all files in order, loadLibs() canhelp during
development to load the most used libraries.

analysis/common_raw.R:
Handles construction of the usable dataset.
Takes as input the CSV outputted by the various python files above.
Because the run could take a while, a few optimizations (such as saving
RData files) were used to speed development. If you are using these,
change READ_CACHE, WRITE_CACHE and CACHE_FILENAME to suit your needs.

If modifying common_raw.R extensively, you can change the commented-out
save/load statements to use an RData file instead of a csv every time
(which is slower).

You will have to modify the first read.csv() call to point to the
proper input file.

common_raw.R is meant to be source()'d from stats.R

analysis/output.txt:
The raw output from stats.R when run against our data set.

analysis/stats.R:
Primary statistical analysis file.
```

```
analysis/WOSstats.R:
For calculating some of the fit statistics with Web of Science data.
```

extract_urls.py

```
#!/usr/bin/python

# Version 1.2 of the URL extractor
# Copyright 2013, Jason Hennessey. See README.txt for license.

# Extracts the URLs from a tab-delimited file, such as that output by
ISI Web
# Of Science and output them to a CSV


### CHANGELOG
# v1.2: Added 'badchars', which will prevent disjoint URL joining
#        - Added single quote and charat to the badendings, which will
be
#        stripped
#        - Added logic in matchUrls() to eliminate duplicate URLs from
the same
#        article


import csv,re

# Unlikely endings for URLs to be stripped
badendings = '.":;)}\'<>[]'

# Characters that won't be in a URL component
badchars = ['<','>','\'', '\"', '\`']

# Number of characters before a URL to include for informational
purposes
appendExtraStart = 30

# Number of characters after a URL to include for informational
purposes
appendExtraEnd = 60

# These are some statistics kept for informational purposes
corrected = 0
total = 0
linecnt = 0

firstY = 3000 # First year seen -- will always be rounded down to
lowest
lastY = 0 # Last year seen -- will always be rounded up to highest

filters = [re.compile("^\S*/\S*"),  # Look for a slash in the block
           re.compile("^\S*\.html?"), # Look for .htm[|l]
           re.compile("^~\S*"), # Catch spaces then tilde
           re.compile("^\.\S+"), # If the domain name is in there
           re.compile("^\S+\.\S*")] # Try to catch latent domain names
                                    # Common error is: "www.
domain.edu"

http_next = re.compile("^\S*http\S*") # Find http in the next word
```

```
def addMore(extraStr):
        '''Look for trailing URL stuff separated by a space.
        Returns a tuple containing (any extra, a relative index of the
last
        character scanned to'''

        if len(extraStr) < 2 or extraStr[0] != ' ' or extraStr[1] in
badchars:
            return ('',0)

        # Lop off the first space
        extraStr = extraStr[1:]

        # Having "http" in the next word is a show stopper, since that
        # indicates the start of a new URL
        if http_next.search(extraStr) != None:
            return ('',0)

        # Look for each of the regexps
        for regexp in filters:
            a = regexp.search(extraStr)
            if a != None:
                end = a.end()
                ret = extraStr[:a.end()]
                global corrected
                corrected += 1
                extra, newend = addMore(extraStr[end:])
                return ret + extra, newend + end

        return ('',0) # Didn't find anything this time

#url_patt = re.compile("([0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-
9]{1,3}|(((news|telnet|nttp|file|http|ftp|https)://)|(www|ftp)[-A-Za-
z0-9]*\\.)[-A-Za-z0-9\\.]+)(:[0-9]*)?/[-A-Za-z0-
9_\\$\\.\\+\\!\\*\\(\\),;:@&=\\?/~\\#\\%]*[^]'\\.}>\\),\\\"]",
re.IGNORECASE)

# Matches just the hostname portion
#url_patt = re.compile("https?://[a-zA-Z0-9\-\.]*", re.IGNORECASE)

# Easy expression... let's start this way
# Ensure there is at least one . somewhere...
url_patt = re.compile("(?:ht|f)tps?://[a-zA-Z0-9\-]+\.[^\s\[\]\)]*")

def matchUrls(urlStr):

        '''Examines an abstract and returns a list containing tuples
of:
        (URL, start_offset, end_offset)'''

        urls = []

        # Translate all commas into .'s. Sometimes these get mistaken.
Also,
        # less chance for the output CSV to be corrupted
        urlStr = urlStr.replace(',', '.')
        # Replace improbable/likely OCR bug ' .' with '.'
        urlStr = urlStr.replace(' .', '.')

        # Replace instances of "similar to" with "~", as that is how
the OCR
        # seems to have sometimes translated it
        urlStr = urlStr.replace('/ similar to ', '/~')
        urlStr = urlStr.replace('/similar to ', '/~')
        urlStr = urlStr.replace('/similar to', '/~')
        urlStr = urlStr.replace('/ similar to', '/~')
        urlStr = urlStr.replace('/(similar to) ', '/~')
        urlStr = urlStr.replace('/(similar to)', '/~')
        urlStr = urlStr.replace('/ (similar to)', '/~')


        matches = url_patt.finditer(urlStr)
        found = []

        prev_urls = set()

        # Look for more OCR errors where there is a
URL<space>continued_URL
        for match in matches:
            start = match.start()
            end = match.end()
            urlMatch = urlStr[start:end]

            # Check for any additional things that might have been
mistaken
            # due to OCR
            extraStr, extraEnd = addMore(urlStr[end:])
            urlMatch += extraStr
            pre_len = len(urlMatch)
            urlMatch = urlMatch.rstrip(badendings)
            extraEnd -= pre_len - len(urlMatch)

            # Check if we've already recorded this URL for this article
            if urlMatch in prev_urls:
                continue
            prev_urls.add(urlMatch)
            found.append((urlMatch, start, end + extraEnd))

        return found

from urlparse import urlparse

def extractUrls(inCSV, outCSV):
       '''Extracts the URLs from a passed-in DictReader
        Outputs to the given csv.writer(). We write the header to
outCSV'''

       global firstY,lastY, appendExtraStart, appendExtraEnd, linecnt

       for line in inCSV:
           # Search for URLs in the abstract
           linecnt += 1
```

```
            abstract = line['AB']
            urls = matchUrls(abstract)

            # Update the min/max year if required
            year = int(line['PY']) if line.has_key('PY') else ''
            if year > lastY:
                lastY = year

            if year < firstY:
                firstY = year

            url_num = 0
            for url, start, end in urls:
                # First - make sure it can be parsed
                # In practice, urlparse() takes pretty much anything,
but one
                # can hope
                parsed = host = ''
                try:
                    parsed = urlparse(url)
                except:
                    continue

                host = parsed.hostname
                parts = host.split('.')
                if len(parts) > 1:
                        dom = parts[-1] if not host.endswith('.') else
parts[-2]
                else:
                        dom = ''

                # Fix up start and end indexes to be included as a
snippet
                # Due to replacements these may not be exact, but
                # the appendExtraStart and appendExtraEnd parameters
should give enough
                # leeway
                if start - appendExtraStart < 0:
                    start = 0
                else:
                    start = start - appendExtraStart

                endIdx = len(abstract)
                if end + appendExtraEnd > endIdx:
                    end = endIdx
                else:
                    end = end + appendExtraEnd

                line['extraText'] = abstract[start:end]
                line['url_num'] = url_num
                line['url'] = url
                line['host'] = host
                line['dom'] = dom

                outCSV.writerow(line)
                global total
                total += 1
                url_num += 1

 import sys
 def main(args = sys.argv):
     '''<program> <input file> <output-file>. Use "-" for stdin or
 stdout'''
     inStr = args[1]
     outStr = args[2]

     import time
     begin = time.time()

     inF  = sys.stdin if inStr == "-" else open(inStr, "rU")
     outF = sys.stdout if outStr == "-" else open(outStr, "wb")

     try:
         inCSV = csv.DictReader(inF, dialect="excel-tab")

         # Calculate list of outputted fields
         # Consists of new ones + sorted old ones
         outFields = ['url', 'host', 'url_num', 'extraText', 'dom']
         outFields.extend(sorted(inCSV.fieldnames))

         outCSV = csv.DictWriter(outF, outFields, dialect="excel")

         # Create and write a header row
         header = dict()
         for field in outFields:
             header[field] = field
         outCSV.writerow(header)

         extractUrls(inCSV, outCSV)

     finally:
         inF.close()
         outF.close()

     print "Total URLs corrected/processed (%.1f%%) for %s[lines = %d]:"
 \
     % ((corrected * 100)/(total + .0001), inStr, linecnt), corrected,
 total, \
     "Year range: %d-%d" %(firstY,lastY)
     print "Total runtime: %d seconds" % (time.time() - begin)


 if __name__ == "__main__":
     main()
```

scheduler.py

```
#!/usr/bin/python

### Intended to be run once a day (preferably at midnight) to schedule
### 3 random runs during a day of a given program
### Will only schedule runs between (midnight + spacing/2) and
(midnight + 24
### hours - spacing/2) in order to adhere to the spacing rules
### Copyright 2013, Jason Hennessey. See README.txt for license.

# Tunables
spacing = 2 # Minimum time (in hours) by which runs must be separated
spacing_secs = spacing * 3600
num     = 3 # Number of times to schedule the script per day


cwd = "/home/user/"
cmdFile = "run_check_urls_web.txt"

log = cwd + "/scheduler.log"


from datetime import datetime, timedelta
import random

def getTimes(n):
    '''Generate a list of 'num' times spaced 'spacing' number of hours
apart'''
    times = []

    # When is it?
    now = datetime.now()

    # Obtain midnight for calculations
    midnight = datetime(now.year, now.month, now.day)

    early = midnight + timedelta(hours=spacing/2)
    late = midnight + timedelta(days=1) - timedelta(hours=spacing/2)

    span = (late - early).seconds

    # Initially populate times with integers; convert them to datetime
    # afterwards
    times = [random.randint(0,span)] # Pick an initial time
    while len(times) < n:
        candidate = random.randint(0,span)
        tooClose = [x for x in times if abs(x - candidate) <
spacing_secs]
        if not tooClose:
            times.append(candidate)

    # Return times converted to datetime type
    return [early + timedelta(seconds=x) for x in times]

def logTimes(log, cmds):
    outF = open(log, "a")

    outF.write("Scheduler running at %s\n"
%(datetime.now().isoformat(),))
    for x in cmds:
        outF.write("%s\n" % x)
    outF.write('\n')

    outF.close()

import os
def scheduleTimes(times):
    '''Schedule (using at) the times listed. Returns a list of strings
executed'''
    os.chdir(cwd)

    cmds = []
    for t in times:
        execute = "at -f %s -t %s" % (cmdFile,
t.strftime('%Y%m%d%H%M.%S'))
        os.system(execute)
        cmds.append("%s # %s" % (execute, t.ctime()))

    return cmds

import sys
def main(args = sys.argv):
    # Obtain a list of times
    times = getTimes(num)

    # Schedule the runs
    cmds = scheduleTimes(times)

    # Log
    logTimes(log, cmds)

if __name__ == "__main__":
    main()
```

check_urls_web.py

```
#!/usr/bin/python

### Test URLs to see if they are valid or not.
### Requires python 2.6
### v1.1
### Copyright 2013, Jason Hennessey. See README.txt for license.

import urllib2
from urllib2 import HTTPError, URLError, urlopen
from httplib import HTTPException
from urlparse import urlparse

import threading

import socket

from multiprocessing import Pool
import sys, time, csv, re

# URL cache so we don't recheck the same URLs
# Key: url Value: (success, code)
urlcache = dict()

## Tunables
drop_duplines = False # Whether to omit entries for duplicate URLs
                      # We will always use a previous lookup if one
                      # is available in the cache

timeout = 15 # socket timeout in seconds
processes = 15 # Number of processes to create

# These are some statistics kept for informational purposes
success = 0
success_temp = 0
total = 0

firstY = 3000 # First year seen -- will always be rounded down to
lowest
lastY = 0 # Last year seen -- will always be rounded up to highest

def process(line, fields):
    code = fetch = parsed = scheme = None

    try:
        global timeout
        parsed = urlparse(line['url'])
        scheme = parsed.scheme

        req = urllib2.Request(line['url'])

        # Adjust user-agent to be a desktop browser
        req.add_header('User-Agent', 'Mozilla/5.0 (Windows; U; Windows
NT 6.1; '\
        'en-US; rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13')
        fetch = urlopen(req, timeout=timeout)

    except HTTPError, e:
        code = e.code
    except URLError, e:
        error = str(e.reason)
        if ("http" in scheme and "Name or service not known" in error)
or \
            ("ftp" in scheme and "No address associated with hostname"
in \
            error):
            code = "lookup failure"
        elif "Connection refused" in error:
            code = "connection refused"
        else:
            code = error
    except HTTPException:
        code = "unknown http error"
    except KeyboardInterrupt:
        print "Caught KeyboardInterrupt; exiting"
        sys.exit()
    except:
        code = "unknown exception raised"
    else:
        fetch.close()

    if (code is None):
        status = 'True'
        code = fetch.getcode()
    else:
        status = 'False'

    statusF, reasonF = fields
    line[statusF] = status
    line[reasonF] = code

    return (line,fields)

outList = []

outLock = threading.Lock()

# Assuming this runs in the context of the parent
def anotherIn((line, (statusF, reasonF))):
    global outList,outLock,total,success,success_temp,urlcache

    with outLock:
        i = len(outList)
        outList.append(line)
    if i % 100 == 0 and i > 0:
        print "Success rate for group %d:" % (i,), success_temp, '%'
        success += success_temp
        success_temp = 0

    total += 1
    if line[statusF] == 'True':
            success_temp += 1

        # Update the cache
        urlcache[line['url']] = (line[statusF], line[reasonF])

def deferredIn(deferred, (statusF, reasonF)):
    '''Function to complete entries not passed along due to caching'''
    global urlcache

    results = []
    for line in deferred:
        status, code = urlcache[line['url']]
        line[statusF] = status
        line[reasonF] = code
        results.append(line)

    return results

def testUrls(pool, inCSV, outCSV, fields):
    '''pool: process Pool, inCSV: DictReader, outCSV: plain CSV
writer'''

    begin = time.time()
    print "Starting", time.ctime()

    global firstY,lastY,urlcache,outList,outLock

    # Special treatment given to previously-outputted files: If we see
    # a "success" line, pass it along verbatim (we're only trying to
refine
    # the failed ones)
    recycled = True if "web" in inCSV.fieldnames else False

    deferred = [] # List of dict()'s

    # Fill the input queue
    count = 0
    for line in inCSV:
        count += 1

        if recycled and line['web'] == 'True':
            anotherIn((line, fields))
            continue

        url = line['url']
        if url not in urlcache:
            urlcache[url] = None
            pool.apply_async(process, (line,fields),
callback=anotherIn)
            year = int(line['PY'])

            # Keep year stats
            if year > lastY:
                lastY = year
            if year < firstY:
                firstY = year
        else:
                # URL is in cache
                if drop_duplines:
                    continue

                # Add to the deferred list to be completed at the end
                deferred.append(line)

    cachecount = count - len(urlcache) if drop_duplines else
len(deferred)

    if drop_duplines:
        print "%d lines dispatched (%d omitted). Waiting..." %
(count,cachecount),\
        time.ctime()
    else:
        print "%d lines dispatched (%d cached). Waiting..." %
(count,cachecount),\
        time.ctime()

    # Need to wait for the rest of the jobs to finish,
    # then write out their data
    pool.close()
    pool.join()

    print "Workers done. outList = %d" % (len(outList),)
    print time.ctime()

    # Incorporate the deferred entries
    outList.extend(deferredIn(deferred,fields))

    from operator import itemgetter

    # Sort the list
    outList.sort(key=itemgetter('PY'))

    # Output format

    for line in outList:
        outCSV.writerow(line)

    print "Took", round(time.time() - begin), "seconds total"

    global success,success_temp,total
    success += success_temp

    print "Total found/total (%.1f%% success rate) (deduplicated %d):"
\
    % ((success * 100)/((count - cachecount) + .0001), cachecount),
success,\
    count, "Year range: %d-%d" % (firstY,lastY)


import fcntl
def main(args = sys.argv):
    '''Called as: <program> <input file> [output file]
    If output not specified, input file is overwritten with results'''
    import time
    begin = time.time()

    inStr = args[1]
    outStr = args[2] if len(args) >= 3 else None

    inF  = sys.stdin if inStr == "-" else open(inStr, "rU")

    # If input and output files are the same, create a tempfile

    outF = None
    sameFile = False
    if outStr == None or (inStr == outStr and outStr != "-"):
        import tempfile
        sameFile = True
        outF = tempfile.NamedTemporaryFile(mode="wb", delete=False)
    else:
        outF = sys.stdout if outStr == "-" else open(outStr, "wb")

    pool = None
    normalFinish = False # Boolean to see if we finished normally
                         # and thus should overwrite an existing output
file
    try:
        # Take exclusive lock on input file
        if inStr != "-":
            fcntl.flock(inF, fcntl.LOCK_EX)

        inCSV = csv.DictReader(inF, dialect="excel")

