

Spring 2016

Scripts in a Frame: A Framework for Archiving Deferred Representations

Justin F. Brunelle
Old Dominion University, jbrunelle008@gmail.com

Follow this and additional works at: https://digitalcommons.odu.edu/computerscience_etds



Part of the [Computer Sciences Commons](#), and the [Library and Information Science Commons](#)

Recommended Citation

Brunelle, Justin F.. "Scripts in a Frame: A Framework for Archiving Deferred Representations" (2016).
Doctor of Philosophy (PhD), Dissertation, Computer Science, Old Dominion University, DOI: 10.25777/
k8px-z178
https://digitalcommons.odu.edu/computerscience_etds/10

This Dissertation is brought to you for free and open access by the Computer Science at ODU Digital Commons. It has been accepted for inclusion in Computer Science Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.

**SCRIPTS IN A FRAME:
A FRAMEWORK FOR ARCHIVING DEFERRED
REPRESENTATIONS**

by

Justin F. Brunelle
B.S. May 2008, Old Dominion University
M.S. May 2010, Old Dominion University

A Dissertation Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

COMPUTER SCIENCE

OLD DOMINION UNIVERSITY
May 2016

Approved by:

Dr. Michael L. Nelson (Director)

Dr. Michele C. Weigle (Member)

Dr. Elizabeth J. Vincelette (Member)

Dr. Irwin B. Levinstein (Member)

ABSTRACT

SCRIPTS IN A FRAME: A FRAMEWORK FOR ARCHIVING DEFERRED REPRESENTATIONS

Justin F. Brunelle
Old Dominion University, 2016
Director: Dr. Michael L. Nelson

Web archives provide a view of the Web as seen by Web crawlers. Because of rapid advancements and adoption of client-side technologies like JavaScript and Ajax, coupled with the inability of crawlers to execute these technologies effectively, Web resources become harder to archive as they become more interactive. At Web scale, we cannot capture client-side representations using the current state-of-the-art toolsets because of the migration from Web *pages* to Web *applications*. Web applications increasingly rely on JavaScript and other client-side programming languages to load embedded resources and change client-side state. We demonstrate that Web crawlers and other automatic archival tools are unable to archive the resulting JavaScript-dependent representations (what we term *deferred representations*), resulting in missing or incorrect content in the archives and the general inability to replay the archived resource as it existed at the time of capture.

Building on prior studies on Web archiving, client-side monitoring of events and embedded resources, and studies of the Web, we establish an understanding of the trends contributing to the increasing unarchivability of deferred representations. We show that JavaScript leads to lower-quality mementos (archived Web resources) due to the archival difficulties it introduces. We measure the historical impact of JavaScript on mementos, demonstrating that the increased adoption of JavaScript and Ajax correlates with the increase in missing embedded resources. To measure memento and archive quality, we propose and evaluate a metric to assess memento quality closer to Web users' perception.

We propose a two-tiered crawling approach that enables crawlers to capture embedded resources dependent upon JavaScript. Measuring the performance benefits between crawl approaches, we propose a classification method that mitigates the

performance impacts of the two-tiered crawling approach, and we measure the frontier size improvements observed with the two-tiered approach. Using the two-tiered crawling approach, we measure the number of client-side states associated with each URI-R and propose a mechanism for storing the mementos of deferred representations.

In short, this dissertation details a body of work that explores the following: *why* JavaScript and deferred representations are difficult to archive (establishing the term *deferred representation* to describe JavaScript dependent representations); *the extent to which* JavaScript impacts archivability along with its impact on current archival tools; a metric for *measuring the quality* of mementos, which we use to describe the impact of JavaScript on archival quality; the *performance trade-offs* between traditional archival tools and technologies that better archive JavaScript; and *a two-tiered crawling approach* for discovering and archiving currently unarchivable descendants (representations generated by client-side user events) of deferred representations to mitigate the impact of JavaScript on our archives.

In summary, what we archive is increasingly different from what we as interactive users experience. Using the approaches detailed in this dissertation, archives can create mementos closer to what users experience rather than archiving the crawlers' experiences on the Web.

Copyright, 2016, by Justin F. Brunelle, All Rights Reserved.

ACKNOWLEDGEMENTS

My academic journey would not have been a success without an impressive cast of supporters; this dissertation and my success is attributed to these individuals.

I would like to thank my committee members for their valuable guidance and feedback. I also thank the rest of the ODU Computer Science Department for the support they have provided over the course of my academic tenure. I would also like to thank Samuel L. Jackson for teaching me to never be ashamed of being tired of these monkey fighting scripts on this Monday through Friday frame.

Dr. Michael L. Nelson has guided my journey through the Ph.D. program. His expertise, leadership, and guidance are without compare, and I cannot thank him enough for his patience, direction, and role in my academic career. I also look forward to continued mutual appreciation of Ford muscle cars, debates over the merits of certain football programs over others, and will work tirelessly to teach him the value of American IPAs.

I would also like to thank my family (mom, dad, and brothers) for their support, their push and motivation, and their understanding throughout my undergraduate and graduate work. They served as great role models and sounding boards throughout this process. I also would like to thank my grandma, who expressed her support and pride in my doctoral studies each time we spoke.

Most importantly, I thank my wife, June, who routinely took over my responsibilities allowing me to focus on my research and who has remained supportive despite my long nights, unavailable weekends, and crankiness during paper season. She is a constant source of support, external review, and venting recipient. I am sure Dr. Nelson thanks her, too.

Finally, I would not have completed my research without financial support from the National Science Foundation Division of Information & Intelligent Systems 1009392, National Endowment for the Humanities Digital Humanities Implementation Grant HK-50181-14, and the Library of Congress.