        # The newly generated columns are formatted:
        # web-YYYY-MM-DD-HH-MM
        import datetime
        cur = datetime.datetime.now()
        timeFmt = "%04d-%02d-%02d-%02d-%02d" % (cur.year, cur.month,
cur.day,\
                                              cur.hour, cur.minute)

        statusF = 'web' + timeFmt
        reasonF = 'web_reason' + timeFmt

        outFields = [statusF, reasonF]
        inFields = inCSV.fieldnames

        # Check for empty input
        if not inFields:
            print "Error: empty input file"
            sys.exit(1)

        # Ensure we aren't creating duplicate fields
        shouldBeEmpty = [x for x in inFields if x in outFields]
        if len(shouldBeEmpty) > 0:
            print "Error: column(s)", shouldBeEmpty ,"in input and
output"
            sys.exit(1)

        # Take all incoming fields and prepend the new ones
        outFields.extend(inFields)

        # Use commas for output
        outCSV = csv.DictWriter(outF, outFields, dialect="excel")

        # Create and write a header row
        header = dict()
        for field in outFields:
            header[field] = field
        outCSV.writerow(header)

        # Start process pool
        global processes
        pool = Pool(processes=processes)

        fields = (statusF, reasonF)
        testUrls(pool, inCSV, outCSV, fields)

        normalFinish = True

    except KeyboardInterrupt:
        print "Main caught Keyboard. Exiting..."
    finally:
        if inStr != "-":
            fcntl.flock(inF, fcntl.LOCK_UN)
            inF.close()
        if outStr != "-":
            outF.close()

        # If we used a temporary file, move the temp file to the input
file,
        # since that was what was requested
        if sameFile and normalFinish:
            import shutil
            shutil.move(outF.name, inStr)

        if pool:
            pool.terminate()

        if not normalFinish:
            print "Did NOT finish normally"


    print "Total runtime: %d seconds" % (time.time() - begin)

if __name__ == "__main__":
    main()
```

check_urls_archived.py

```
#!/usr/bin/python

### Copyright 2013, Jason Hennessey. See README.txt for license.
### Test URLs to see if they are valid or not. Purposely unthreaded,
since we're
### utilizing finite server resources
### Requires python 2.6.
### V1.1

### Dependencies: httplib2

# v1.1: Reduced tries to 4 after seeing that it's an optimal
time/reward
# balance


## Tunables
timeout = 60 # socket timeout in seconds

give_status = 100 # Print status every 100 entries

ia_enabled = True  # Internet Archive checking enabled
wc_enabled = True # WebCitation.org checking enabled

max_tries = 4   # Maximum number of tries if we get a 503 or other
transient
                # response from IA

# Configure methods
methods = set()
if ia_enabled:
    methods.add('ia')

if wc_enabled:
    methods.add('wc')

# URL cache so we don't recheck the same URLs
# Key: url Value: (success, code)
# One URL cache per method
urlcache = dict()

for method in methods:
    urlcache[method] = dict()

## Statistics
firstY = 3000 # First year seen -- will always be rounded down to
lowest
lastY = 0 # Last year seen -- will always be rounded up to highest

import sys, time, csv

# URL to which we append the desired URL
ia_url = 'http://web.archive.org/web/*/'
wc_url = 'http://www.webcitation.org/query?returnxml=true&url='
```

```
# URL fetching stuff...
import httplib2
http = httplib2.Http(cache=".cache", timeout=timeout)

from xml.dom import minidom

def process(method, url):
  """ Attempts to access a URL's status with the given mechanism.
Returns the
  status (True = Suceess; False = Failure).

  Methods consist of: ia (internet archive), wc (webcitation.org) and
web (if
  URL is still live; not implemented)"""

  code = resp = None

  for tries in xrange(max_tries):
    if tries > 0:
        time.sleep(tries * 1.1) # Increase our backoff if the server is
too busy
    try:
        if method == 'ia':
            ialine = ia_url + url
            resp, data = http.request(ialine, "HEAD")

            status = int(resp['status'])

            if status == 404:
                return False
            elif status == 200:
                return True
            elif status == 403:
                return 'CrawlingBlocked' # Crawling blocked by
robots.txt
            elif status == 503:
                print "URL(ia):", url, "status = ", status, "try",
tries
                continue # Let's try again
            else:
                print "URL(ia):", url, "status = ", status

        elif method == 'wc':
            # For Webcitation, we parse the XML returned to determine
if
            # the URL is archived
            wcline = wc_url + url
            resp, data = http.request(wcline, "GET")

            status = int(resp['status'])
            if status != 200:
                print  "URL(wc):", url, "status = ", status

            # Embed in "try" in case XML isn't what we expected
            # Expected XML format:
```

```
            # (per
http://webcitation.org/doc/WebCiteBestPracticesGuide.pdf)
            #   Success is <queryresult><resultset><result status="...
            #   Error is <queryresult><error>
            try:
                xml = minidom.parseString(data)

                # For readability below.
                # FC should point to either <resultset> or <error>
                FC = xml.firstChild.firstChild

                if FC.tagName == 'error':
                    return False

                for x in FC.childNodes:
                    if x.attributes['status'].value == 'success':
                        return True

                # No success entry found, but request wasn't in error
either
                return 'NoSuccessXML'
            except:
                return 'UnexpectedXML'

        else:
            # Unimplemented method
            print "Unimplemented method in process()!"
            sys.exit()
    except KeyboardInterrupt:
        print "Caught KeyboardInterrupt; exiting"
        sys.exit()
    except:
        code = "unknown exception raised"

    return False
  return '503retryExpired' # We couldn't get it after trying several
times

def testUrls(inCSV, outCSV, methodFields):
    """Run through the URLs and check them.

     inCSV: DictReader, outCSV: DictWriter,
     fieldOut: dictionary mapping methods to their output field
names"""

    begin = time.time()
    print "Starting", time.ctime()

    global firstY,lastY,urlcache
    # Special treatment given to previously-outputted statuses: If we
see
    # a "success" line, pass it along verbatim (we're only trying to
refine
    # the failed ones).
    inFields = inCSV.fieldnames

    count = 0
    # Run the tests
    for line in inCSV:
        count += 1

        if (count % give_status) == 0:
             print "Completed", count

        # Keep year stats
        global lastY, firstY

        year = int(line['PY'])
        if year > lastY:
             lastY = year
        if year < firstY:
             firstY = year

        url = line['url']

        for method,methodF in methodFields.iteritems():
            if url in urlcache[method]:
                # Try to use a cached value if one exists
                line[methodF] = urlcache[method][url]
            else:
                # Not cached - run it!
                ret = process(method,url)
                urlcache[method][url] = line[methodF] = ret

        outCSV.writerow(line)

    print count," lines checked;", methodFields.keys(), time.ctime()

    print "Took", round(time.time() - begin), "seconds total"

def main(args = sys.argv):
    '''Called as: <program> <input file> <output file>'''
    inStr = args[1]
    outStr = args[2]

    import time
    begin = time.time()

    inF  = sys.stdin if inStr == "-" else open(inStr, "rU")
    outF = sys.stdout if outStr == "-" else open(outStr, "wb")

    try:
        inCSV = csv.DictReader(inF, dialect="excel")

        # Keep all incoming fields, and ensure that the ones we want
are included
        # for output
        inFields = inCSV.fieldnames
        outFields = sorted(list(methods)) # Convert back to list for
ordering
        import datetime
        cur = datetime.datetime.now()
        timeFmt = "%04d-%02d-%02d" % (cur.year, cur.month, cur.day)
        fieldOut = {} # dict mapping the methods to their output
columns
        for m in methods:
            fieldOut[m] = m + timeFmt

        outFields = fieldOut.values()
        outFields.extend(inFields)

        outCSV = csv.DictWriter(outF, outFields, dialect="excel")

        # Create and write a header row
        header = dict()
        for field in outFields:
            header[field] = field
        outCSV.writerow(header)

        testUrls(inCSV, outCSV, fieldOut)

    except KeyboardInterrupt:
        print "Main caught Keyboard. Exiting..."
    finally:
        if inStr != "-":
            inF.close()
        if outStr != "-":
            outF.close()

    print "Total runtime: %d seconds" % (time.time() - begin)

if __name__ == "__main__":
    main()
```

submit_urls.py

```
#!/usr/bin/python

### Copyright 2013, Jason Hennessey. See README.txt for license.
### Submit URLs to archiving engines
### utilizing finite server resources
### Requires python 2.6.
### V1.1

### Dependencies: httplib2

# v1.1: Reduced tries to 4 after seeing that it's an optimal
time/reward
# balance


## Tunables
timeout = 300 # socket timeout in seconds. Set high due to WC

give_status = 100 # Print status every 100 entries

inArchiveDate = '2012-10-18' # Which archival fields to use

minSleep = 3840 # Min time to sleep during WC's rate limiting
maxSleep = 4*3600 # Max time to sleep during backoffs

# For time-related names
import datetime
cur = datetime.datetime.now()
timeFmt = "%04d-%02d-%02d" % (cur.year, cur.month, cur.day)

# Configure debug logging
debugFileName = "debugSubmitUrls" + timeFmt + ".txt"
debugFile = None

# Methods- set that holds which methods (ia, wc) we'll use
methods = set()

# URL cache so we don't recheck the same URLs
# Key: url Value: (success, code)
# One URL cache per method
urlcache = dict()

import sys, time, csv

# URL to which we append the desired URL
ia_url = 'http://liveweb.archive.org/'
wc_url =
'http://www.webcitation.org/archive?returnxml=true&email=jason.hennesey
@jacks.sdstate.edu&url='

# URL fetching stuff...

userAgent = 'Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US;
rv:1.9.2.13) '\
            'Gecko/20101203 Firefox/3.6.13'

httpHeaders = {'User-Agent':userAgent}

import httplib2
http = None
#httplib2.debuglevel = 255
def newHttp():
    '''Instantiates a new httplib2 instance. Done as a function so we
can
    reinit later if needbe'''
    global http
    http = httplib2.Http(timeout=timeout)
    #http = httplib2.Http(cache=".cache", timeout=timeout)

newHttp()

# For IA archiving - we spawn wget with a temp file to access the URL
import os,tempfile,subprocess
devnull = open(os.devnull, "wb")


# For processing webcitation responses
from xml.dom import minidom

# For adding delays in order to be nice users of the archive services
from datetime import datetime
import time

# Enforce a minimum delay between requests to be nice to the archive
servers
delay   = {'wc':130.0, 'ia':30.0}
lastRan = {'wc':datetime.min, 'ia':datetime.min} # Use min to avoid an
initial delay

def delayMin(method):
    '''Ensure that we sleep a minimum of 'delay' seconds in order to
    to be nice to the archive servers'''

    timeLen = datetime.now() - lastRan[method]
    secs = timeLen.microseconds / 1000000.0 + timeLen.seconds +
timeLen.days * 24*3600

    if secs < delay[method]:
        time.sleep(delay[method] - secs)


import socket,httplib

def webcite(wcline):
    # For Webcitation, we parse the XML returned to determine if
    # the URL was archived. It turns out that WC says it archived
    # everything you throw at it (even if it can't), but maybe it
    # could be updated
    try:
        resp, xmldata = http.request(wcline, "GET",
headers=httpHeaders)
```

```
    except httplib2.HttpLib2Error, e:
        print "httplib2 error (", wcline, "): ", e
        print >>debugFile,"httplib2 error (", url, "): ", e
        return 'httplib2 error'
    except socket.error, e:
        print 'httplib Socket error', e
        print >>debugFile,'httplib Socket error', e
        return 'socket'
    except httplib.HTTPException as e:
        print >>debugFile, "httplib error:", e
        return 'httplib error'

    print >> debugFile, xmldata

    if resp.status != 200:
        print  "URL(wc):", wcline, "status = ", resp.status
        print  >>debugFile,"URL(wc):", wcline, "status = ", resp.status

    # Check if webcite down
    if xmldata.find('WebCite is currently unavailable') >= 0:
        return 'wcDown'

    # Embed in "try" in case XML isn't what we expected
    # Expected XML format:
    # (per http://webcitation.org/doc/WebCiteBestPracticesGuide.pdf)
    #   Success is <queryresult><resultset><result status="...
    #   Error is <queryresult><error>
    try:
        wcXml = minidom.parseString(xmldata)

        res = wcXml.firstChild.getElementsByTagName("resultset")[0]

        # Check for error
        err = res.getElementsByTagName("error")
        if len(err) > 0:
            # Only one seen so far is "rate", though there could be
others
            return str(err[0].getAttribute("type")) + "Error"

        res = res.getElementsByTagName("result")

        if len(res) == 0:
            # No resultset - don't know how to process!
            return "NoResultNoError"

        res = res[0]
        status = res.getAttribute('status')

        if status != 'success':
            return 'status' + status

        shortId =
res.getElementsByTagName('webcite_id_short')[0].firstChild.data

        return shortId
    except:
        # Add fatal error check so that we don't end in an infinite
loop
        if xmldata.count("Fatal error"):
            return 'fatalError'
        else:
            return 'UnexpectedXML'

def process(method, url):
    """Attempts to archive a URL with the given mechanism. Returns the
    status (True|WC ID = Suceess; False = Failure).

    Methods consist of: ia (internet archive) and wc
(webcitation.org)"""

    global wcLast, iaLast

    code = resp = None
    delayMin(method)

    try:
        if method == 'ia':
            ialine = ia_url + url

            # Using wget with -O /dev/null doesn't work due to the
            # --page-requisites dependency processing analyzing an
empty file,
            # so we create a tempfile then delete it
            tempNum,tempName = tempfile.mkstemp()
            os.close(tempNum)

            # We need to request the page requisites in order to ensure
            # that they are archived (empirical tests show that ia only
            # fetches explicitly requested files).
            # Related files are stored on web.archive.org. Unrelated
            # files (javascript, surrounding images) are stored on
            # staticweb.archive.org.

            debugFile.flush()

            status = subprocess.call(['wget','-O',tempName,'--page-
requisites',
            '-H', '-D','web.archive.org','--exclude-domains',
            'staticweb.archive.org', '-e', 'robots=off','--wait',
'.25',
            ialine],stdin=devnull, stdout=debugFile, stderr=debugFile)

            debugFile.flush()

            os.remove(tempName) # We don't care about the contents of
the site; we're just
                                # fetching them so that the IA has them

            iaLast = datetime.now()

            if status == 0:
                return True
            else:
                return 'wgetStatus' + str(status)

        elif method == 'wc':
            wcline = wc_url + url

            # We attempt to access webcitation in a loop since in the
past
            # they've a) either blocked our IP or gone down (socket
timeout)
            # b) served up "WebCite has flagged your IP..." errors
            #
            # To address these concerns, implement an exponential
backoff that
            # resets each time we enter with min and max values shown
prudent
            # by experience
            wDelay = delay['wc']
            incDelay = True # Set until the first rateError so we only
inc
                            # delay['wc'] once

            tries = 0

            while tries < 10:
                status = webcite(wcline)

                if status in ['socket', 'rateError', 'UnexpectedXML',
'wcDown']:
                    # Exponential backoff
                    if status == 'rateError':
                        thisSleep = max(wDelay,minSleep)
                    else:
                        thisSleep = wDelay
                    print "Received status {0}".format(status)
                    print >> debugFile,time.ctime(),"Received status
{0} for URL {1}. "\
                    "Sleeping {2} seconds (wDelay: {3})".format(status,
url, thisSleep, wDelay)
                    debugFile.flush()
                    time.sleep(thisSleep)
                    wDelay = min(wDelay*2,maxSleep)
                    if status == 'rateError' and incDelay:
                        '''Increment the rate delay so we can avoid
hitting the
                        rate limit. Only done the first time we hit a
rate
                        limit for a URL.'''
                        delay['wc'] += 5 # Increment our delay to
hopefully
                                         # not hit this again
                        print >> debugFile,"Incrementing wc delay to",\
                        delay['wc']
                        incDelay = False
                    else:
                        # Socket error - Perhaps something in httplib2
                        # might not be working so let's create a new
instance.
```

```
                        newHttp()
                    else:
                        return status

                    # Keep looping if WC is down/limiting us rather than
proceeding
                    if status not in ['wcDown', 'rateError']:
                        tries += 1

                else: # while
                    return 'triesExhausted'
            else:
                assert False, "Unimplemented method"

    except KeyboardInterrupt:
        print "Caught KeyboardInterrupt; exiting"
        sys.exit()
    finally:
        lastRan[method] = datetime.now() # update time for minimum
delay

    assert False,"Shouldn't be here"


def testUrls(inCSV, outCSV, methodFields):
    """Run through the URLs and submit them.

     inCSV: DictReader, outCSV: DictWriter"""

    begin = time.time()
    print "Starting", time.ctime()
    print >>debugFile,"Starting", time.ctime()

    inFields = inCSV.fieldnames
    webFields = [f for f in inFields if f.startswith("web2011")]
    lenWebFields = len(webFields)

    count = iacount = wccount = 0

    # Run the tests
    global urlcache
    for line in inCSV:
        count += 1

        if (count % give_status) == 0:
             print "Completed %d urls, ia %d, wc %d" % (count, iacount,
wccount)
             print >>debugFile,time.ctime(),\
                   "Completed %d urls, ia %d, wc %d" % (count,
iacount, wccount)

        url = line['url']

        # Calculate the web metric using the same algorithm as done in
R
        # Repeated so that we aren't tossing data back and forth
between R and
```

```
        # python
        webT = [line[w] for w in webFields].count('True') * 1.0
        pTrue = webT / lenWebFields # Percent of true entries
        web = True if pTrue >= .9 else False
        print >>debugFile, time.ctime(),\
                "URL: {0:25} pTrue:{1:4} web:{2:5}".format(url,pTrue,\
                str(web))
        debugFile.flush()

        for method,(methodI,methodO) in methodFields.iteritems():
            if url in urlcache[method]:
                # Try to use a cached value if one exists
                line[methodO] = urlcache[method][url]
            else:
                # If not cached, run it as long as it's still alive and
wasn't
                # already archived using this method
                if web and line[methodI] != 'True':
                    ret = process(method,url)
                    if method == 'ia':
                        iacount += 1
                    elif method == 'wc':
                        wccount += 1
                else:
                    ret = 'Skipped'

                urlcache[method][url] = line[methodO] = ret

        outCSV.writerow(line)

    print "iacount = %d, wccount = %d" % (iacount, wccount)
    print >>debugFile,"iacount = %d, wccount = %d" % (iacount, wccount)

    print "Took", round(time.time() - begin), "seconds total",
time.ctime()
    print >>debugFile,"Took", round(time.time() - begin), "seconds
total",\
                      time.ctime()

def catchup(inCSV,outCSV,inF2,methodFields):
    '''Copy the already-completed lines from inCSV2 to outCSV,
    fast-forwarding through inCSV appropriately in the process'''

    begin = time.time()

    inCSV2 = csv.DictReader(inF2, dialect="excel")

    i = 0
    for line in inCSV2:
        line2 = inCSV.next()
        if line['url'] != line2['url']:
            print "Input and output file do not
agree.",line['url'],line2['url']
            sys.exit("Error in file synchronization")

        # Insert into cache if appropriate
        url = line['url']
```

```
            for method,(methodI,methodO) in methodFields.iteritems():
                if url not in urlcache[method]:
                    urlcache[method][url] = line[methodO]

            outCSV.writerow(line)
            i+= 1

        inF2.close()

        print "Reused", i, "lines."
        print >>debugFile,"Reused", i, "lines in", time.time() - begin,
"seconds"

def main(args = sys.argv):
    '''Called as: <program> <-i|-w|-iw> <input file> <output file> [-c
output_column to continue]
    If -c passed, uses output_column in the <output file>[-n] to
continue
    from where the previous run left off. Creates a new filename of
<output
    file>-n where n is incremented and starts at 1. Use only the
basename for
    <output file> and the program will figure out the latest one to
use'''

    if len(args) != 4 and len(args) != 6:
        print "Too {0} arguments".format('few' if len(args) < 4 else
'many')
        sys.exit(main.__doc__)

    # Is this a continuation?
    cont = False
    if len(args) == 6 and args[4] == '-c':
        cont = args[5]

    # Which methods were selected?
    if 'i' in args[1]:
        methods.add('ia')

    if 'w' in args[1]:
        methods.add('wc')

    for method in methods:
        urlcache[method] = dict()

    if len(urlcache) == 0:
        # We found no methods. Abort!
        print "No method selected. Please use -i, -w, or -iw"
        sys.exit(main.__doc__)

    inStr = args[2]
    outStr = args[3]

    if not (inStr == '-' or os.path.exists(inStr)):
        print "Input file does not exist"
        sys.exit(main.__doc__)
    # In case we are continuing, adjust outStr and inStr2 appropriately
    if cont:
        if outStr == '-':
            print "Error: -c specified while output is set to stdout."
            sys.exit(main.__doc__)

        if not os.path.exists(outStr):
            print 'Output file does not exist and -c specified'
            sys.exit(main.__doc__)

        import itertools
        for i in itertools.count(1): # find the next unused filename
            base = outStr + '-' + str(i)
            validSuffices = ['','.gz','.bz2','.xz']

            for test in validSuffices:
                if os.path.exists(base + test):
                    break
            else:
                break # We break here when we *don't* find a file

        inStr2 = outStr if i == 1 else outStr + '-' + str(i - 1)
        outStr = outStr + '-' + str(i)

    # Debug output
    global debugFile
    debugFile = open(debugFileName, "a")
    print "Logging to", debugFileName
    print "Continue set to", cont
    print "Input:", inStr, "Output:", outStr, "inStr2:", inStr2
    print >>debugFile,"Logging to", debugFileName
    print >>debugFile,"Continue set to", cont
    print >>debugFile,"Input:", inStr, "Output:", outStr, "inStr2:",
inStr2

    import time
    begin = time.time()

    # Open appropriate files. If there's an error, we'll bail here
    inF  = sys.stdin if inStr == "-" else open(inStr, "rU")
    outF = sys.stdout if outStr == "-" else open(outStr, "wb")
    if cont:
        inF2 = open(inStr2, "rU")

    try:
        inCSV = csv.DictReader(inF, dialect="excel")

        # Keep all incoming fields, and ensure that the ones we want
are included
        # for output
        inFields = inCSV.fieldnames
        outFields = sorted(list(methods)) # Convert back to list for
ordering

        # Format for this dictionary:
        #   method : (input column name, output column name)
        fieldOut = {} # dict mapping the methods to their output
columns

        for m in methods:# input col          output col
            fieldOut[m] = (m + inArchiveDate, m + 'Submit' + timeFmt)
if not cont else \
                            (m + inArchiveDate, m + cont)

        outFields = [f[1] for f in fieldOut.values()]
        outFields.extend(inFields)

        outCSV = csv.DictWriter(outF, outFields, dialect="excel")

        # Create and write a header row
        header = dict()
        for field in outFields:
            header[field] = field
        outCSV.writerow(header)

        # If continuing, read completed lines from the second input
file inF2.
        # This forwards the input pointer for inCSV to where we left
off.
        if cont:
            catchup(inCSV,outCSV,inF2, fieldOut)

        testUrls(inCSV, outCSV, fieldOut)
    except KeyboardInterrupt:
        print "Main caught Keyboard. Exiting..."
    finally:
        if inStr != "-":
            inF.close()
        if outStr != "-":
            outF.close()

    print "Total runtime: %d seconds. Finishing at" % (time.time() -
begin), time.ctime()
    print >>debugFile, "Total runtime: %d seconds. Finishing at" %
(time.time() - begin), time.ctime()

if __name__ == "__main__":
    main()
```

analysis/common_raw.R

```
#### Common commands for all analyses. Data is taken directly from CSV
output
#### by python program
#### Copyright 2013, Jason Hennessey. See README.txt for license.

# Check if the parent script set the directory
if (!"DIR_SET" %in% ls()) {
  setwd("/path/to/analysis/")
}

### Tunables
CACHE_FILENAME = "urls.Rdata"

WRITE_CACHE = FALSE  # Set to TRUE to (re)generate the cache file

READ_CACHE = TRUE
READ_CACHE = READ_CACHE & !WRITE_CACHE & file.exists(CACHE_FILENAME) #
Verify the cache is there


TOP_SIGCOUNT = 100 # Same tunable used for both domains and sources
SRC_SIGCOUNT = TOP_SIGCOUNT # Number of URLs a source (like a Journal
or Conference) must have to be significant
DOM_SIGCOUNT = TOP_SIGCOUNT # Number of URLs a domain must have to be
significant

YEAR_MIN = 1996 # Earliest year we want to analyze

# URLs columns to keep (can always add more later)
keep_columns = c("url","web", "web_pct",
"depth","PY96","LogTimesCited",
                 "Source_t", "TimesCited","dom","Dom_top30",
                 "ia", "wc", "archived",
                 "FundTextPresent","num_f", "Source_top20",
                 "Source_top30_t","Source_over100","DocType",
                 "PM96", "SC")

# If SHOW_THINKING not set by the parent script, initialize it here
if (! "SHOW_THINKING" %in% ls()) {
  SHOW_THINKING=FALSE
}

### Filter data as necessary
### 'urls_raw' contains the raw, unfiltered output of the python
programs
### 'urls' contains the screened variables that we want, one row per
instance of a URL being published
### 'uniq' contains one row per URL, with the appropriate variables
included.

if (READ_CACHE) {
  load(CACHE_FILENAME)
} else {
  ### Process urls_raw into urls.
  ## These next 3 lines are uncommented initially. After that, we use
"load"
  options(stringsAsFactors=FALSE)
  urls_raw = read.csv("urls.csv", header=T, stringsAsFactors=FALSE)
  #save(urls_raw, file="urls_raw.RData", compression_level=9)
  #load("urls_raw.RData") # Saves time if run instead of the above

  ## Elimination round: this section is where we remove URLs for QC
purposes.

  # Remove positive/negative controls
  urls = urls_raw[!(urls_raw$TI %in% c("Negative Control","Positive
Control")),] # Should be 7

  rm(urls_raw) # Not needed anymore

  # Only modelling years 1996-2010
  urls = urls[urls$PY %in% YEAR_MIN:2010,]

  # Webcitation detected some invalidly-parsed URLs.
  # Remove them from further analysis. n=18
  urls = urls[urls$wc2011.05.23 != "UnexpectedXML",]

  if (SHOW_THINKING) {
    # Examine journals where a large number of URLs in a journal are
from the same host
    # within a given year.
    # These types of URLs could represent things we're not after, such
as pointing to the
    # PDF version of a paper or promoting the website of the journal.
    # We set a minimum number of 20 URLs per journal to avoid small
sample size artifacts
    temp.journs = table(urls$SO)
    temp.journs = temp.journs[temp.journs >= 20] # filter journals with
low sample size
    temp.tophost = sapply(rownames(temp.journs),
                          function(x) {
                            temp.hosts = urls$host[urls$SO == x]
                            temp.all = length(temp.hosts)
                            temp.hosts = sort(table(temp.hosts),
decreasing=T)
                            return (temp.hosts[1] / temp.all) #
Percentage of total hosts accounted for by
                            # the most popular.
                          })
    names(temp.tophost) = rownames(temp.journs)
    # Journals with > 90% URLs from the same host. 90% is rather high,
but this shows
    # us that some journals with many URLs referring to the same host
are what the type
    # we are looking for (such as in the journal AMERICAN JOURNAL OF
NURSING, where the
    # links point to supplementary videos) while others are not what we
are looking for
    # (like MULTIPLE SCLEROSIS, where all point to the website for the
journal)
    for(i in names(temp.tophost[temp.tophost > .9])) {
      cat("Journal: ", i, "\n")
      print(urls$url[urls$SO == i])
    }
    # Look at distribution
    hist(temp.tophost)

    temp.journyears = data.frame(year=integer(), journ=character(),
                                 percent = numeric(), nTotal=integer(),
nTopUrl=integer(),
                                 topUrl=character())

    MIN_PER_YEAR = 10 # Minimum number of URLs a journal must publish
within a year to be considered

    temp.years = table(urls$SO, urls$PY)
    for(j in rownames(temp.journs)) {
      for (y in unique(urls$PY[urls$SO == j])) {
        temp.urls = urls$url[urls$SO == j & urls$PY == y]
        temp.all = length(temp.urls)
        if (temp.all < MIN_PER_YEAR) { # Skip ones that don't meet a
threshhold
          next
        }
        temp.urls = sort(table(temp.urls), decreasing=T)
        p = temp.urls[1] / temp.all
        entry = data.frame(year=y, journ=j, percent=p,
                           nTotal=temp.all,
                           nTopUrl=temp.urls[1],
                           names(temp.urls[1]))
        temp.journyears = rbind(temp.journyears, entry)
      }
    }
    rm(j,y,temp.urls,temp.all,p,entry, MIN_PER_YEAR)

    # For almost all journals, those which with high dups in one year
were high
    # for most years
    library(lattice)
    xyplot(percent ~ year, data=temp.journyears, type="l", group=journ)

    # .3 looks like a good cutoff

    temp.dups =  temp.journyears[temp.journyears$percent > .3,]
    xyplot(percent ~ year, data=temp.dups, type="l", group=journ)
  }

  # From the temp.dups list generated above, we identified journal/year
combos where
  # there are URLs that aren't the Internet-based academic tools we are
examining in this
  # study.
  # Many of them are journals pointing to their website.
  # Since supplementary information is of academic value, those URLs
were kept.
  # This step eliminates 943 URLs

  urls = urls[!(urls$url == "http://www.bjcancer.com" &
    urls$SO == "BRITISH JOURNAL OF CANCER" &
    urls$PY %in% 2000:2001),] # Journal website

  urls = urls[!(urls$url ==
"http://www3.interscience.wiley.com/journal/121548564/issueyear?year=20
09" &
    urls$SO == "BRITISH JOURNAL OF PHARMACOLOGY" &
    urls$PY == 2009),] # Advertising collection of papers

  urls = urls[!(urls$url == "http://dx.doi.org/10.1111/j.1476-
5381.2010.00831.x" &
    urls$SO == "BRITISH JOURNAL OF PHARMACOLOGY" &
    urls$PY == 2010),] # Advertising collection of papers

  urls = urls[!(urls$url == "http://www.circresaha.org" &
    urls$SO == "CIRCULATION RESEARCH" &
    urls$PY %in% 2000:2003),] # Journal website

  urls = urls[!(urls$url == "http://circres.ahajournals.org" &
    urls$SO == "CIRCULATION RESEARCH" &
    urls$PY == 2004),] # Journal website

  urls = urls[!(urls$url == "http://ctj.sagepub.com" &
    urls$SO == "CLINICAL TRIALS" &
    urls$PY %in% 2008:2010),] # Journal website

  urls = urls[!(urls$url ==
"http://cpc.cs.qub.ac.uk/licence/licence.htmlNo" &
    urls$SO == "COMPUTER PHYSICS COMMUNICATIONS" &
    urls$PY == 2008),] # Parsing mistake.

  urls = urls[!(urls$url == "http://www.jstage.jst.go.jp/browse/jpa2" &
    urls$SO == "JOURNAL OF PHYSIOLOGICAL ANTHROPOLOGY" &
    urls$PY == 2010),] # Journal website

  urls = urls[!(urls$url == "http://www.molmed.org" &
    urls$SO == "MOLECULAR MEDICINE" &
    urls$PY %in% 2009:2010),] # Journal website

  urls = urls[!(urls$url == "http://msj.sagepub.com" &
    urls$SO == "MULTIPLE SCLEROSIS" &
    urls$PY %in% 2007:2009),] # Journal website

  urls = urls[!(urls$url == "http://neuro-oncology.dukejournals.org" &
    urls$SO == "NEURO-ONCOLOGY" &
    urls$PY %in% 2008:2009),] # Journal website

  urls = urls[!(urls$url == "http://www.insp.mx/salud/index.html" &
    urls$SO == "SALUD PUBLICA DE MEXICO" &
    urls$PY %in% 2001:2004),] # Journal website; points to English-
version of papers.