To all I have mentioned, as well as those that I have not: *Thank You!*

TABLE OF CONTENTS

	Page
LIST OF TABLES	xi
LIST OF FIGURES	xviii
Chapter	
1. INTRODUCTION	1
1.1 YOU ARE NOT A GADGET	2
1.2 #SOPABLACKOUT: A CASE STUDY	9
1.2.1 CRAIGSLIST	9
1.2.2 WIKIPEDIA	13
1.3 MODERN WEB ARCHIVING	16
1.4 RESEARCH QUESTIONS.....	19
1.5 DISSERTATION ROADMAP	21
2. BACKGROUND	23
2.1 WEB ARCHITECTURE	23
2.2 FROM HTML TO RICH INTERNET APPLICATIONS.....	27
2.2.1 HTML	27
2.2.2 REST PRINCIPLES	28
2.2.3 JAVASCRIPT AND AJAX	29
2.2.4 PROGRESSION OF WEB LANGUAGES	35
2.3 TRENDING TOWARD THE DYNAMIC WEB	39
2.4 HASHBANG URIS	41
2.5 TRENDS IN WEB ARCHIVING	43
2.5.1 MEMENTO	44
2.5.2 ARCHIVING MORE THAN JUST THE DESKTOP WEB ...	45
The Personalized Web	46
The Mobile Web	47
The Linked Data Web	49
2.6 DAVID HOCKNEY’S JOINERS.....	55
2.7 SUMMARY	57
3. RELATED WORK.....	58
3.1 ARCHIVAL EFFORTS AND TOOLS	58
3.2 ARCHIVAL CHALLENGES: THE DEEP, HIDDEN, AND EPHEMERAL WEB.....	66
3.3 WHAT MAKES A REPRESENTATION?	69
3.4 MEASURING ARCHIVE QUALITY	70
3.5 CACHING JAVASCRIPT-DEPENDENT REPRESENTATIONS	73

3.6	MONITORING THE CLIENT FOR DEBUGGING AND SECURITY	75
3.6.1	DETECTING MALWARE	76
3.6.2	DEBUGGING AND ERROR TRACKING	77
3.6.3	USER INTERACTIONS	78
3.7	SESSION REPLAY AND SHARING	79
3.8	SUMMARY	82
4.	UNDERSTANDING THE CHALLENGES OF ARCHIVING JAVASCRIPT	83
4.1	REVISITING THE #SOPABLACKOUT: TECHNICAL DETAILS	83
4.1.1	CRAIGSLIST: REVISITED	83
4.1.2	WIKIPEDIA: REVISITED	85
4.2	JAVASCRIPT AND MEMENTOS OF MEMENTOS	89
4.3	ZOMBIES IN THE ARCHIVES	97
4.4	MULTIPLE REPRESENTATIONS	101
4.5	SUMMARY	108
5.	MEASURING JAVASCRIPT IN THE ARCHIVES	110
5.1	MOTIVATING EXAMPLE	110
5.2	EXPERIMENT DESIGN	116
5.2.1	DATASETS	118
	Twitter	118
	Archive-It	119
	Collection Differences	119
5.3	ARCHIVING THE RESOURCES	121
5.4	RESOURCE METRICS	123
5.4.1	URI COMPLEXITY	123
5.4.2	CONTENT COMPLEXITY	124
5.5	REQUESTS FROM HTML AND JAVASCRIPT	124
5.6	RESOURCE SET ANALYSIS	125
5.6.1	URI COMPLEXITY	125
5.6.2	CONTENT COMPLEXITY	125
5.7	ARCHIVING EXPERIMENT DISCUSSION	127
5.8	MISSING EMBEDDED RESOURCES	127
5.9	LEAKAGE	128
5.10	IMPACT OF ACCESSIBILITY	132
5.11	MEASURING ARCHIVABILITY IN THE INTERNET ARCHIVE	134
5.12	CHALLENGES IN MEASURING PAST PERFORMANCE	135
5.13	THE IMPACT OF JAVASCRIPT ON MEMENTO COMPLETENESS	140
5.14	SUMMARY OF FINDINGS	146
5.15	CONTRIBUTION TO RESEARCH QUESTION 1	147
6.	MEASURING ARCHIVE QUALITY	149
6.1	MOTIVATING EXAMPLES	150
6.2	USERS' PERCEPTION OF DAMAGE	152
6.3	EVALUATING ORGANIC DAMAGE	157

6.4	CALCULATING MEMENTO DAMAGE	159
6.4.1	DEFINING D_M	159
6.4.2	WEIGHTING EMBEDDED RESOURCES	160
6.4.3	IMAGE DAMAGE CALCULATION	161
6.4.4	STYLE SHEET DAMAGE CALCULATION	161
6.4.5	THE D_M ALGORITHM	163
6.4.6	LIMITATIONS OF D_M CALCULATION	166
6.5	DAMAGE IN THE ARCHIVES	167
6.5.1	TURKER ASSESSMENT OF D_M	167
6.5.2	MEASURING THE INTERNET ARCHIVE	167
6.5.3	MEASURING WEBCITE	170
6.6	IMPACT OF JAVASCRIPT ON DAMAGE	175
6.6.1	CRAWLING DEFERRED REPRESENTATIONS	175
6.7	MEASURING ARCHIVE.IS	178
6.8	CONTRIBUTION TO RESEARCH QUESTION 2.....	180
7.	A TWO-TIERED APPROACH FOR CRAWLING DEFERRED REPRESENTATIONS	184
7.1	MOTIVATING EXAMPLES	185
7.2	A FRAMEWORK FOR TWO-TIERED CRAWLING	188
7.2.1	SINGLE-TIER CRAWLING FRAMEWORK	188
7.2.2	TWO-TIERED CRAWLING APPROACH	189
7.3	COMPARING CRAWLS	191
7.3.1	CRAWL TIME BY URI	192
7.3.2	URI DISCOVERY AND FRONTIER SIZE	194
7.3.3	FRONTIER PROPERTIES AND DEDUPLICATION	197
7.3.4	DEFERRED VS. NONDEFERRED CRAWLS	200
7.4	CLASSIFYING REPRESENTATIONS	200
7.5	PERFORMANCE OF TWO-TIERED CRAWLING	204
7.6	MAPPING AND IDENTIFYING DESCENDANTS	207
7.7	DESCENDANT EQUIVALENCY	210
7.8	DESCENDANT CRAWLING APPROACH	211
7.9	EDGE CASES	216
7.10	DESCENDANT STATES	217
7.10.1	DATASET DIFFERENCES	217
7.10.2	TRAVERSING PATHS	220
7.10.3	IMPACT ON CRAWL TIME	221
7.11	ARCHIVAL COVERAGE	222
7.12	STORING DESCENDANTS	226
7.13	CONTRIBUTION TO RESEARCH QUESTION 3.....	230
8.	FUTURE WORK, CONTRIBUTIONS, AND CONCLUSIONS.....	232
8.1	RESEARCH QUESTIONS REVISITED	233
8.2	FUTURE WORK	235
8.3	CONTRIBUTIONS	236