  ## End of elimination

  ### Computed variables
  urls.len = length(urls$url)
  ## Create a best estimate of the date in PM96. We reflect this in
months
  ## since Jan 1, 1996
  monthList =
c("JAN","FEB","MAR","APR","MAY","JUN","JUL","AUG","SEP","OCT",
                "NOV","DEC")

  # By truncating at 3 chars, we round down for both ranges (like JUL-
AUG) and
  # for dates that values that have days (like JAN 1). No month
specified is
  # assumed to be January, since it could be an annual issue.
  #
  # These substitutions rely on journals publishing at the beginning of
a
  # given period.
  urls$PD2 = ifelse(urls$PD == "", "JAN", substr(urls$PD, 1, 3))
  seasonsList = c("WIN","SPR","SUM","FAL")
  seasonsListReplace = c("JAN","APR","JUL","OCT")

  # Convert PD2 into a numeric representing the number of the month
  urls$PD2 = sapply(urls$PD2,
                    function(x) {
                      # Convert seasons to a month
                      if (x %in% seasonsList) {
                        x = seasonsListReplace[which(x == seasonsList)]
                      }
                      # An assert to make sure all months are set to
something
                      stopifnot(x %in% monthList)
                      return (which(x == monthList) - 1)
                    })
  urls$PM96 = (urls$PY - YEAR_MIN)*12 + urls$PD2

  ## Calculate web column. Response >= 90% are considered present. <
90% is down.

  # These runs were outside of the study window. Eliminate them.
  urls$web2011.04.15.18.47 = NULL
  urls$web_reason2011.04.15.18.47 = NULL

  temp.web_cols = grep("web2011", colnames(urls)) # Column numbers
containing web avail
  temp.web_cnt  = rowSums(urls[,temp.web_cols] == 'True') # Tally the
'True' values
  temp.web_pct  = temp.web_cnt / max(temp.web_cnt) # Percentage
available

  urls$web = urls$web_pct = temp.web_pct
  urls$web = TRUE
  urls$web[urls$web_pct < .9] = FALSE
  rm(temp.web_cols, temp.web_cnt, temp.web_pct)

  ## Archive Engine Cleanup
  # Clean up the archive engine response columns by setting
  # their non-trues to falses
  temp.ia_cols = grep("ia20", colnames(urls), value=T)
  temp.wc_cols = grep("wc20", colnames(urls), value=T)
  urls.archive_cols = c(temp.ia_cols, temp.wc_cols)

  for(col in urls.archive_cols) {
    origColName = paste(col, "Orig", sep="")
    urls[,origColName] = urls[,col] # Make a backup of the original
statuses
    urls.archive_cols = c(urls.archive_cols, origColName) # Preserve
them for analysis

    # Recode everything to boolean
    urls[urls[,col] != "True", col] = "False"
    urls[,col] = as.logical(urls[,col])
  }

  temp.archDates = sub("ia","",temp.ia_cols)
  for(date in temp.archDates) {
    temp.archCol = paste("archived", date, sep="")
    temp.iaCol = paste("ia", date, sep="")
    temp.wcCol = paste("wc", date, sep="")

    urls[,temp.archCol] = (urls[,temp.wcCol] | urls[,temp.iaCol])
  }

  # Calculate a "SubmitFinal" column that is based on whether the
  # URL was submitted AND whether it tested positive in the next run.
  # The idea is that the return status alone from the submission isn't
enough
  # to gauge whether we freshly archived the site; if it showed up in
the subsequent
  # query then we know that it did.
  urls$iaSubmitFinal2012.11.15 = ifelse(urls$iaSubmit2012.11.15 !=
"Skipped" &
                                        urls$ia2013.02.05, TRUE, FALSE)
  urls$wcSubmitFinal2012.11.30 =
ifelse(grepl('^6',urls$wcSubmit2012.11.30) &
                                        urls$wc2013.02.05, TRUE, FALSE)

  # To capture the columns for pages submitted to the archive engines
  temp.submitCols = grep("Submit", colnames(urls), value=T)

  urls.archive_cols = c(urls.archive_cols, paste("archived",
temp.archDates, sep=""),
                        temp.submitCols)
  keep_columns = c(keep_columns, urls.archive_cols)
  rm(temp.ia_cols, temp.wc_cols, temp.archDates, temp.archCol,
temp.iaCol, temp.wcCol, temp.submitCols)

  ## Internet Archive
  ## For our purposes, use the archive snapshots that were taken right
after the web survey
  urls$ia = urls$ia2011.05.23

  ## WebCitation
  urls$wc = urls$wc2011.05.23

  ## Computed archived - whether a URL is archived in either system
  urls$archived = (urls$ia | urls$wc)

  ## Assess directory depth
  library(stringr)
  urls$depth = str_count(urls$url, "/") - 2 # Remove two due to the
initial "http://"

  # Subtract one if the URL ends with a "/"
  temp.lens = str_length(urls$url)
  temp.end_slash = (substr(urls$url, temp.lens, temp.lens) == "/")
  urls$depth[temp.end_slash] = urls$depth[temp.end_slash] - 1
  rm(temp.lens, temp.end_slash)

  ## Introduce PY96
  urls$PY96 = urls$PY - 1996

  ## Add TimesCited & Cited References
  urls$TimesCited = urls$TC
  urls$LogTimesCited = log2(urls$TC + 1)
  urls$CitesToOthers = urls$NR
  urls$LogCitesToOthers = log2(urls$NR + 1)

  ## Handle journal/source-related columns

  # It would have been nice to use the abbreviated variety (J9),
however
  # not every entry has it (347 are missing it) and some of the
journals that
  # are missing a J9 entry have it in other places, leading to
potential
  # misclassification (for example, SO="NUCLEIC ACIDS RESEARCH" URLs
published
  # in 1998 lack the J9 entry but other years have it)
  urls$Source = urls$SO

  if (SHOW_THINKING) {
    # Determine optimal truncation limit -- the lowest number of
characters while preserving
    # unique names
    length(unique(urls$SO)) # 3176
    for (i in 100:30) { print(cbind(i,length(unique(substr(urls$SO, 0,
i))))) } # It's 78, so use 80
  }

  # Create truncated Source column
  urls$Source_t = substr(urls$SO, 0, 80)

  # Calculate Source_top30_t
  temp.srcCounts = sort(table(urls$Source_t), decreasing=T)
  temp.srcSig = rownames(temp.srcCounts[1:30])
  temp.srcSig100 = rownames(temp.srcCounts[temp.srcCounts > 100])
  temp.srcSig20 = rownames(temp.srcCounts[temp.srcCounts >= 20])  # For
WC

  urls$Source_top30_t = ifelse(urls$Source_t %in% temp.srcSig,
urls$Source_t, "aaOTHER")
```

```
  urls$Source_over100 = ifelse(urls$Source_t %in% temp.srcSig100,
urls$Source_t, "aaOTHER")
  urls$Source_top20 = ifelse(urls$Source_t %in% temp.srcSig20,
urls$Source_t, "aaOTHER")
  urls$Source_top20 = factor(urls$Source_top20)

  rm(temp.srcSig100, temp.srcSig20)

  # For these sources, web is at or close to 100% either true or false
and/or
  # there are few unique urls.
  if (SHOW_THINKING) {
    # Show number of unique URLs by Source
    for (i in levels(urls$Source_top30_t)) {
      out = cat(i,length(which(urls$Source_top30_t == i)),
                length(unique(urls$url[urls$Source_top30_t == i])))
      print(out)
    }
    table(urls$web, urls$Source_top30_t)
  }

  # It's no longer necessary to remove these sources since we use the
uniq URLs for modelling.
  # Uncomment the next few commented lines in order to restore this
functionality.
  # temp.exclSources contains journals whereby almost all URLs are
duplicates.
  #   temp.exclSources = c("EPILEPSIA", "GENES CHROMOSOMES & CANCER")

  #   urls$Source_top30_t = ifelse(urls$Source_top30_t %in%
temp.exclSources, "aaOTHER", urls$Source_top30_t)
  urls$Source_top30_t = substr(urls$Source_top30_t, 0, 50) # top30
doesn't need as many chars to differentiate

  #   urls$Source_over100 = ifelse(urls$Source_over100 %in%
temp.exclSources, "aaOTHER", urls$Source_over100)

  urls$Source_over100 = factor(urls$Source_over100)
  urls$Source_top30_t = factor(urls$Source_top30_t)

  rm(temp.srcCounts, temp.srcSig)
  #     rm(temp.exclSources)
  # Obtain Dom_top30
  temp.domCounts = sort(table(urls$dom), decreasing=T)
  temp.sigDoms = rownames(temp.domCounts[1:30])
  temp.sigDoms = temp.sigDoms[temp.sigDoms != "mx"] # Remove .mx due to
being predominantly a single URL
  urls$Dom_top30 = factor(ifelse(urls$dom %in% temp.sigDoms, urls$dom,
"aaOTHER"))
  urls$dom = as.factor(urls$dom)    # recast as a factor
  rm(temp.domCounts, temp.sigDoms)

  # Generate FundTextPresent
  urls$FundTextPresent = str_length(urls$FX) > 0

  # Convert url_num to num_f  (number as factor)
  urls$num_f = factor(ifelse(urls$url_num < 3, urls$url_num, "3+"))
  urls$DocType = factor(urls$DT) # DocType

  # doSubjectAnalysis takes a urlList and returns a new one with a new
row
  # per subject. "chop" determines whether to truncate subjects with
commas
  # (such as "Psychology, Multidisciplinary") so that subspecialties
get
  # grouped into more general categories.
  doSubjectAnalysis = function(urlList, chop=FALSE) {
    # Subjects appear to be tied to the journal, not the article

    library(stringr)
    # For these purposes, URLs without subjects don't contribute, so we
remove them.
    urlList = subset(urlList, SC != "")

    # Duplicate each row; one for each subject contained in the SC
column
    subjs = str_split(urlList$SC, ";")
    subjs = lapply(subjs, str_trim)

    if (chop == TRUE) {
      subjs = lapply(subjs, sub, pattern="[[:space:]]*,.*$",
replacement="")
    }

    # Ensure only one entry per subject, per URL This is most pertinent
in cases
    # where we just truncated and there's multiple subjects for a given
URL
    subjs = lapply(subjs, unique)

    if (SHOW_THINKING) {
      subjs2 = unlist(subjs)
      subjCnt = sort(table(subjs2), decreasing=T) # Show unique number
of subjects
      subjCnt[1:20]

      # Look at more broad subjects by removing the text after a comma
      subjsGen = sub("[[:space:]]*,.*$", "", subjs2)
      subjsGenCnt = sort(table(subjsGen), decreasing=T)
      subjsGenCnt[1:20]
    }

    # Create empty list of proper length that will be populated shortly
    urlsExpanded = urlList[0,]
    urlsExpanded = urlsExpanded[1:length(unlist(subjs)),]

    begin = Sys.time()

    urlListLen = nrow(urlList)
    dest = 1
    for(src in 1:urlListLen) {
      subj = subjs[[src]]
      subjCnt = length(subj)
      for(j in 1:subjCnt) {
        urlsExpanded[dest,] = urlList[src,]
        urlsExpanded$SC[dest] = subj[j]
        dest = dest + 1
      }
    }
    cat("Parsing subjects took ", Sys.time() - begin, "\n")

    return(urlsExpanded)
  }

  # Generate urls list, expanded by having one entry per subject code.
  # We select keep_columns to reduce copy time (it adds up!).
  urlsSubjExp = doSubjectAnalysis(urls[,keep_columns], chop=TRUE)
  urlsSubjExp.len = length(urlsSubjExp$url)

  ### Unique URLs
  ### Build data frame with unique URLs and include variables
appropriate to a single URL
  uniq = data.frame(url = unique(urls$url))
  temp.join1 = match(uniq$url, urls$url)

  # Columns to transfer verbatum from urls to uniq. These should be the
same for every
  # uniq URL across all of the urls data frame entries pertaining to
it.
  uniq_cols = c("web", "web_pct", "depth", "dom", "archived", "ia",
"wc", urls.archive_cols)

  uniq[,uniq_cols] = urls[temp.join1,uniq_cols]

#   uniq$web = urls$web[temp.join1] # Was the URL available?
#   uniq$depth = urls$depth[temp.join1] # Domain depth
#   uniq$dom = urls$dom[temp.join1] # URL's domain
#   uniq$ia  = as.logical(urls$ia[temp.join1]) # Internet Archive
#   uniq$ia_new = urls$ia_new[temp.join1]
#   uniq$ia2011.05.23 = urls$ia2011.05.23[temp.join1]
#   uniq$wc  = as.logical(urls$wc[temp.join1]) # WebCitation
#   uniq$wc_new = as.logical(urls$wc_new[temp.join1])
#   uniq$wc2011.05.23 = urls$wc2011.05.23[temp.join1]
#   uniq$archived = uniq$ia | uniq$wc #  Computed column = archived by
either method

  ## Calculate number of times a URL has been published(similar to
Wren, 2008)
  ## Since a single journal article should only be able to publish a
URL once,
  ## we eliminate multiple URL entries for a single journal article by
matching
  ## the AB, AU, PY and SO fields.
  uniq$nPub = sapply(uniq$url,
                     function (x) {
                       unq = unique(urls[urls$url ==
x,c("AB","AU","PY","SO")])
                       return(nrow(unq))
                     })
  uniq$nJourns = sapply(uniq$url,
                        function (x) {
                          unq = unique(urls[urls$url == x,"SO"])
                          return(length(unq))
                        })

  uniq$LogPub = log2(uniq$nPub)

  if (SHOW_THINKING) {
    # How many URLs are published multiple times in a single article?
    temp.appears = table(urls$url)
    temp.join2 = match(uniq$url, rownames(temp.appears))
    uniq$nPubDups = temp.appears[temp.join2]
    rm(temp.appears, temp.join2)
    uniq[uniq$nPub != uniq$nPubDups, c("url","nPub","nPubDups")] # 7
urls
  }

  ## Average the Times Cited across all papers
  uniq$TimesCited = urls$TimesCited[temp.join1]
  # Create a list of URLs that appear multiple times. We will reuse
this.
  temp.multUrls = uniq$url[uniq$nPub > 1]
  temp.multUrlsCount = sapply(temp.multUrls,
                              function (x) {
                                mean(urls$TimesCited[urls$url == x],
na.rm=T)
                              })
  uniq$TimesCited[uniq$nPub > 1] = temp.multUrlsCount
  uniq$LogTimesCited = log2(uniq$TimesCited + 1)

  ## Calculate first/last seen
  ## It doesn't make sense to have a "published year" variable when
  ## there could be multiple...
  uniq$firstPY96 = uniq$lastPY96 = urls$PY96[temp.join1] # Published
Year, zero'd to 1996
  temp.first = sapply(temp.multUrls,
                      function (x) {
                        min(urls$PY96[urls$url == x])
                      })
  temp.last = sapply(temp.multUrls,
                     function (x) {
                      max(urls$PY96[urls$url == x])
                     })
  uniq$firstPY96[uniq$nPub > 1] = temp.first
  uniq$lastPY96[uniq$nPub > 1] = temp.last

  # Do it again for PM96
  uniq$firstPM96 = uniq$lastPM96 = urls$PM96[temp.join1] # Published
Year, zero'd to 1996
  temp.first = sapply(temp.multUrls,
                      function (x) {
                        min(urls$PM96[urls$url == x])
                      })
  temp.last = sapply(temp.multUrls,
                     function (x) {
                          max(urls$PM96[urls$url == x])
                        })
  uniq$firstPM96[uniq$nPub > 1] = temp.first
  uniq$lastPM96[uniq$nPub > 1] = temp.last

  rm(temp.first, temp.last)

  ## Import sources (journals)
  uniq$Source = urls$Source_t[temp.join1]

  temp.multUrlsSrc = sapply(temp.multUrls,
                             function (x) {
                               temp = unique(urls$Source_t[urls$url ==
x], na.rm=T)
                               if (length(temp) > 1) {
                                 # Cap concatenated strings at 100 chars
for readability
                                 concat = paste(sort(temp),
collapse="+")
                                 return(strtrim(concat, 100))
                               } else
                                 return(temp)
                             })
  uniq$Source[uniq$nPub > 1] = temp.multUrlsSrc

  if (SHOW_THINKING) {
    # Examine which URLs were published in multiple journals
    temp.multUrlsSrcCnt = sapply(temp.multUrls,
                                 function (x) {
                                   length(unique(urls$Source_t[urls$url
== x], na.rm=T))
                                 })
    # Interesting tidbit- of the minority (1145) of URLs published >
once,
    # most were published in just one or two journals
    densityplot(temp.multUrlsSrcCnt)
    table(temp.multUrlsSrcCnt)
    #  1   2   3   4   5   6   7   8   9  11  12  13  15  17  38
    #500 500  84  26  11   3   2   1   4   1   1   1   2   1   1

    # Which URLs were published in > 10 journals?
    uniq$url[uniq$nPub > 1][temp.multUrlsSrcCnt > 10]
    # [1] http://www.ncbi.nlm.nih.gov/     http://www.ncbi.nlm.nih.gov
http://imgt.cines.fr
    # [4] http://www.HaworthPress.com
http://www.clinicaltrials.gov    http://clinicaltrials.gov
    # [7] http://www.controlled-trials.com
    # FYI - http://www.clinicaltrials.gov was the one in 38 journals
    rm(temp.multUrlsSrcCnt)
  }

  # Calculate Source_top. We set the threshhold for being included
  # as publishing > 100 unique URLs
  temp.srcCounts = sort(table(uniq$Source), decreasing=T)
  temp.srcSig = rownames(temp.srcCounts[temp.srcCounts > SRC_SIGCOUNT])
  uniq$Source_top = ifelse(uniq$Source %in% temp.srcSig, uniq$Source,
"aaOTHER")
```

```
  uniq$Source_top = factor(uniq$Source_top)

  # Those with >= 20 for the purposes of figuring which journals use WC
  temp.srcSig = rownames(temp.srcCounts[temp.srcCounts >= 20])
  uniq$Source_top20 = ifelse(uniq$Source %in% temp.srcSig, uniq$Source,
"aaOTHER")
  uniq$Source_top20 = factor(uniq$Source_top20)

  rm(temp.srcCounts, temp.srcSig)

  ## Calculate the combined FundTextPresent value as a number between 0
and 1
  ## 0 = FALSE, 1 = TRUE
  ## For those where the answer is not 0 or 1 (most URLs only
  ## appear once and most of the repeats do not have differing
  ## values), we use the percentage appearing TRUE
  uniq$FundTextPresent = ifelse(urls$FundTextPresent[temp.join1], 1, 0)

  temp.multFundText = sapply(temp.multUrls,
                             function (x) {
                               texts = urls$FundTextPresent[urls$url ==
x]
                               total = length(texts)
                               return (length(texts[texts ==
TRUE])/total)
                             })

  uniq$FundTextPresent[uniq$nPub > 1] = temp.multFundText

  rm(temp.multFundText)

  if (SHOW_THINKING) {
    # Look at FundTextPresent status
    temp.FundTextDiffers = sapply(temp.multUrls,
                                  function (x) {
                                    temp.fundtexts =
urls$FundTextPresent[urls$url == x]
                                    return
(ifelse(length(unique(temp.fundtexts)) > 1, TRUE, FALSE))
                                  })
    table(temp.FundTextDiffers)
    # FALSE  TRUE
    # 855   283
  }
  rm(temp.join1, temp.multUrls, temp.multUrlsCount, temp.multUrlsSrc)

  # Determine significant domains
  if (SHOW_THINKING) {
    # Look at the top domains
    sort(table(uniq$dom), decreasing = T)[1:30]
  }

  temp.domCounts = sort(table(uniq$dom), decreasing=T)
  temp.sigDoms = rownames(temp.domCounts[temp.domCounts >
DOM_SIGCOUNT])
  uniq$domSig = as.character(uniq$dom)
  uniq$domSig[!(uniq$dom %in% temp.sigDoms)] = "aaOTHER"
  uniq$domSig = factor(uniq$domSig)

  uniq.len = length(uniq$url)

  rm(temp.domCounts, temp.sigDoms)

  # Only keep the columns we need
  urls = urls[,keep_columns]

  if (WRITE_CACHE) {
    # cols =
c("web","depth","PY96","LogTimesCited","J9","TimesCited","dom","ia","wc
")
    # write.csv(urls[,cols], file=CACHE_FILENAME)
    save(urls, urls.len, urlsSubjExp, urlsSubjExp.len,
urls.archive_cols, uniq, uniq.len, compression_level=9,
file=CACHE_FILENAME)
  } # WRITE_CACHE

} # READ_CACHE
```

analysis/output.txt

```
> source('/path/to/analysis/stats.r')
Number of URLs (unique):  17110 ( 14489 )
Web overall explained deviance: 2019
IA overall explained deviance: 2545
WC overall explained deviance: 2651
[1] "Deviance explained by each predictor"
                       web         ia          wc
Model Dev       2018.53454 2545.20629 2650.817959
Unique Dev      1357.82396 1505.95474 2348.088062
lastPM96         570.76149  149.67841   93.416797
LogPub            32.21306   48.30516  185.740691
depth            196.50737  677.45375  338.336658
LogTimesCited     17.68087   35.35097  404.633766
FundTextPresent   48.11457  203.52509    2.654017
domSig           355.29157  225.58620  173.710916
Source_top       137.25503  166.05516 1149.595217
[1] "Median survival times for URLs"
Call: survfit(formula = urls.survFormM ~ 1, data = urls)

records   n.max n.start  events  median 0.95LCL 0.95UCL
  17110   17110   17110    4356     112     112     120
[1] "Median survival times for unique URLs"
Call: survfit(formula = uniq.survFormM ~ 1, data = uniq)

records   n.max n.start  events  median 0.95LCL 0.95UCL
  14489   14489   14489    3266     112     112     112
Percent URLs (unique) dead:  31 ( 33 )
Percent URLs (unique) alive:  69 ( 67 )
Percent URLs (unique) in IA:  62 ( 59 )
Percent URLs (unique) in WC:  21 ( 16 )
Percent URLs (unique) archived:  65 ( 62 )
Percent URLs (unique) available in some manner:  84 ( 82 )
Percent of missing URLs (unique) archived: 49 ( 47 )
Percent of missing URLs (unique) archived by IA: 47 ( 46 )
Percent of missing URLs (unique) archived by WC: 7 ( 6 )
Number of uniq URLs submitted to archiving engines: IA 1163 , WC 7285
Number of DOI sites (unique): 189 ( 167 )
Number of PURL sites (unique): 9 ( 8 )
Living URLs published > 1 times living and missing: 0.7874225 0.2125775
Living URLs published 1 time living and missing: 0.6560629 0.3439371
Internet Archive sites that were blocked from archiving due to
robots.txt (uniq): 507, 2.963179% (352, 2.429429%)
Unavailable Internet Archive sites that were blocked from archiving due
to robots.txt (uniq): 81, 15.97633% (76, 21.59091%)
Survival Times for subject  Biochemistry & Molecular Biology
Survival Times for subject  Biotechnology & Applied Microbiology
Survival Times for subject  Computer Science
Survival Times for subject  Biochemical Research Methods
Survival Times for subject  Mathematical & Computational Biology
Survival Times for subject  Genetics & Heredity
Survival Times for subject  Physics
Survival Times for subject  Engineering
Survival Times for subject  Statistics & Probability
Survival Times for subject  Chemistry
Survival Times for subject  Biophysics
Survival Times for subject  Astronomy & Astrophysics
Survival Times for subject  Mathematics
Survival Times for subject  Zoology
Survival Times for subject  Cell Biology
Survival Times for subject  Biology
Survival Times for subject  Oncology
Survival Times for subject  Plant Sciences
Survival Times for subject  Environmental Sciences
Survival Times for subject  Medicine
Time difference of 2.76343 mins
Linear coefficients (R^2) for URLs percentage by year overall:
0.03664289 ( 95.50238 %)
Uniq Percent increase for IA ( 11356 - 9276 = 2080 ): 22.42346
Uniq Percent increase for WC ( 8842 - 2494 = 6348 ): 254.5309
URLs submitted to IA (unique): 3039 ( 2662 )
URLs submitted to WC (unique): 8486 ( 7477 )
URLs submitted to IA which returned error but were successfully
archived: 872
URLs submitted to WC which returned error but were successfully
archived: 12
URLs submitted to WC which returned success but were unsuccessfully
archived: 955
URL count (percent) whose availability was > 0 or < .9: 466 (
0.03216233 )
Variation (max-min) between 1996 and 1999, inclusive (uniq): 0.02356403
( 0.01461575 )
R squared for 1996-1999 linear fit (unique): 0.5132796 ( 0.1808213 )
Variation (max-min) between 2000 and 2010, inclusive (uniq): 0.4264276
( 0.4107151 )
R squared for 2000-2010 linear fit (unique): 0.9479806 ( 0.9457182 )
URLs appearing more than once: 1129 or 0.07792118 %
Multiply published URLs only published in 1 journal: 0.4348981 %
Funding text in multiply published URLs is different 0.2444641 % of the
time
Overall elapsed time: 18.01948
```

analysis/stats.R

```
#### Survival Analysis
#### We construct two models -
####  urls.surv: every published URL is an entry
####  uniq.surv: every unique URL is an entry
#### Data is from the raw output of the python scripts
#### and has not been otherwise modified.
#### Copyright 2013, Jason Hennessey. See README.txt for license.


# Enable to follow logic used while determining the final model
SHOW_THINKING = FALSE
DIR_SET = TRUE

setwd("/path/to/analysis/")
temp.begin = Sys.time()
source("common_raw.R", echo=F) # Load data and transforms
Sys.time() - temp.begin
library(survival)

methods = c("web","ia","wc")

### Overall statistics
cat("Number of URLs (unique): ", length(urls$url), "(",
length(uniq$url), ")\n")

### Custom functions

# A rough AIC. The coefficient penalty could be better, but none
# of the best model candidates are close enough to make it matter.
survAIC = function (x) {
      -2*x$loglik[2]+2*(length(x$coef)-1)
}

installLibs = function() {
  # List of packages to install
  packages = c("HH", "stringr", "Hmisc", "lattice")
  install.packages(pkgs=packages)
}

loadLibs = function() {
  # Load the libraries necessary for other things. Used as a
convenience
  # function during development for when we load our variables from a
workspace
  # (which doesn't load the libraries too)
  packages = c("HH", "stringr", "Hmisc", "lattice")
  for (p in packages) {
    library(p, character.only=TRUE)
  }
}
## Loop through survreg() distributions to identify (and return) the
best

compareSurvregAIC = function(inFormula, inData) {
	## Compute Full Model and compare AIC values.
	print("Comparing AIC for formula")
	print(inFormula)
	dists = c("weibull", "exponential", "gaussian", "logistic",
"lognormal", "loglogistic")
	aic = NULL
	bestAIC = Inf
	bestSurv = NULL
	for(d in dists) {
		surv = survreg(inFormula, data=inData,dist=d)
		thisAIC = survAIC(surv)
		aic = append(aic, thisAIC)
		if (thisAIC < bestAIC) {
			bestSurv = surv
			bestAIC = thisAIC
		}
	}
	vals = data.frame(dists,aic)
	vals = vals[order(vals$aic),]
	rownames(vals) = NULL
	return(vals)
}

### Survival Transforms

# Number of years from the beginning (1996) until when the sample was
taken
# (2011)
# Used for lifetime survival calculations. Specified when the
measurements were
# taken.
LIFE_DIFF = 15
LIFE_DIFFM = 15*12 + 4

# Create web inverse indicator where TRUE means DEAD and FALSE means
ALIVE
# ALIVE is a synonym for "right-censored" in survival parlance
urls$dead = !urls$web
urlsSubjExp$dead = !urlsSubjExp$web

# For event, 0=right censored, 1=event at ?time?, 2=left censored,
3=interval censored
# Therefore, if URL alive then event=0, otherwise 2
urls$event = rep.int(0,urls.len)
urls$event[urls$dead] = 2

urlsSubjExp$event = rep.int(0,urlsSubjExp.len)
urlsSubjExp$event[urlsSubjExp$dead] = 2

# Use -Inf as the second arg, since it should be ignored due to
# event always being 0 or 2. This gives us a verification of this
behavior,
# since if it weren't ignored, we would hopefully see some errors due
to mismatched
# vector size or out of bounds.
urls.survForm = Surv(LIFE_DIFF - urls$PY96, -Inf, urls$event,
type="interval")
```

```
urls.survFormM = Surv(LIFE_DIFF*12 - urls$PM96, -Inf, urls$event,
type="interval")


### Model building - urls

# We use the uniq entries for the survival model due to model
assumptions of
# independence between observations. Were we using the non-deduplicated
urls for
# the model, certain variables (like the outcome variables, domain,
etc) would
# be the same across instances of a particular URL.
URLS_SURV_MODEL = FALSE
if (URLS_SURV_MODEL) {
    urls.form1 = formula("urls.survForm ~ DocType + Dom_top30 +
FundTextPresent +
                        num_f + Source_top30_t + LogTimesCited +
depth")
    urls.form2 = formula("urls.survForm ~ DocType + Dom_top30 +
FundTextPresent +
                        num_f + Source_over100 + LogTimesCited +
depth")
    if (SHOW_THINKING) {
      ## Compare using the top 30 sources vs. using those that have >
100 URLs
      compareSurvregAIC(urls.form1, urls) # top30
      compareSurvregAIC(urls.form2, urls) # over100
    }

    urls.fullForm = urls.form2
    urls.survFull = survreg(urls.fullForm, data=urls, dist="gaussian")

    if (SHOW_THINKING) {
      ## Correlation exists between CitesToOthers and TimesCited
      cor.test(urls$CitesToOthers,urls$TimesCited, use="complete.obs")
      cor.test(urls$CitesToOthers,urls$TimesCited, use="complete.obs",
method="spearman")
      # pearson = .0053, spearman ~ 0. High correlation

      # Check p value for single model with just those variables to see
which is better
      summary(survreg(urls.survForm ~ CitesToOthers, data=urls)) # .456
      summary(survreg(urls.survForm ~ LogCitesToOthers, data=urls)) #
.0369
      summary(survreg(urls.survForm ~ TimesCited, data=urls)) # .000125
      summary(survreg(urls.survForm ~ LogTimesCited, data=urls)) #
3.31e-27

      # We will model without CitesToOthers due to collinearity
      # and use LogTimesCited

      # Look for the best distribution
      compareSurvregAIC(urls.fullForm, urls)
      summary(urls.survFull)
    }

    ## Compute Reduced Model
    # Use AICs to screen for other variables to remove

    if (SHOW_THINKING) {
      urls.surv1 = step(urls.survFull)     # Shows that removing DocType
increases
                                                              # AIC only
marginally (~ 1).
                                                              # Since it
is marginal and not central to study, remove.
                                                              # This
step() call doesn't actually remove anything.
    }

    urls.form2 = formula("urls.survForm ~ Dom_top30 + FundTextPresent +
num_f +
                              Source_top30_t + LogTimesCited")
    urls.surv2 = survreg(urls.form2, data=urls, dist="logistic")

    if (SHOW_THINKING) {
      compareSurvregAIC(urls.form2,urls)
      rm(temp.notused)
    }

    # Final models
    urls.surv = urls.surv2

    if (SHOW_THINKING) {
      anova(urls.surv) # Everything has p < .001
      urls.survTable = summary(urls.surv)$table
      urls.survInsignificant = urls.survTable[urls.survTable[,"p"] >
.001,]
      urls.survInsignificant
      urls.survSignificant = urls.survTable[urls.survTable[,"p"] <=
.001,]
      urls.survSignificant
    }
} # if URLS_SURV_MODEL

### Survival model building - uniq

uniq$dead = !uniq$web
uniq$event = rep.int(0,uniq.len)
uniq$event[uniq$dead] = 2

# For the uniq URLs, we have a different scenario: some URLs have had
# multiple publishings. In those cases, we assume the URL was
functional
# from the first published date through the last.
# Thus, for living URLs use the first published date and dead ones use
# the last (all are relative to 2011).
uniq$survAge = LIFE_DIFF - ifelse(uniq$dead, uniq$lastPY96,
uniq$firstPY96)

# Models based on months are appended with an M
```

```
uniq$survAgeM = LIFE_DIFF*12 - ifelse(uniq$dead, uniq$lastPM96,
uniq$firstPM96)

uniq.survForm = Surv(uniq$survAge, sample(uniq.len), uniq$event,
type="interval")
uniq.survFormM = Surv(uniq$survAgeM, sample(uniq.len), uniq$event,
type="interval")

UNIQ_SURV_MODEL = TRUE
if (UNIQ_SURV_MODEL) {
    uniq.Surv = Surv(uniq$survAge, uniq$dead)
    uniq.SurvM = Surv(uniq$survAgeM, uniq$dead)
    uniq.coxph = coxph(uniq.Surv ~ LogPub + depth + LogTimesCited +
FundTextPresent + domSig +
        Source_top, data=uniq)

    if (SHOW_THINKING) {
      # Compare using nPub to using log2(nPub)
      uniq.surv1Form = formula("uniq.survFormM ~ domSig + nPub + depth
+ LogTimesCited + Source_top")
      compareSurvregAIC(uniq.surv1Form, uniq)
      uniq.surv1 = survreg(uniq.surv1Form, data=uniq, dist="logistic")
      uniq.surv2Form = formula("uniq.survFormM ~ domSig + LogPub +
depth + LogTimesCited + Source_top")
      compareSurvregAIC(uniq.surv2Form, uniq)  # logistic the best for
both
      uniq.surv2 = survreg(uniq.surv2Form, data=uniq, dist="logistic")
      anova(uniq.surv1, uniq.surv2)  # Compare models - LogPub is much
better!
    }

    if (SHOW_THINKING) {
      # Look for collinearity between predictors using Variance
Inflation Factor
      library(HH)
      temp.vif = vif(web ~ domSig + LogPub + depth + LogTimesCited +
Source_top, data=uniq)
      length(temp.vif[temp.vif > 5]) # None
      rm(temp.vif)
    }

    uniq.fullForm = formula("uniq.survForm ~ LogPub + depth +
LogTimesCited + FundTextPresent + domSig +
                        Source_top")
    uniq.fullFormM = formula("uniq.survFormM ~ LogPub + depth +
LogTimesCited + FundTextPresent + domSig +
                        Source_top")

    #uniq.survFull = survreg(uniq.fullForm, data=uniq, dist="logistic")
    uniq.survFullM = survreg(uniq.fullFormM, data=uniq,
dist="logistic")