8.4	CONCLUSIONS	237
-----	-------------------	-----

LIST OF TABLES

Table	Page
1. Replay behavior of mementos of the Craigslist SOPA protest.....	12
2. Replay behavior of mementos of the Wikipedia SOPA protest.	13
3. The cardinality of the TimeMaps varies between the Desktop and Linked Data Webs (TimeMaps retrieved 2015-07-14).	51
4. Example URIs	119
5. Content Features of Each Collection.....	120
6. URI Features of Each Collection	121
7. Number of requests per memento by archive year.	144
8. The 11 URI-Rs used to create the manually damaged dataset. M_m values are provided for each m_1	154
9. The turkers selected m_0 as the preferred memento 81% of the time, and more consistently for larger ΔM_m values.	156
10. Confusion matrix of the turker assessments of the m_0 vs m_1 comparison test.	156
11. The turker evaluations of the m_2 vs m_3 comparisons when using M_m as a damage measurement.	158
12. Confusion matrix of the turker assessments of the m_2 vs m_3 comparison test against M_m	159
13. When compared to random, M_m performs worse than random selection and is worse than the optimal performance of m_0 vs m_1	159
14. D_m vs M_m for the images in Figures 72 and 73. Note $M_m > D_m$ in 2 of 5 cases.	166
15. The turker evaluations of the m_2 vs m_3 (sampled mementos) comparisons when using D_m as a damage measurement.....	168
16. Confusion matrix of the turker assessments of the m_2 vs m_3 comparison test against D_m	168

17.	D_m provides a closer estimate of turker perception of damage and our optimal performance of m_0 vs m_1 than M_m	168
18.	Performance of wget, Heritrix, and PhantomJS for crawls of 10,000 seed URIs.	194
19.	Detected duplicate URIs, entity bodies, and the overlap between the two using the five URI string trimming policies.	199
20.	Confusion matrix for the entire feature vector (F-Measure = 0.791).	203
21.	Confusion matrix for the resource features (features 9-12 of the vector; F-Measure = 0.844).	203
22.	Confusion matrix for the DOM features (features 1-8 of the vector; F-Measure = 0.806).	204
23.	Classification success statistics for DOM-only and DOM and Resource feature sets.	204
24.	A summary of <i>extrapolated</i> performance (based on our calculations) of single- and two-tiered crawling approaches.	205
25.	A simulated two-tiered crawl showing that the frontier sizes can be optimized while mitigating the performance impact of PhantomJS's crawl speed vs Heritrix's.	206
26.	The average distribution of descendants within the deferred representation URI-R set.	217
27.	The range of descendants varies greatly among the deferred representations.	218
28.	Breakdown of the URI-Rs with various events attached to their DOMs and the percent of all new embedded resources contributed by the events.	219
29.	The increases in run time and frontier size relative to the Heritrix only run.	222
30.	The top 10 URI-Rs that appear as embedded resources in descendants make up 22.4% of all resources added to the crawl frontier.	225
31.	JSON object representing s_n stored as the metadata of a WARC.	227
32.	The storage impact of deferred representations and their descendants is 5.12 times higher per URI-R than archiving nondeferred representations.	229

LIST OF FIGURES

Figure	Page
1. New York Times article on the September 11th attacks.	3
2. September 11th mementos invoke a different experience than Wikipedia. .	4
3. Hany SalahEldeen’s Facebook page is not represented (i.e., not archived at all due to Facebook’s use of the robots.txt protocol) in the archives. ..	6
4. A screenshot of the SOPA blackout on the Washington D.C. Craigslist site.	10
5. Mementos of the <code>craigslist.org</code> protest of SOPA contain evidence of the protest and their associated targets.	11
6. Screenshots of the live versions of the Wikipedia blackout in protest of SOPA.	14
7. Mementos of the <code>Wikipedia.org</code> protest of SOPA do not show the black-out splash page.	15
8. Tweets showing the increasing understanding of archival challenges and need for paradigm shifts in archiving.	18
9. An example from W3C of the URI-representation-resource relationship [135].	24
10. A HTTP GET Request for <code>http://www.justinfbrunelle.com</code> as captured by Mozilla FireFox.	25
11. The HTTP Response for <code>http://www.justinfbrunelle.com</code> as captured by Mozilla FireFox.	26
12. The DOM is a tree representation of the HTML.	28
13. Normal request-response interaction between client and server for a resource R.	30
14. Request-response interactions between client and server and page and server for resource R and embedded content.	31
15. JavaScript can interact with the local DOM when executed.	33
16. Events occur on the client, and can cause JavaScript to execute and modify the client-side DOM.	34

17.	Ajax interactions modify the DOM after the original page load.	35
18.	Google Maps is a Web <i>application</i> that changes as a result of user interactions. Changes are outlined in red annotations. Note that the URI does not change as a result of the interactions.	36
19.	Extending the example in Figure 15, HTML5 can read from local storage and modify the DOM.	39
20.	Resources are using more JavaScript to load embedded resources over time (Figure from [52]).	40
21.	The Memento Framework enables temporal browsing and aggregates the holdings of multiple archives. This image is available as part of the Memento Framework Introduction [289].	45
22.	President Obama’s Facebook page (as observed 2011-05-11) changes based on client-side parameters and is not adequately archived.	48
23.	HTTP Response for the Wikipedia SOPA page.	50
24.	An HTTP 303 redirect for the DBPedia resource.	52
25.	Further information resources are provided in the Link header.	53
26.	Following the Link header leads to new information resources in new Link headers.	54
27.	The Internet Archive TimeMap cardinality is much smaller for the Linked Data Web.	54
28.	Smaller, focused snapshots, in aggregate, form a complete picture [75]. This Hockney joiner is called “Yosemite.”	55
29.	David Hockney’s “joiners” are single images that, together, create a larger image [199]. This joiner, constructed in Hockney’s style, is by Jordan Mills.	56
30.	While Archive.is offers a high quality memento, some functionality is missing because the JavaScript is removed during archiving. This and URI rewriting leads to a paradox: archives have to transform HTML in order to “preserve” it.	63
31.	URI re-writing in WebCite converts embedded URI-Rs to URI-Ms.	64
32.	Brewster Kahle tweeted that the Internet Archive’s Wayback Machine contains roughly 479 billion pages (2014-05-27).	73

33.	The JavaScript embedded in the HTML provides the countdown functionality.	84
34.	Google Chrome's developer console showing the resources requested by http://web.archive.org/web/20130824022954/http://en.wikipedia.org/?banner=blackout and their associated response codes.	86
35.	An HTTP Request for the blackout banner.	87
36.	The JSON object that identifies the blackout page background image. ...	88
37.	The JavaScript that constructs the banner URI and uses Ajax to request the blackout banner.	89
38.	HTTP headers of the robustify JavaScript.	91
39.	This memento of a memento does not exist, but references a URI-M as a live-Web URI-R.	96
40.	A temporal inconsistency is shown in the 2008 memento of CNN.com from the Wayback Machine at URI-M http://web.archive.org/web/20080916123132/http://www.cnn.com/	99
41.	A temporally correct snapshot of a live CNN.com site	99
42.	A 2011 memento of IMDB.com from the Wayback Machine at URI-M http://web.archive.org/web/20110728165802/http://www.imdb.com/ 100	
43.	Another example of temporal inconsistencies from leakage in IMDB.com shows that leakage occurs in the archives.	100
44.	Live Web resources are requested when viewing a CNN.com memento from the Internet Archive.	101
45.	Embedded JavaScript (i.e., the scripts in a frame) is responsible for the leakage.	102
46.	WebCite memento of CNN.com with leakage.	102
47.	Live CNN.com resource containing the shared article.	103
48.	WARC Record headers for the same resource accessed using different <code>user-agent</code> strings.	105
49.	Mementos differ based on the parameters influencing the representations at crawl/capture time and the devices used to access the mementos.	106

50.	Resources return different representations based on the environment.	107
51.	Google Maps as it exists live and as a memento.	112
52.	The live version of http://www.albop.com/main.html (from the Archive-It collection) is perfectly archived using Heritrix, wget, and Web-Cite.	113
53.	Live version of http://www.desbarresmanor.com/daytouring/history.html from the Twitter collection.	114
54.	As shown by these mementos, the site http://www.desbarresmanor.com/daytouring/history.html is mostly archived with minor content leakage.	114
55.	Live version of http://perezhilton.com/2009-06-11-world-premiere-gaga-manson from the Twitter collection.	115
56.	As shown by these mementos, the site http://perezhilton.com/2009-06-11-world-premiere-gaga-manson is extremely difficult to archive.	117
57.	URI Complexity measure (UC).	126
58.	CC of the HTML.	126
59.	Percentage of resource requests from JavaScript show patterns similar to <i>CC</i>	127
60.	The HTTP 200 response codes for embedded resources.	129
61.	Percentage of resource requests going to remote hosts from both HTML and JavaScript.	131
62.	The Twitter collection (n=596) is, on average, younger than the Archive-It collection (n=590).	135
63.	An abbreviated TimeMap for http://www.doc.alabama.gov/	136
64.	The http://www.doc.alabama.gov/ mementos are perfectly archived through time since they limit their reliance on JavaScript to load embedded resources.	137