    ### Survival Regression

    if (SHOW_THINKING) {
      ## Compare AIC values between using the year-based method and the
more precise month-based one.
```

```
      compareSurvregAIC(uniq.fullForm, uniq)
      summary(uniq.survFullM)
      compareSurvregAIC(uniq.fullFormM, uniq)
      summary(uniq.survFullM)
    }

    if (SHOW_THINKING) {
      # Look for non-significant variables
      uniq.surv1 = step(uniq.survFullM) # This step() call doesn't
actually remove anything.
    }
    #uniq.surv = uniq.survFull
    uniq.survM = uniq.survFullM

    # Examine significant variables
    ALPHA = .001
    anova(uniq.survM) # Everything has p < ALPHA
    uniq.survTableM = summary(uniq.survM)$table
    uniq.survInsignificantM = uniq.survTableM[uniq.survTableM[,"p"] >
ALPHA,]
    uniq.survInsignificantM
    uniq.survSignificantM = uniq.survTableM[uniq.survTableM[,"p"] <=
ALPHA,]
    uniq.survSignificantM

    # 95% confidence intervals
    uniq.confints = confint(uniq.survM, level=.9)

    # New table containing coefs, std error, z, p and 95% conf ints
    # Need to leave off last row (log(scale)) due to no conf int

    uniq.survTableConfM =
cbind(uniq.survTableM[1:(nrow(uniq.survTableM)-1),],
                        uniq.confints)

    uniq.survTableConf = uniq.survTableConfM

    # Convert to years instead of months
    uniq.survTableConf[,c("Value","5 %", "95 %")] =
uniq.survTableConf[,c("Value","5 %", "95 %")]/12

    if (SHOW_THINKING) {
      # Combine confidence intervals
      # Write results to file
      write.csv(uniq.survTableConfM, file="uniq_sigvarsM.csv")
      write.csv(uniq.survTableConf, file="uniq_sigvars.csv")
    }
    ### Analysis of the model

    if (SHOW_THINKING) {
      # Show statistics for different domains and journals
      uniq.survFitDom = survfit(uniq.survFormM ~ domSig, data=uniq)
      uniq.survFitSrc = survfit(uniq.survFormM ~ Source_top, data=uniq)

      # Overall survival graph
      plot(uniq.survFitM)
      plot(urls.survFitM)
      # A logistic regression similar to the survival for comparison
purposes
      urls.webLogSurvit = glm(web ~ firstPY96 + LogPub + depth +
LogTimesCited + domSig + Source_top,
                              family=binomial("logit"), data=uniq)
    }

    ### Test assumptions of logistic survival regression by using the
model to predict
    testSurvival = function() {
        # First row is the default -- should be equal to intercept
        # Second row is same as default except domain=au
        # Third row is a less likely paper (LogTimesCited=1, depth=3,
LogPub=1,domSig=au)
        # Fourth row is a popular paper.
        # Fifth row is the example used in the paper
        predictData =
uniq[1:5,c("domSig","LogPub","depth","LogTimesCited","Source_top",
"FundTextPresent")]
        predictData[,"domSig"] = c("aaOTHER","au","au", "org", "au")
        predictData[,"LogPub"] = c(0,0,1,3,0)
        predictData[,"depth"] = c(0,0,3,0,0)
        predictData[,"LogTimesCited"] = c(0,0,1,7,1)
        predictData[,"Source_top"] =
c("aaOTHER","aaOTHER","aaOTHER","BMC BIOINFORMATICS","aaOTHER")
        predictData[,"FundTextPresent"] = c(0,0,0,1,1)

        # Let's test it
        uniq.survM.pred = predict(uniq.survM, newdata=predictData,
type="response")


        if (SHOW_THINKING) {
            ## Show the logistic curves for the hazard ratios
            temp.range1=-10:50

            plot(temp.range1, dsurvreg(temp.range1, uniq.survM.pred[1],
uniq.survM$scale, dist=uniq.survM$dist), type="l")  # 1
            lines(temp.range1, dsurvreg(temp.range1,
uniq.survM.pred[2], uniq.survM$scale, dist=uniq.survM$dist),
col="green") # 2
            lines(temp.range1, dsurvreg(temp.range1,
uniq.survM.pred[3], uniq.survM$scale, dist=uniq.survM$dist),
col="blue") # 3
            lines(temp.range1, dsurvreg(temp.range1,
uniq.survM.pred[4], uniq.survM$scale, dist=uniq.survM$dist),
col="orange") # 4

            ## Show logistic curves for the cumulative distribution
            plot(temp.range1, psurvreg(temp.range1, uniq.survM.pred[1],
uniq.survM$scale, dist=uniq.survM$dist), type="l") # 1
            lines(temp.range1, psurvreg(temp.range1,
uniq.survM.pred[2], uniq.survM$scale, dist=uniq.survM$dist),
col="green") # 2
```

```
            lines(temp.range1, psurvreg(temp.range1,
uniq.survM.pred[3], uniq.survM$scale, dist=uniq.survM$dist),
col="blue") # 3
            lines(temp.range1, psurvreg(temp.range1,
uniq.survM.pred[4], uniq.survM$scale, dist=uniq.survM$dist),
col="orange") # 4


            temp.range2=0:100/100
            ## Show logistic curves for the cumulative distribution
            plot(temp.range2, qsurvreg(temp.range2, uniq.survM.pred[1],
uniq.survM$scale, dist=uniq.survM$dist), ylim=c(-30,50), type="l") # 1
            lines(temp.range2, qsurvreg(temp.range2,
uniq.survM.pred[2], uniq.survM$scale, dist=uniq.survM$dist),
col="green") # 2
            lines(temp.range2, qsurvreg(temp.range2,
uniq.survM.pred[3], uniq.survM$scale, dist=uniq.survM$dist),
col="blue") # 3
            lines(temp.range2, qsurvreg(temp.range2,
uniq.survM.pred[4], uniq.survM$scale, dist=uniq.survM$dist),
col="orange") # 4

            ## This should be the same as above
            plot(temp.range2, predict(uniq.survM,
newdata=predictData[1,], type="quantile", p=temp.range2), ylim=c(-
30,50), type="l")
            lines(temp.range2, predict(uniq.survM,
newdata=predictData[2,], type="quantile", p=temp.range2), col="green",
type="l")
            lines(temp.range2, predict(uniq.survM,
newdata=predictData[3,], type="quantile", p=temp.range2), col="blue",
type="l")
            lines(temp.range2, predict(uniq.survM,
newdata=predictData[4,], type="quantile", p=temp.range2), col="orange",
type="l")

            rm(temp.range1,temp.range2)
        }

    }
    testSurvivalM = function() {
      # First row is the default -- should be equal to intercept
      # Second row is same as default except domain=au
      # Third row is a less likely paper (LogTimesCited=1, depth=3,
LogPub=1,domSig=au)
      # Fourth row is a popular paper.
      # Fifth row is the example used in the paper
      predictData =
uniq[1:5,c("domSig","LogPub","depth","LogTimesCited","Source_top",
"FundTextPresent")]
      predictData[,"domSig"] = c("aaOTHER","au","au", "org", "au")
      predictData[,"LogPub"] = c(0,0,1,3,0)
      predictData[,"depth"] = c(0,0,3,0,0)
      predictData[,"LogTimesCited"] = c(0,0,1,7,1)
      predictData[,"Source_top"] = c("aaOTHER","aaOTHER","aaOTHER","BMC
BIOINFORMATICS","aaOTHER")
      predictData[,"FundTextPresent"] = c(0,0,0,1,1)
      # Let's test it
      uniq.survM.pred = predict(uniq.survM, newdata=predictData,
type="response")

      if (SHOW_THINKING) {
        ## Show the logistic curves for the hazard ratios
        temp.range1=-10:50

        plot(temp.range1, dsurvreg(temp.range1, uniq.survM.pred[1],
uniq.survM$scale, dist=uniq.survM$dist), type="l")  # 1
        lines(temp.range1, dsurvreg(temp.range1, uniq.survM.pred[2],
uniq.survM$scale, dist=uniq.survM$dist), col="green") # 2
        lines(temp.range1, dsurvreg(temp.range1, uniq.survM.pred[3],
uniq.survM$scale, dist=uniq.survM$dist), col="blue") # 3
        lines(temp.range1, dsurvreg(temp.range1, uniq.survM.pred[4],
uniq.survM$scale, dist=uniq.survM$dist), col="orange") # 4

        ## Show logistic curves for the cumulative distribution
        plot(temp.range1, psurvreg(temp.range1, uniq.survM.pred[1],
uniq.survM$scale, dist=uniq.survM$dist), type="l") # 1
        lines(temp.range1, psurvreg(temp.range1, uniq.survM.pred[2],
uniq.survM$scale, dist=uniq.survM$dist), col="green") # 2
        lines(temp.range1, psurvreg(temp.range1, uniq.survM.pred[3],
uniq.survM$scale, dist=uniq.survM$dist), col="blue") # 3
        lines(temp.range1, psurvreg(temp.range1, uniq.survM.pred[4],
uniq.survM$scale, dist=uniq.survM$dist), col="orange") # 4


        temp.range2=0:100/100
        ## Show logistic curves for the cumulative distribution
        plot(temp.range2, qsurvreg(temp.range2, uniq.survM.pred[1],
uniq.survM$scale, dist=uniq.survM$dist), ylim=c(-30,50), type="l") # 1
        lines(temp.range2, qsurvreg(temp.range2, uniq.survM.pred[2],
uniq.survM$scale, dist=uniq.survM$dist), col="green") # 2
        lines(temp.range2, qsurvreg(temp.range2, uniq.survM.pred[3],
uniq.survM$scale, dist=uniq.survM$dist), col="blue") # 3
        lines(temp.range2, qsurvreg(temp.range2, uniq.survM.pred[4],
uniq.survM$scale, dist=uniq.survM$dist), col="orange") # 4

        ## This should be the same as above
        plot(temp.range2, predict(uniq.survM, newdata=predictData[1,],
type="quantile", p=temp.range2), ylim=c(-30,250), type="l")
        lines(temp.range2, predict(uniq.survM, newdata=predictData[2,],
type="quantile", p=temp.range2), col="green", type="l")
        lines(temp.range2, predict(uniq.survM, newdata=predictData[3,],
type="quantile", p=temp.range2), col="blue", type="l")
        lines(temp.range2, predict(uniq.survM, newdata=predictData[4,],
type="quantile", p=temp.range2), col="orange", type="l")

        rm(temp.range1,temp.range2)

        predict(uniq.survM, newdata=predictData[1,], type="quantile",
p=.5) # Predicts median for default
        predict(uniq.survM, newdata=predictData[2,], type="quantile",
p=.5) # domain AU
```

```
      }
    }
} # if UNIQ_SURV_MODEL

### IA

if (SHOW_THINKING) {
      # IA works better using the first published date
      temp.ia = glm(ia ~ firstPY96 + LogPub + depth + LogTimesCited +
FundTextPresent + domSig + Source_top,
                        family=binomial("logit"), data=uniq)
      AIC(temp.ia)
      temp.ia = glm(ia ~ lastPY96 + LogPub + depth + LogTimesCited +
FundTextPresent + domSig + Source_top,
                        family=binomial("logit"), data=uniq)
      AIC(temp.ia)
      rm(temp.ia)
}

uniq.ia = glm(ia ~ firstPY96 + LogPub + depth + LogTimesCited +
FundTextPresent + domSig + Source_top,
                              family=binomial("logit"), data=uniq)
if (SHOW_THINKING) {
      uniq.ia2 = glm(ia ~ firstPY96*(LogPub + depth + LogTimesCited +
FundTextPresent + domSig + Source_top),
                        family=binomial("logit"), data=uniq)
      anova(uniq.ia2, test="Chisq") # LogPub and depth insignificant,
so  drop.
      uniq.ia3 = glm(ia ~ firstPY96*(LogTimesCited + FundTextPresent +
domSig + Source_top) + LogPub + depth,
                        family=binomial("logit"), data=uniq)
      anova(uniq.ia3, test="Chisq")
      # Check for confounders
      uniq.ia3vif = vif(uniq.ia3)
      uniq.ia3vif[uniq.ia3vif > 10]

      uniq.ia4 = glm(ia ~ firstPY96*(LogTimesCited + domSig +
Source_top) + LogPub + depth + FundTextPresent,
                        family=binomial("logit"), data=uniq)
      uniq.ia4vif = vif(uniq.ia4)
      uniq.ia4vif[uniq.ia4vif > 10]

}
if (SHOW_THINKING) {
      anova(uniq.ia)
      uniq.ia.vif = vif(uniq.ia)
}


### Webcite

if (SHOW_THINKING) {
      # WC works better using the last published date
      temp.wc = glm(wc ~ firstPY96 + LogPub + depth + LogTimesCited +
FundTextPresent + domSig + Source_top,
                        family=binomial("logit"), data=uniq)
      AIC(temp.wc)
      temp.wc = glm(wc ~ lastPY96 + LogPub + depth + LogTimesCited +
FundTextPresent + domSig + Source_top,
                        family=binomial("logit"), data=uniq)
      AIC(temp.wc)
      rm(temp.wc)
}

uniq.wc = glm(wc ~ lastPY96 + LogPub + depth + LogTimesCited +
FundTextPresent + domSig + Source_top,
                              family=binomial("logit"), data=uniq)

### Both

if (SHOW_THINKING) {
      # archived works better using the first published date
      temp.archived = glm(archived ~ firstPY96 + LogPub + depth +
LogTimesCited + FundTextPresent + domSig + Source_top,
                        family=binomial("logit"), data=uniq)
      AIC(temp.archived)
      temp.archived = glm(archived ~ lastPY96 + LogPub + depth +
LogTimesCited + FundTextPresent + domSig + Source_top,
                        family=binomial("logit"), data=uniq)
      AIC(temp.archived)
      rm(temp.archived)
}

uniq.archForm = as.formula("archived ~ firstPY96 + LogPub + depth +
LogTimesCited + FundTextPresent + domSig + Source_top")
uniq.archived = glm(uniq.archForm, family=binomial("logit"), data=uniq)

if (SHOW_THINKING) {
  eval_model = function(inModel, inData) {
    require(HH)
    print(AIC(inModel))
    temp.vif = vif(inModel)
    print("Finished vif")
    print(temp.vif[temp.vif > 5])
    print(anova(inModel, test="Chisq"))
  }

      eval_model(uniq.archived, uniq) # The basic model. AIC 16447

      # Look for interactions
      temp.archived = glm(archived ~ firstPY96*(LogTimesCited + LogPub
+ depth  + FundTextPresent + Source_top + domSig),
                        family=binomial("logit"), data=uniq)
      eval_model(temp.archived) # Too much collinearity -- overfit

      # Remove interactions between firstPY96 and (LogPub and depth)
due to low Chisq tests
      temp.archived2 = glm(archived ~ firstPY96*(LogTimesCited +
FundTextPresent + Source_top + domSig) + LogPub + depth,
                        family=binomial("logit"), data=uniq)
      eval_model(temp.archived2) # Still too much collinearity

      # Remove Source interactions, since some vif values are highest
there
```

```
      temp.archived3 = glm(archived ~ firstPY96*(LogTimesCited +
FundTextPresent + domSig) + LogPub + depth + Source_top,
                        family=binomial("logit"), data=uniq)
      eval_model(temp.archived3) #


      temp.archived2 = glm(archived ~ firstPY96 +
firstPY96:LogTimesCited + LogPub + depth  + FundTextPresent +
Source_top + domSig,
                        family=binomial("logit"), data=uniq)
      AIC(temp.archived2)
      anova(temp.archived2, test="Chisq")

      require(HH)
      temp.archived2Vif = vif(temp.archived2)
      temp.archived2Vif[temp.archived2Vif > 3] # empty
}

### Models used to show relative importance of descriptors

#Compared using the year vs. month published vars; month yields less
deviance
#except for WebCitation, though to be consistent we'll use the month
for each one.

uniq.webBasic = glm(web ~ lastPM96 + LogPub + depth + LogTimesCited +
                    FundTextPresent + domSig + Source_top, data=uniq,
                    family=binomial("logit"))

uniq.iaBasic = glm(ia ~ firstPM96 + LogPub + depth + LogTimesCited +
                    FundTextPresent + domSig + Source_top, data=uniq,
                    family=binomial("logit"))

uniq.wcBasic = glm(wc ~ lastPM96 + LogPub + depth + LogTimesCited +
                    FundTextPresent + domSig + Source_top,
                    data=uniq, family=binomial("logit"))

cat("Web overall explained deviance:",
round(uniq.webBasic$null.deviance - uniq.webBasic$deviance), "\n")
cat("IA overall explained deviance:", round(uniq.iaBasic$null.deviance
- uniq.iaBasic$deviance), "\n")
cat("WC overall explained deviance:", round(uniq.wcBasic$null.deviance
- uniq.wcBasic$deviance), "\n")

temp = drop1(uniq.webBasic)
temp.webBasic = c(uniq.webBasic$null.deviance - uniq.webBasic$deviance,
                      sum(temp$Deviance - temp$Deviance[1]),
                      temp$Deviance[-1] - temp$Deviance[1])

temp = drop1(uniq.iaBasic)
temp.iaBasic = c(uniq.iaBasic$null.deviance - uniq.iaBasic$deviance,
                  sum(temp$Deviance - temp$Deviance[1]),
                  temp$Deviance[-1] - temp$Deviance[1])

temp = drop1(uniq.wcBasic)
temp.wcBasic = c(uniq.wcBasic$null.deviance - uniq.wcBasic$deviance,
                  sum(temp$Deviance - temp$Deviance[1]),
                        temp$Deviance[-1] - temp$Deviance[1])

uniq.predCombined = data.frame(web = temp.webBasic, ia = temp.iaBasic,
wc = temp.wcBasic)
rownames(uniq.predCombined) = c("Model Dev", "Unique Dev",
rownames(temp)[-1])

print("Deviance explained by each predictor")
print(uniq.predCombined)

for (method in methods) {
  newCol = paste(method, "Pct", sep="")
  uniq.predCombined[,newCol] = uniq.predCombined[,method]
  uniq.predCombined[2:nrow(uniq.predCombined),newCol] =

uniq.predCombined[2:nrow(uniq.predCombined),method]/uniq.predCombined["
Unique Dev",method]
}

rm(temp, temp.webBasic, temp.iaBasic, temp.wcBasic, newCol)

if (SHOW_THINKING) {
  # Collinearity estimation -- divide the model deviance by the unique
deviance
  # accounted for by each of the predictors.
  # The reasoning is that a model built without a particular predictor
  # could have some of its prediction capability explained by another
var
  uniq.predCombined[2,methods]/uniq.predCombined[1,methods]

  # Output the predictor importance
  write.csv(uniq.predCombined, file="uniq_reduced_contributions.csv")

}

### Descriptive statistics about URL retention and availability
urls.webTrueLen = length(which(urls$web))
uniq.webTrueLen = length(which(uniq$web))

# Examining survFit gives median survival times
#urls.survFit = survfit(urls.survForm ~ 1, data=urls)
urls.survFitM = survfit(urls.survFormM ~ 1, data=urls)

print("Median survival times for URLs")
print(urls.survFitM)
#plot(urls.survFit)
#plot(urls.survFitM)

#uniq.survFit = survfit(uniq.survForm ~ 1, data=uniq)
uniq.survFitM = survfit(uniq.survFormM ~ 1, data=uniq)

print("Median survival times for unique URLs")
#print(uniq.survFit)
print(uniq.survFitM)

cat("Percent URLs (unique) dead: ", round(1 - urls.webTrueLen/urls.len,
2)*100,
    "(", round(1 - uniq.webTrueLen/uniq.len, 2)*100, ")\n")

cat("Percent URLs (unique) alive: ", round(urls.webTrueLen/urls.len,
2)*100,
    "(", round(uniq.webTrueLen/uniq.len, 2)*100, ")\n")

urls.iaLen = length(which(urls$ia))
uniq.iaLen = length(which(uniq$ia))

cat("Percent URLs (unique) in IA: ", round(urls.iaLen/urls.len, 2)*100,
"(",
    round(uniq.iaLen/uniq.len, 2)*100, ")\n")

urls.wcLen = length(which(urls$wc))
uniq.wcLen = length(which(uniq$wc))

cat("Percent URLs (unique) in WC: ", round(urls.wcLen/urls.len, 2)*100,
"(",
    round(uniq.wcLen/uniq.len, 2)*100, ")\n")

urls.archivedLen = length(which(urls$wc | urls$ia))
uniq.archivedLen = length(which(uniq$wc | uniq$ia))
cat("Percent URLs (unique) archived: ",
round(urls.archivedLen/urls.len, 2)*100,
    "(", round(uniq.archivedLen/uniq.len, 2)*100, ")\n")

# URLs available by any mechanism (live, archived)
urls.availableAnyLen = length(which(urls$archived | urls$web))
uniq.availableAnyLen = length(which(uniq$archived | uniq$web))

cat("Percent URLs (unique) available in some manner: ",
    round(urls.availableAnyLen/urls.len, 2)*100,
    "(", round(uniq.availableAnyLen/uniq.len, 2)*100, ")\n")

# Archive Site Performance for Missing URLs
urls.savedLen = length(urls[urls$web == F & (urls$ia == T | urls$wc ==
T), "web"])
urls.deadLen = length(which(!urls$web))

uniq.savedLen = length(uniq[uniq$web == F & (uniq$ia == T | uniq$wc ==
T), "web"])
uniq.deadLen = length(which(!uniq$web)) # For u

cat("Percent of missing URLs (unique) archived:",
round(urls.savedLen/urls.deadLen, 2)*100, "(",
    round(uniq.savedLen/uniq.deadLen, 2)*100, ")\n")

urls.iaSavedLen = length(urls[urls$web == F & urls$ia == T, "web"])
uniq.iaSavedLen = length(uniq[uniq$web == F & uniq$ia == T, "web"])

cat("Percent of missing URLs (unique) archived by IA:",
round(urls.iaSavedLen/urls.deadLen, 2)*100, "(",
    round(uniq.iaSavedLen/uniq.deadLen, 2)*100, ")\n")

urls.wcSavedLen = length(urls[urls$web == F & urls$wc == T, "web"])
uniq.wcSavedLen = length(uniq[uniq$web == F & uniq$wc == T, "web"])
```

```
cat("Percent of missing URLs (unique) archived by WC:",
round(urls.wcSavedLen/urls.deadLen, 2)*100, "(",
    round(uniq.wcSavedLen/uniq.deadLen, 2)*100, ")\n")

cat("Number of uniq URLs submitted to archiving engines: IA",
    length(which(uniq$iaSubmit2012.11.15 == "True")),
    ", WC", length(which(substr(uniq$wcSubmit2012.11.30, 0, 1) ==
"6")), "\n")

if (SHOW_THINKING) {
    # For excel/graph use
    write.csv(uniq,"uniq_data.csv")
}
### Other random descriptives

# Redirectors (DOI, PURL)
cat("Number of DOI sites (unique):",
length(grep("http://dx\\.doi\\.org", urls$url, ignore.case=T)),
    "(", length(grep("http://dx\\.doi\\.org", uniq$url,
ignore.case=T)), ")\n")

cat("Number of PURL sites (unique):", length(grep("http://purl\\.",
urls$url, ignore.case=T)),
    "(", length(grep("http://purl\\.", uniq$url, ignore.case=T)),
")\n")

## Remedies

# Compare our results with Wren, 2008's of 5% disappeared from papers
with > 2 publishings
temp.alive = prop.table(table(uniq[uniq$nPub > 1, "web"]))

cat("Living URLs published > 1 times living and missing:",
    temp.alive[2], temp.alive[1], "\n")

temp.alive = prop.table(table(uniq[uniq$nPub == 1, "web"]))

cat("Living URLs published 1 time living and missing:",
    temp.alive[2], temp.alive[1], "\n")

rm(temp.alive)

# how many IA sites were blocked?
table(urls$ia2011.05.23Orig)
table(uniq$ia2011.05.23Orig)

# What percentage of the blocking sites are missing?
temp.urlsBlocked = subset(urls,ia2011.05.23Orig == "CrawlingBlocked")
temp.uniqBlocked = subset(uniq,ia2011.05.23Orig == "CrawlingBlocked")

cat("Internet Archive sites that were blocked from archiving due to
robots.txt (uniq): ",
    nrow(temp.urlsBlocked), ", ",
    prop.table(table(urls$ia2011.05.23Orig == "CrawlingBlocked"))[2] *
100,
    "% (",
    nrow(temp.uniqBlocked), ", ",
    prop.table(table(uniq$ia2011.05.23Orig == "CrawlingBlocked"))[2] *
100,
    "%)\n", sep="")

cat("Unavailable Internet Archive sites that were blocked from
archiving due to robots.txt (uniq): ",
    table(temp.urlsBlocked$web)[1], ", ",
    prop.table(table(temp.urlsBlocked$web))[1] * 100,
    "% (",
    table(temp.uniqBlocked$web)[1], ", ",
    prop.table(table(temp.uniqBlocked$web))[1] * 100,
    "%)\n", sep="")

rm(temp.uniqBlocked, temp.urlsBlocked)

### Calculate median lifetimes and other statistics for particular
subject areas

temp.subjTop20 = names(sort(table(urlsSubjExp$SC), decreasing=T)[1:20])

begin = Sys.time()
subjSurvModels = lapply(temp.subjTop20, function(x) {
    cat("Survival Times for subject ", x, "\n")
    temp.us = subset(urlsSubjExp, SC == x)
    temp.survFormM = Surv(LIFE_DIFF*12 - temp.us$PM96, -Inf,
temp.us$event, type="interval")
    return(survfit(temp.survFormM ~ 1, data=temp.us))
})
print(Sys.time() - begin)

names(subjSurvModels) = temp.subjTop20

survMedians = data.frame(t(sapply(subjSurvModels, function(x) {
    return(summary(x)$table)
})))

## Count number of living and dead URLs for each subject
## for displaying in table form.

survMedians$nAlive = sapply(temp.subjTop20, function(x) {
  length(which(subset(urlsSubjExp, SC == x)$web == TRUE))
})

survMedians$nDead = sapply(temp.subjTop20, function(x) {
  length(which(subset(urlsSubjExp, SC == x)$web == FALSE))
})

# TODO: perhaps convert to months?
survMedians$medianPY = sapply(temp.subjTop20, function(x) {
  temp.us = urlsSubjExp[urlsSubjExp$SC == x,]
  return(median(temp.us$PY96))
}) + 1996

if (SHOW_THINKING) {
  # Output survMedians table
  write.csv(survMedians, file="survMedians.csv")
}
```

```
rm(temp.subjTop20)

# Produce box plots of the top 20 subject areas
library(lattice)

# Draw a horizontal bar chart with each row being a subject
# and the corresponding row containing the median survival time as well
# as the median age of URLs
library(Hmisc)
# Sort by median first
#survMedians = survMedians[order(survMedians$median, decreasing=T),]
errbar(row.names(survMedians), survMedians$median/12,
survMedians$X0.95UCL/12, survMedians$X0.95LCL/12)



urls.pctByYear = data.frame(year   = c(sort(unique(urls$PY96)) + 1996,
"Total"),
                            year96 = c(sort(unique(urls$PY96)),
"Total"))
uniq.pctByYear = data.frame(year   = c(sort(unique(uniq$firstPY96)) +
1996, "Total"),
                            year96 = c(sort(unique(uniq$firstPY96)),
"Total"))

# List of columns to include

temp.pctByYearCols = c("web", "archived", urls.archive_cols)

for(colName in temp.pctByYearCols) {
  urls.pctByYear[,colName] = c(prop.table(table(urls$PY96,
urls[,colName]), 1)[,2],
                               prop.table(table(urls[,colName]))[2])
  uniq.pctByYear[,colName] = c(prop.table(table(uniq$firstPY96,
uniq[,colName]), 1)[,2],
                               prop.table(table(uniq[,colName]))[2])

}

rm(temp.pctByYearCols)

# Draw a bar chart to show what difference submitting missing URLs made
# temp.totals = subset(urls.pctByYear, year == "Total",
select=urls.archive_cols)
# temp.totals = temp.totals[,order(colnames(temp.totals))] # Reorder
columns by name
# rownames(temp.totals) = "Pct"
# temp.totals = data.frame(Pct = t(temp.totals))
# temp.trunc = substr(rownames(temp.totals), 0, 2)
# temp.totals$Type = ifelse(temp.trunc == "ar", "archived", temp.trunc)

if (SHOW_THINKING) {
  write.csv(urls.pctByYear, file="urls_pctByYear.csv")
  write.csv(temp.totals, file="urls_totalsByYear.csv")
}
```

```
# Test some models - since we've written the file out, we can drop the
"total" line
urls.pctByYear = subset(urls.pctByYear, year!="Total")
urls.pctByYear$year = as.integer(as.character(urls.pctByYear$year))
urls.pctByYear$year96 = as.integer(as.character(urls.pctByYear$year96))

urls.webByYear.lm = lm(web ~ year96, data=urls.pctByYear)
urls.webByYear.glm = glm(web ~ PY96, family=binomial(link="logit"),
data=urls)
urls.archivedByYear.lm = lm(archived ~ year96, data=urls.pctByYear)

cat("Linear coefficients (R^2) for URLs percentage by year overall:",
    urls.webByYear.lm$coefficients[2], "(",
    summary(urls.webByYear.lm)$r.squared * 100, "%)\n")

if (SHOW_THINKING) {
  # Display components of linear models
  summary(urls.webByYear.lm)
  summary(urls.archivedByYear.lm)
}

######## Do it for the unique ones too

#print(round(uniq.pctByYear, 2))

# Calculate what percentage increase was seen by IA and WC due to
submissions
before = table(uniq$ia2012.10.18)[2]
after  = table(uniq$ia2013.02.05)[2]
cat("Uniq Percent increase for IA (", after, "-", before, "=", after-
before,
    "):", 100*(after - before)/before, "\n")

before = table(uniq$wc2012.10.18)[2]
after  = table(uniq$wc2013.02.05)[2]
cat("Uniq Percent increase for WC (", after, "-", before, "=", after-
before,
    "):", 100*(after - before)/before, "\n")

# Calculate URLs (+uniq) submitted to IA and WC
temp.iaSubmitted = nrow(subset(urls, iaSubmit2012.11.15 != 'Skipped'))
temp.iaSubmittedUniq = nrow(subset(uniq, iaSubmit2012.11.15 !=
'Skipped'))
cat("URLs submitted to IA (unique):", temp.iaSubmitted, "(",
    temp.iaSubmittedUniq, ")\n")

temp.wcSubmitted = nrow(subset(urls, wcSubmit2012.11.30 != 'Skipped'))
temp.wcSubmittedUniq = nrow(subset(uniq, wcSubmit2012.11.30 !=
'Skipped'))
cat("URLs submitted to WC (unique):", temp.wcSubmitted, "(",
    temp.wcSubmittedUniq, ")\n")



rm(before, after, temp.iaSubmitted, temp.iaSubmittedUniq,
temp.wcSubmitted,
   temp.wcSubmittedUniq)
```

```
### How many URLs in the IA returned an error status, but were
successfully archived?
cat("URLs submitted to IA which returned error but were successfully
archived:",
    length(which(uniq$iaSubmit2012.11.15 %in%
c("wgetStatus8","wgetStatus6") & uniq$ia2013.02.05)),
    "\n")

cat("URLs submitted to WC which returned error but were successfully
archived:",
    length(which(uniq$wcSubmit2012.11.30 %in%
c("UnexpectedXML","NoResultNoError","fatalError","emailError","httplib
error") & uniq$wc2013.02.05)),
    "\n")

cat("URLs submitted to WC which returned success but were
unsuccessfully archived:",
    length(which(grepl('^6', uniq$wcSubmit2012.11.30) &
!uniq$wc2013.02.05)),
    "\n")

uniq.webByYear.lm = lm(web ~ year96, data=uniq.pctByYear)

uniq.webByYear.glm = glm(web ~ firstPY96,
family=binomial(link="logit"), data=uniq)

uniq.archivedByYear.lm = lm(archived ~ year96, data=uniq.pctByYear)

if (SHOW_THINKING) {
  # Display components of linear models
  summary(uniq.webByYear.lm)
  summary(uniq.archivedByYear.lm)
  write.csv(uniq.pctByYear, file="uniq_pctByYear.csv")
  write.csv(temp.totals, file="uniq_totalsByYear.csv")
}

temp.midcount = length(which((uniq$web_pct < .9 & uniq$web_pct > 0)))

cat("URL count (percent) whose availability was > 0 or < .9:",
    temp.midcount, "(",
    temp.midcount/length(uniq$web_pct),
    ")\n")

rm(temp.midcount)

### Decay rate stability: comparing 1996-1999 to 2000-2010. In
Conclusions.

# 1996-1999
urls.survTable = prop.table(table(urls$web, urls$PY96+1996),
margin=2)[2,]
uniq.survTable = prop.table(table(uniq$web, uniq$firstPY96+1996),
margin=2)[2,]

temp.urlsAvail = urls.survTable[names(urls.survTable) < 2000]
temp.uniqAvail = uniq.survTable[names(uniq.survTable) < 2000]
```

```
cat("Variation (max-min) between 1996 and 1999, inclusive (uniq):",
max(temp.urlsAvail) - min(temp.urlsAvail),
    "(", max(temp.uniqAvail) - min(temp.uniqAvail), ")\n")