65.	CMT.com over time. Changes in design and thus the technologies used are easily observable after mementos archived in 2009-2013 (Figures 65(l)-65(p)), which is when jQuery is introduced into the page and used to load embedded resources.	138
66.	Resources are using more JavaScript to load embedded resources over time.	140
67.	JavaScript is responsible for loading an increasing proportion of mementos over time.	142
68.	JavaScript is responsible for an increasing proportion of missing resources.	143
69.	Number of requests per memento by archive year.	144
70.	Percent of missing resources from JavaScript by year.	145
71.	As JavaScript is relied on more heavily to load embedded resources, more resources are missed.	146
72.	The XKCD example demonstrates that embedded resources have varying human-perceived importance to their page.	150
73.	Mementos have different meanings and usefulness depending on which embedded resources are missing from the memento (and the proportion of missing resources, M_m).	151
74.	We asked the turkers to select the less damaged of two mementos. The two versions of the page are accessible in separate tabs.	155
75.	Missing style sheets causes content to shift left. We show the percent of content in the vertical partitions of the viewport.	164
76.	Missing style sheets causes content to shift left. We show the percent of content in the vertical partitions of the page.	165
77.	The average percentage of embedded resources missed per memento per year in the Internet Archive as compared to damage per memento per year ($\overline{D_m}=0.128$, $\overline{M_m}=0.132$).	169
78.	The distribution of the number of missing embedded resources per URI-M in the Internet Archive. Note that we limited the figures to 100 missing embedded resources.	171
79.	The distribution of the number of successfully dereferenced embedded resources per URI-M in the Internet Archive. Note that we limited the figures to 100 successfully dereferenced embedded resources.	172

80.	The number of missed embedded resources per Internet Archive memento per year and MIME type.	173
81.	The average percentage of embedded resources missed per memento per year in WebCite as compared to damage per memento per year ($\overline{D_m}=0.397$, $\overline{M_m}=0.176$).	174
82.	The distribution of the number of missing embedded resources per URI-M in WebCite. Note that we limited the figures to 60 missing embedded resources.	176
83.	The distribution of the number of successfully dereferenced embedded resources per URI-M in WebCite. Note that we limited the figures to 60 successfully dereferenced embedded resources.	177
84.	The number of missed embedded resources per WebCite memento per year and MIME type.	178
85.	The Δ_m measurements of Archive.is and WebCite indicate that Archive.is creates higher fidelity mementos than WebCite.	181
86.	Neither archival tool captures all embedded resources, but PhantomJS discovers the URI-Rs of two out of three embedded resources dependent upon JavaScript (B, C) while Heritrix misses them.	186
87.	The current archival work flow was designed to archive pre-JavaScript nondeferred representations.	189
88.	The proposed adaptation to the archival work flow handles deferred representations and descendants.	190
89.	Heritrix crawls 12.13 times faster than PhantomJS. The error lines indicate the standard deviation across all ten runs.	193
90.	PhantomJS discovers 1.75 times more embedded resources than Heritrix and 4.11 times more resources than wget. The averages and error lines indicate the standard deviation across all ten runs.	194
91.	Heritrix, PhantomJS, and wget frontiers as an Euler Diagram. The overlap changes depending on how duplicate URIs are identified.	195
92.	Frontier size and crawl speed are dependent upon seed size, with PhantomJS creating a larger crawl frontier but running more slowly than wget and Heritrix. Note that the plotted dots are measured performance, while the lines are predictions.	196

93.	A generic, three-level client-side state tree with interactions as state transitions.	209
94.	Example tree of http://www.bloomberg.com/bw/articles/2014-06-16/open-plan-offices-for-people-who-hate-open-plan-offices . Mouseover events lead to multiple descendants at s_1 and further mouseover events lead to descendants at s_2 , each requiring Ajax requests for JSON and image resources. Please refer to Figures 95 and 96 for larger representations of this Figure.	state 212
95.	Mouseover events attached to the menu bar lead to Ajax requests for JSON and images to build submenus. Note that this is an up-close view of the left side of Figure 94.	213
96.	Click events attached to embedded advertisements and the comment section of the page lead to Ajax requests for JSON to serve new advertisements and to sort the comments. Note that this is an up-close view of the right side of Figure 94.	214
97.	Crawling s_1 provides the greatest contribution to RP ; the additions to the crawl frontier by s_0 (10,623) and s_2 (10,208) only differ by 415 URIs. .	220
98.	Embedded resources discovered in s_1 and s_2 are much more frequently unarchived (92% and 96%, respectively) than s_0 (12% unarchived).	223
99.	Various mime-types of embedded resources are specific to deferred representations. Data, text, and new JavaScript is loaded by JavaScript into the deferred representations.	224
100.	The occurrence of embedded resources loaded into deferred representation descendants.	225
101.	Contributions of each URI-R and its descendants to RP	226
102.	JSON metadata to be added to WARCS.	228

CHAPTER 1

INTRODUCTION

The Web has become an ubiquitous utility. Once a network tethered to a desktop machine, the Web is now accessible via mobile devices and is part of everyday life. With the increasing ubiquity of the Web, users demand increasing responsiveness and personalization, causing Web resources to change more frequently. Web resources that change more frequently are shown to be more popular [3]. As a result of this frequent change, Web pages are ephemeral, and new versions of pages overwrite prior versions, effectively removing them from Web history unless otherwise stored or archived.

How well can we archive the Web? This is a question that is becoming more important and more difficult to answer. Additionally, this question has significant impact on Web users [178, 184] and commercial and government compliance [204, 203, 285]. As the Web increases in importance and prevalence, the importance of Web archiving also increases. Our modern social discourse ubiquitously involves the Web. Any future discussion of the early 21st century will have to involve the Web and Web archives.

The same technologies that enable interactive, personalized, and appealing content for Web users make archiving more impactful since users are placing expanded emphasis on this content and using it as an increasingly important information medium. Ironically, the technologies that are accelerating the importance of Web archiving also make it more difficult to capture Web content. We will discuss *how* these technologies operate and *why* they are increasing archival difficulty.

To discuss the technologies, Web content, and difficulties we will solve in the work discussed in this dissertation, we will use Memento Framework terminology to establish a common vocabulary. Memento [292] is a framework that allows Web users to browse in the temporal dimension by aggregating the offerings of Web archives at a single point of access. Original (or live Web) resources are identified by URI-Rs¹, and archived versions of URI-Rs are called *mementos* and are identified by URI-Ms.