# Convert back to PY96 format for lm
names(temp.urlsAvail) = as.integer(names(temp.urlsAvail)) - 1996
names(temp.uniqAvail) = as.integer(names(temp.uniqAvail)) - 1996

temp.urlsLm = summary(lm(temp.urlsAvail ~
as.integer(names(temp.urlsAvail))))
temp.uniqLm = summary(lm(temp.uniqAvail ~
as.integer(names(temp.uniqAvail))))

cat("R squared for 1996-1999 linear fit (unique):",
temp.urlsLm$r.squared,
    "(", temp.uniqLm$r.squared, ")\n")

# 2000-2010
urls.survTable = prop.table(table(urls$web, urls$PY96+1996),
margin=2)[2,]
uniq.survTable = prop.table(table(uniq$web, uniq$firstPY96+1996),
margin=2)[2,]

temp.urlsAvail = urls.survTable[names(urls.survTable) >= 2000]
temp.uniqAvail = uniq.survTable[names(uniq.survTable) >= 2000]

cat("Variation (max-min) between 2000 and 2010, inclusive (uniq):",
max(temp.urlsAvail) - min(temp.urlsAvail),
    "(", max(temp.uniqAvail) - min(temp.uniqAvail), ")\n")

# Convert back to PY96 format for lm
names(temp.urlsAvail) = as.integer(names(temp.urlsAvail)) - 1996
names(temp.uniqAvail) = as.integer(names(temp.uniqAvail)) - 1996

temp.urlsLm = summary(lm(temp.urlsAvail ~
as.integer(names(temp.urlsAvail))))
temp.uniqLm = summary(lm(temp.uniqAvail ~
as.integer(names(temp.uniqAvail))))

cat("R squared for 2000-2010 linear fit (unique):",
temp.urlsLm$r.squared,
    "(", temp.uniqLm$r.squared, ")\n")

rm(temp.urlsLm, temp.uniqLm, temp.uniqAvail, temp.urlsAvail)

### URLs appearing more than once
temp.multPub = subset(uniq, nPub > 1) # URLs published more than once

cat("URLs appearing more than once:",
    nrow(temp.multPub), "or", nrow(temp.multPub)/nrow(uniq), "%\n")

cat("Multiply published URLs only published in 1 journal:",
    nrow(subset(temp.multPub, nJourns == 1))/nrow(temp.multPub), "%\n")

cat("Funding text in multiply published URLs is different",
    nrow(subset(temp.multPub, FundTextPresent != 1.0 & FundTextPresent
!= 0.0))/nrow(temp.multPub),
    "% of the time\n")

rm(temp.multPub)

cat("Overall elapsed time:", Sys.time() - temp.begin, "\n")
```

analysis/WOSstats.R

```
# Runs a linear model for WOS data

setwd("/path/to/analysis/")
wos = read.csv("WOSstats.csv", header=T)
# Format is: Year, withHttp, total

wos$pct = wos$withHttp/wos$total
wos$year96 = wos$Year - 1996

wos.pctlm = lm(pct ~ year96, data=wos)
summary(wos.pctlm)
print(wos.pctlm$coefficients * 100) # Show in percentage points

wos.numlm = lm(withHttp ~ year96, data=wos)
summary(wos.numlm)
```

BIBLIOGRAPHY (COMPREHENSIVE)

1. Menezes AJ, Van Oorschot PC, Vanstone SA: *Handbook of applied cryptography.* Boca Raton: CRC Press; 1997.
2. Ducut E, Liu F, Fontelo P: **An update on Uniform Resource Locator (URL) decay in MEDLINE abstracts and measures for its mitigation.** *BMC Med Inform Decis Mak* 2008, **8:**-.
3. Dimitrova DV, Bugeja M: **Consider the source: Predictors of online citation permanence in communication journals.** *Portal-Libraries and the Academy* 2006, **6:**269-283.
4. Wren JD: **URL decay in MEDLINE - a 4-year follow-up study.** *Bioinformatics* 2008, **24:**1381-1385.
5. Wren JD: **404 not found: the stability and persistence of URLs published in MEDLINE.** *Bioinformatics* 2004, **20:**668-U208.
6. Yang SL, Qiu JP, Xiong ZY: **An empirical study on the utilization of web academic resources in humanities and social sciences based on web citations.** *Scientometrics* 2010, **84:**1-19.
7. Koehler W: **An analysis of Web page and Web site constancy and permanence.** *J Am Soc Inf Sci* 1999, **50:**162-180.
8. Aronsky D, Madani S, Carnevale RJ, Duda S, Feyder MT: **The prevalence and inaccessibility of Internet references in the biomedical literature at the time of publication.** *J Am Med Inform Assn* 2007, **14:**232-234.
9. Wren JD, Johnson KR, Crockett DM, Heilig LF, Schilling LM, Dellavalle RP: **Uniform resource locator decay in dermatology journals - Author attitudes and preservation practices.** *Arch Dermatol* 2006, **142:**1147-1152.
10. Ioannidis JPA, Allison DB, Ball CA, Coulibaly I, Cui XQ, Culhane AC, Falchi M, Furlanello C, Game L, Jurman G, et al: **Repeatability of published microarray gene expression analyses.** *Nature Genetics* 2009, **41:**149-155.
11. Jasny BR, Chin G, Chong L, Vignieri S: **Again, and Again, and Again ….** *Science* 2011, **334:**1225.
12. Wicherts JM: **Psychology must learn a lesson from fraud case.** *Nature* 2011, **480:**7-7.
13. Wicherts JM, Bakker M, Molenaar D: **Willingness to Share Research Data Is Related to the Strength of the Evidence and the Quality of Reporting of Statistical Results.** *Plos One* 2011, **6:**e26828.
14. **The DOI System** [http://www.doi.org/]
15. **PURL Home Page** [http://purl.org]
16. Casserly MF, Bird JE: **Web citation availability: Analysis and implications for scholarship.** *College & Research Libraries* 2003, **64:**300-317.
17. **The Internet Archive** [http://www.archive.org/web/web.php]
18. Eysenbach G, Trudell M: **Going, going, still there: Using the WebCite service to permanently archive cited web pages.** *Journal of Medical Internet Research* 2005, **7:**2-6.
19. Barnes N: **Publish your computer code: it is good enough.** *Nature* 2010, **467:**753-753.

20. Peng RD: **Reproducible Research in Computational Science.** *Science* 2011, **334:**1226-1227.
21. **Key Facts on Digital Object identifier System** [http://www.doi.org/factsheets/DOIKeyFacts.html]
22. **EZID: Pricing** [http://n2t.net/ezid/home/pricing]
23. Markwell J, Brooks DW: **"Link rot" limits the usefulness of web-based educational materials in biochemistry and molecular biology.** *Biochemistry and Molecular Biology Education* 2003, **31:**69-72.
24. Thorp AW, Brown L: **Accessibility of internet references in Annals of Emergency Medicine: Is it time to require archiving?** *Ann Emerg Med* 2007, **50:**188-192.
25. Carnevale RJ, Aronsky D: **The life and death of URLs in five biomedical informatics journals.** *International Journal of Medical Informatics* 2007, **76:**269-273.
26. Wagner C, Gebremichael MD, Taylor MK, Soltys MJ: **Disappearing act: decay of uniform resource locators in health care management journals.** *J Med Libr Assoc* 2009, **97:**122-130.
27. Duda JJ, Camp RJ: **Ecology in the information age: patterns of use and attrition rates of internet-based citations in ESA journals, 1997-2005.** *Frontiers in Ecology and the Environment* 2008, **6:**145-151.
28. Rhodes S: **Breaking Down Link Rot: The Chesapeake Project Legal Information Archive's Examination of URL Stability.** *Law Library Journal* 2010, **102:**581-597.
29. Goh DHL, Ng PK: **Link decay in leading information science journals.** *Journal of the American Society for Information Science and Technology* 2007, **58:**15-24.
30. Casserly MF, Bird JE: **Web citation availability - A follow-up study.** *Libr Resour Tech Ser* 2008, **52:**42-53.
31. Russell E, Kane J: **The missing link - Assessing the reliability of Internet citations in history journals.** *Technology and Culture* 2008, **49:**420-429.
32. Koehler W: **A longitudinal study of Web pages continued: a consideration of document persistence.** *Information Research-an International Electronic Journal* 2004, **9:**-.
33. Dellavalle RP, Hester EJ, Heilig LF, Drake AL, Kuntzman JW, Graber M, Schilling LM: **Information science - Going, going, gone: Lost Internet references.** *Science* 2003, **302:**787-788.
34. Evangelou E, Trikalinos TA, Ioannidis JPA: **Unavailability of online supplementary scientific information from articles published in major journals.** *Faseb Journal* 2005, **19:**1943-1944.
35. Sellitto C: **The impact of impermanent web-located citations: A study of 123 scholarly conference publications.** *Journal of the American Society for Information Science and Technology* 2005, **56:**695-703.
36. Bar-Ilan J, Peritz B: **The lifespan of "informetrics" on the Web: An eight year study (1998-2006).** *Scientometrics* 2009, **79:**7-25.
37. Gomes D, Silva MJ: **Modelling Information Persistence on the Web.** In *Book Modelling Information Persistence on the Web* (Editor ed.^eds.). City; 2006.

38. Markwell J, Brooks DW: **Evaluating web-based information: Access and accuracy.** *Journal of Chemical Education* 2008, **85:**458-459.
39. Wu ZQ: **An empirical study of the accessibility of web references in two Chinese academic journals.** *Scientometrics* 2009, **78:**481-503.
40. Klein JP, Moeschberger ML: *Survival analysis : techniques for censored and truncated data.* 2nd edn. New York: Springer; 2003.
41. Bar-Ilan J, Peritz BC: **Evolution, continuity, and disappearance of documents on a specific topic on the web: A longitudinal study of "informetrics".** *Journal of the American Society for Information Science and Technology* 2004, **55:**980-990.
42. Peng RD: **Reproducible research and Biostatistics.** *Biostatistics* 2009, **10:**405-408.
43. Ince DC, Hatton L, Graham-Cumming J: **The case for open computer programs.** *Nature* 2012, **482:**485-488.
44. R Development Core Team: **R: A Language and Environment for Statistical Computing.** In *Book R: A Language and Environment for Statistical Computing* (Editor ed.^eds.). City: R Foundation for Statistical Computing; 2011.
45. Therneau T: **A Package for Survival Analysis in S.** In *Book A Package for Survival Analysis in S* (Editor ed.^eds.), 2.36-12 edition. City; 2012.
46. **WebCite Technical Background and Best Practices Guide** [http://www.webcitation.org/doc/WebCiteBestPracticesGuide.pdf]
47. Hennessey J, Ge S: **A cross disciplinary study of link decay and the effectiveness of mitigation techniques.** *Bmc Bioinformatics* 2013, **14:**S5.
48. Rothenberg J: **Ensuring the Longevity of Digital Documents.** *Scientific American* 1995, **272:**42-47.
49. Rosenthal DSH: **Format obsolescence: assessing the threat and the defenses.** *Libr Hi Tech* 2010, **28:**195-210.
50. Traynor C, Williams MG: **Why are geographic information systems hard to use?** 1995**:**288-289.
51. Howe B: **Virtual Appliances, Cloud Computing, and Reproducible Research.** *Computing in Science & Engineering* 2012, **14:**36-41.
52. **Sitemaps.org** [http://www.sitemaps.org/]
53. **RDF Primer** [http://www.w3.org/TR/2004/REC-rdf-primer-20040210/]
54. **Ensembl Annotated Human Genome Data (MySQL Release 68)** [https://aws.amazon.com/datasets/2315]
55. Katayama T, Nakao M, Takagi T: **TogoWS: integrated SOAP and REST APIs for interoperable bioinformatics Web services.** *Nucleic Acids Res* 2010, **38:**W706-W711.
56. Brase J: **DataCite - A Global Registration Agency for Research Data.** *2009 Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology* 2009**:**257-261.
57. Wang X, Yu H: **How to Break MD5 and Other Hash Functions.** In *Advances in Cryptology – EUROCRYPT 2005. Volume* 3494. Edited by Cramer R: Springer Berlin Heidelberg; 2005: 19-35: *Lecture Notes in Computer Science*].

58. Jin K, Miller EL: **The effectiveness of deduplication on virtual machine disk images.** In *Book The effectiveness of deduplication on virtual machine disk images* (Editor ed.^eds.). pp. 1-12. City: ACM; 2009:1-12.
59. Ng C-H, Ma M, Wong T-Y, Lee PPC, Lui JCS: **Live deduplication storage of virtual machine images in an open-source cloud.** In *Book Live deduplication storage of virtual machine images in an open-source cloud* (Editor ed.^eds.). pp. 80-99. City: International Federation for Information Processing; 2011:80-99.
60. **Linked Virtual Machines** [http://pubs.vmware.com/vsphere-50/topic/com.vmware.wssdk.pg.doc_50/PG_Ch11_VM_Manage.13.4.html]
61. **Ubuntu Cloud Images** [http://cloud-images.ubuntu.com/]
62. Tridgell A: **Efficient Algorithms for Sorting and Synchronization.** Australian National University, 1999.
63. **lrzip: Long Range ZIP or Lzma RZIP** [http://ck.kolivas.org/apps/lrzip/]
64. **Data compression programs** [http://www.mattmahoney.net/dc/]
65. Smith MA, Pieper J, Gruhl D, Real LV: **IZO: Applications of Large-Window Compression to Virtual Machine Management.** In *LISA*. 2008: 121-132.
66. Lam MSL, Sapuntzakis CP, Chandra RUV, Zeldovich NB, Rosenblum M, Chow JE, Brumley DJ: **Cache-based system management architecture with virtual appliances, network repositories, and virtual appliance transceivers.** In *Book Cache-based system management architecture with virtual appliances, network repositories, and virtual appliance transceivers* (Editor ed.^eds.). City: Google Patents; 2008.
67. Reich J, Laadan O, Brosh E, Sherman A, Misra V, Nieh J, Rubenstein D: **VMTorrent: scalable P2P virtual machine streaming.** In *Book VMTorrent: scalable P2P virtual machine streaming* (Editor ed.^eds.). pp. 289-300. City: ACM; 2012:289-300.
68. Lagar-Cavilla HA, Whitney JA, Scannell AM, Patchin P, Rumble SM, De Lara E, Brudno M, Satyanarayanan M: **SnowFlock: rapid virtual machine cloning for cloud computing.** In *Proceedings of the 4th ACM European conference on Computer systems*. ACM; 2009: 1-12.
69. Van Gorp P, Mazanek S: **SHARE: a web portal for creating and sharing executable research papers.** *Procedia Computer Science* 2011, **4:**589-597.
70. **QEMU Official OS Support List - MS-DOS:** [http://www.claunia.com/qemu/objectManager.php?sClass=application&iId=53]
71. **Creating a MS-DOS Virtual PC under Virtualbox** [https://mylinuxramblings.wordpress.com/2010/12/05/linux-mint-debian-edition-lmde-first-impressions/]
72. Angiuoli S, Matalka M, Gussman A, Galens K, Vangala M, Riley D, Arze C, White J, White O, Fricke WF: **CloVR: A virtual machine for automated and portable sequence analysis from the desktop using cloud computing.** *Bmc Bioinformatics* 2011, **12:**356.
73. **Bio-Linux 6.0** [http://nebc.nox.ac.uk/tools/bio-linux]
74. Wang Y, Mehta G, Mayani R, Lu JX, Souaiaia T, Chen YH, Clark A, Yoon HJ, Wan L, Evgrafov OV, et al: **RseqFlow: workflows for RNA-Seq data analysis.** *Bioinformatics* 2011, **27:**2598-2600.

75. Klinginsmith J, Mahoui M, Wu YM: **Towards Reproducible eScience in the Cloud.** 2011**:**582-586.
76. Goecks J, Nekrutenko A, Taylor J, Galaxy T: **Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.** *Genome Biol* 2010, **11**.
77. Leisch F: **Sweave: Dynamic generation of statistical reports using literate data analysis.** In *Compstat*. Springer; 2002: 575-580.

GLOSSARY

Word	Definition
Digital Resource	Useful functionality available through a computer, whether via the Internet or locally.
Internet Archive	A system set up in the 1990s to provide "universal access to all knowledge". One of its services is one which attempts to archive as much of the visible Internet as possible using the "Wayback Machine". Available at http://archive.org
Uniform Resource Locator	Format for a string referring to a resource. Most commonly seen with "http" or "https", such as: https://en.wikipedia.org/wiki/Url
Virtual Machine	A packaging that encompasses all data and logic necessary to replicate a computing environment
WebCite	System meant for the on-demand archival of scholarly resources. Available at http://webcitation.org