¹URIs, or Universal Resource Identifiers [37], are a generic form of URLs, or Universal Resource Locators.

Memento TimeMaps are machine-readable lists of mementos (at the level of single-archives or aggregation-of-archives) sorted by archival date, or memento-datetime.

To begin, we discuss the importance of the Web archives and the societal impact archives have on our understanding of world events (Section 1.1). We do this using analogies from Lanier’s *You Are Not A Gadget* [161] and examples from the archives’ capture of the September 11th, 2001, terrorist attacks. Next, we will discuss the Stop Online Piracy Act (SOPA) protests of 2012 (Section 1.2). While we remember the SOPA protests and the associated pages on the Web, the archives do not. That is, the archives did not adequately capture the pages protesting SOPA. If we – as Web users – remember SOPA but archives do not, it raises the question: “What else have we forgotten that the archives have also forgotten?”

1.1 YOU ARE NOT A GADGET

The September 11 attacks (also referred to as September 11, September 11th, or 9/11) were a series of four coordinated terrorist attacks launched by the Islamic terrorist group al-Qaeda upon the United States in New York City and the Washington, D.C. metropolitan area on Tuesday, September 11, 2001. In total, almost 3,000 people died in the attacks, including the 227 civilians and 19 hijackers aboard the four planes. It also was the deadliest incident for firefighters in the history of the United States. The United States responded to the attacks by launching the War on Terror and invading Afghanistan to depose the Taliban, which had harbored al-Qaeda.

The above recount is one record of the September 11th attacks as quoted from Wikipedia² in 2015, well after the attacks occurred. *The New York Times* published an article detailing the attacks as new reports were available in 2001. Figure 1 is a static representation – or memento – of an image of the newspaper. The Web archives contain an interactive record of the Web page as it existed in 2001; this is a much more emotive recount of the events than the Wikipedia article. We find the first CNN.com record of the terrorist attacks in Figure 2(a). On this page, a Web user can click on the first link, showing the plane hitting the first tower in Figure 2(b). The user can navigate to President George W. Bush vowing to find and bring

²http://en.wikipedia.org/wiki/September_11_attacks

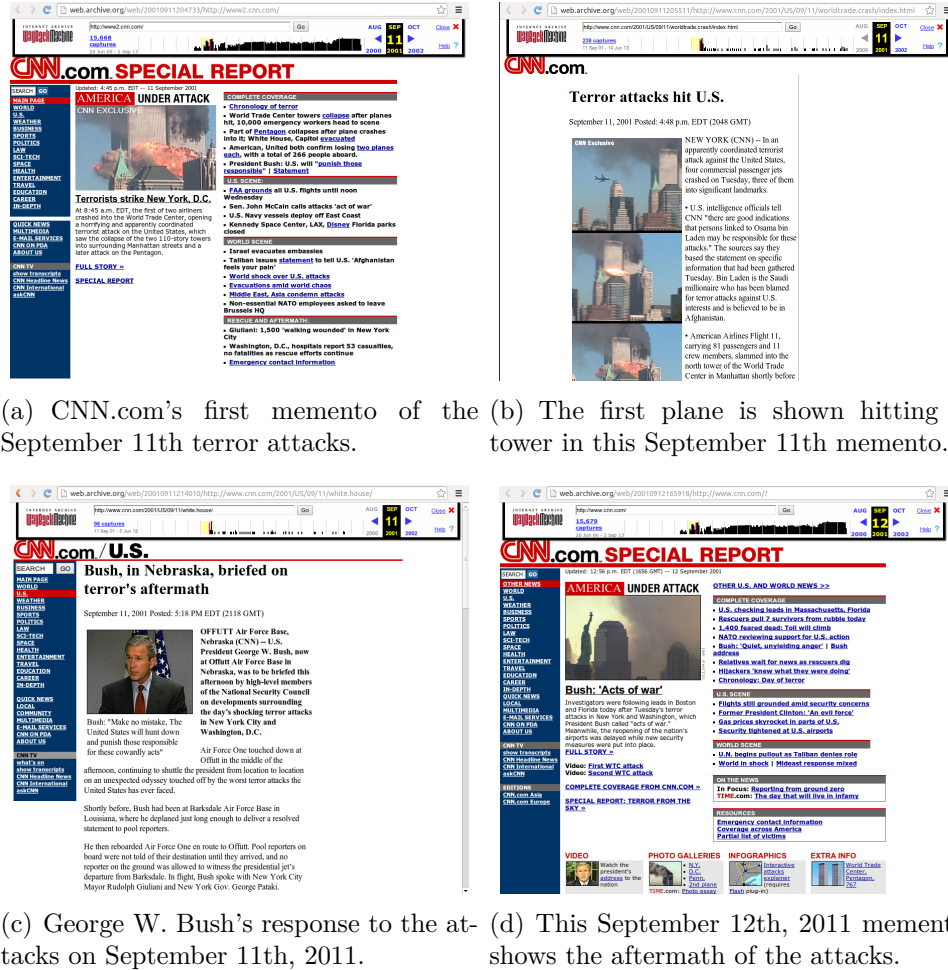


FIG. 2: September 11th mementos invoke a different experience than Wikipedia.

to justice those responsible for the attacks in Figure 2(c). We follow the report through the next morning, with continuing reports from Figure 2(d). This is only one experience we can have using the archives.

Our experiences differ based on which versions of the records we view. The archives are emotive and interactive, allowing the user to explore avenues of the account, constructing new or different experiences from the narrative. The *New York Times* article is interactively limited but emotive, and the Wikipedia article is factual and from a neutral point of view. My experience differs from those of others viewing the article. For example, Keith Reck was in the Pentagon during the attack [284] and his experiences are different than mine, because I learned of the attacks while in a programming class, safe from immediate

danger. Without these mementos, we have no interactive and emotive record of the attacks.

Current mementos of events of similar magnitude, such as the Arab Spring, exist, but are not always exactly how they looked when they were shared. Fellow Web Science and Digital Libraries (WS-DL)³ research laboratory student Hany SalahEldeen⁴ shared news of the revolution on his Facebook page, shown in Figure 3(a), but this resource is not available in the archives, as shown in Figure 3(b). His commentary and interactions – the first- and second-person account of history – are missing from current Internet memory. The screenshot in Figure 3(a) only exists because we understood the importance of SalahEldeen’s commentary and manually captured the image. However, there is no record of the interaction in the public archives.

Other historical events *have* been effectively captured in the archives, such as the MH17 crash [165]⁵. This is evidence that otherwise would be lost were it not for the presence of the Internet Archive.

Jaron Lanier’s book *You Are Not A Gadget* [161] discusses the impacts of technology on our daily lives and on society as a whole, and his discussions are applicable when understanding how different versions of Web resources change in meaning. He focuses on Web services and technologies and how they impact the way we think, use technology as a tool, and influence the experiences we have, just as emotions, experiences, and collective information change based on how we view mementos of the September 11th attacks.

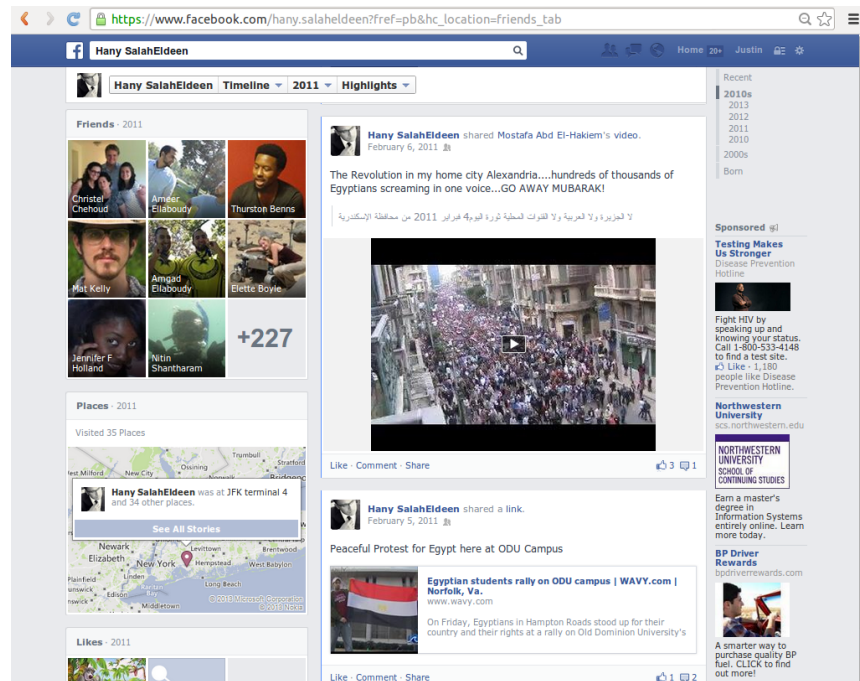
A writer like me might choose to publish a book on paper, not only because it is the only way to get decently paid at the moment, but also because the reader then gets the whole book at once, and just might read it as a whole.

When you come upon a video clip or picture or stretch of writing that has been made available in the Web 2.0 manner, you almost never have access to the history of the locality in which it was perceived to have meaning by the anonymous person who left it there. A song might have

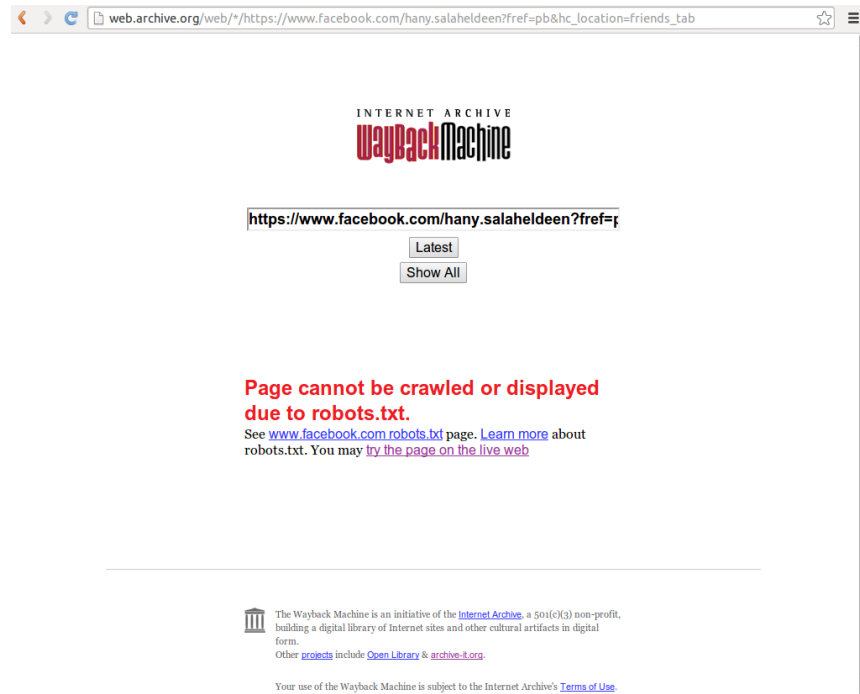
³<http://ws-dl.blogspot.com>

⁴<http://www.cs.odu.edu/~hany/>

⁵Coincidentally, during the authoring process of this dissertation on March 22, 2015, the original New Yorker article returned an HTTP 502 error, making it unreadable (and potentially uncitable) without access to the Internet Archive’s memento of the page at URI-M <http://web.archive.org/web/20150321182718/http://www.newyorker.com/magazine/2015/01/26/cobweb>.



(a) SalahEldeen shares news of the Egyptian Revolution on Facebook.



(b) SalahEldeen's shared news is not available in the archives.

FIG. 3: Hany SalahEldeen's Facebook page is not represented (i.e., not archived at all due to Facebook's use of the robots.txt protocol) in the archives.

been tender, or brave, or redemptive in context, but those qualities will usually be lost. [161]

Lanier’s commentary on an author’s choice of publication medium applies to content authors on the Web; the original information authored on the Web is just as important as traditional publication media. However, the content and published work does not exist in isolation, but exists with surrounding context.

Even if a video of a song is seen a million times, it becomes just one dot in a vast pointillist spew of similar songs when it is robbed of its motivating context. Numerical popularity doesn’t correlate with intensity of connection in the cloud. ... Context has always been part of expression, because expression becomes meaningless if the context becomes arbitrary. You could come up with an invented language in which the letters that compose the words to John Lennon’s ‘Imagine’ instead spell out the instructions for cleaning a refrigerator. Meaning is only ever meaning in context. [161]

Lanier mentions that the context of a Web page is dependent upon the medium, environment, and surrounding influences to create the meaning⁶. In this dissertation, we are attempting to capture this context within a static and stateless environment, using only the factors influencing the representation Web users see. Additionally, the pages without context are meaningless and exist as mere small nodes as opposed to the higher-order understanding or impact of a page when viewed *with* context. Lanier goes on to say:

The distinction between first-order expression and derivative... First-order expression is when someone presents...new in the world. Second-order expression is made of fragmentary reactions to first-order expression. [161]

This shows the need to construct pages and their content based on partial captures (i.e., the interactive CNN.com memento composed of aggregate interactions and environmental conditions with the URI-M). Lanier explains that there are forms of expression in which someone constructs a new, unique idea – a new concept. We

⁶Lanier also alludes to deferred representations (discussed in depth in Chapter 2).

can equate a new form of expression to the first, live publication of a Web page – this is the author’s unique, original idea being presented to the world. Lanier refers to this as first-order expression. Figure 3(a) shows the shared first-order expression in SalahEldeen’s post; the video that SalahEldeen shares is a first-order expression created by the author, rather than by SalahEldeen.

Second-order expression is constructed from select components of the original ideas. Along with the video he shared, SalahEldeen provides his commentary about the first-order expression in Figure 3(a). In aggregate, we have additional information regarding the first-order expression through the accompanying commentary. This aggregate information is important to understanding the context in which Web resources are shared and viewed – independently, the video or SalahEldeen’s commentary may provide a different expression than the resources in tandem.

Deferred representations (discussed further in Chapter 2) are second-order expressions because they are composed of the original resource and are modified by user interaction, client-side events, and other inputs beyond those which the archives and crawlers can observe or represent. These second-order expressions are missing from the archives.

We aim to mitigate the challenges with archiving second-order expressions – we hypothesize that we can capture second-order expressions in ways currently prohibited by the capability gap between archival tools and Web application and browser technologies. Users can experience a content author’s first- and second-order expression, but the archives can only archive a subset of first- and second-order environment and associated expression. We propose a framework to allow archives to capture first- and second-order expressions.

We present Lanier’s view on information and expression as a metaphor for the evolution of information on a Web page. Content authors create information – or representations – and that information can exist independently or in aggregate of additional context information, similar to a representation as it exists when first presented to a user (first-order expression) versus when it has been acted upon by environment variables or other client-side events (second-order expression). In this dissertation, we present a method for crawlers to archive the second-order expressions that are currently unarchivable at Web scale.

1.2 #SOPABLACKOUT: A CASE STUDY

In an attempt to limit online piracy and theft of intellectual property, the U.S. Government proposed the SOPA [310]. This act was widely unpopular. On January 18, 2012, many prominent websites organized a world-wide blackout of their websites in protest of SOPA [86, 229].

While the attempted passing of SOPA may end up being a mere footnote in history, the overwhelming protest in response is significant. This event is an important observance and should be represented in our Web archives. However, some methods of implementing the protest (such as JavaScript and Ajax [103, 180], discussed further in Chapter 2, Section 2.2.3) made some sites' demonstrations unarchiveable by archival services and tools at the time. We examine the Washington, D.C. Craigslist site and the English Wikipedia page as case studies. We took screenshots of the live protests during the protest on January 18, 2012. We took the screenshots of the mementos on November 27, 2013.

1.2.1 CRAIGSLIST

Craigslist⁷ put up a splash page in protest of SOPA that would only provide access to the site through a link that appears after a timeout (Figure 4). In order to preserve the SOPA splash page on the Craigslist site, we submitted the URI-R for the Washington D.C. Craigslist page (<http://washingtondc.craigslist.org>) to WebCite [85], a page-at-a-time archival service, producing a memento for the SOPA screen (URI-M 1 in Table 1), shown in Figure 5(a).

At the bottom of the SOPA splash page, there is a countdown from 10 to 1, and then the page provides a link to enter the site. This behavior is shown in a YouTube video at <http://www.youtube.com/watch?v=QZhUN1WI6ZQ>. The countdown operates properly in the memento, providing an accurate capture of the resource as it existed on January 18, 2012, because the JavaScript enabling the countdown was properly archived.

The countdown behavior is archived along with the page content and is available when users view the memento. However, the link that appears at the bottom of the screen directs the user to the live version of Craigslist. Notice that the live Craigslist page has no reference to the SOPA protest (Figure 5(b)). Since WebCite

⁷<http://craigslist.org/>

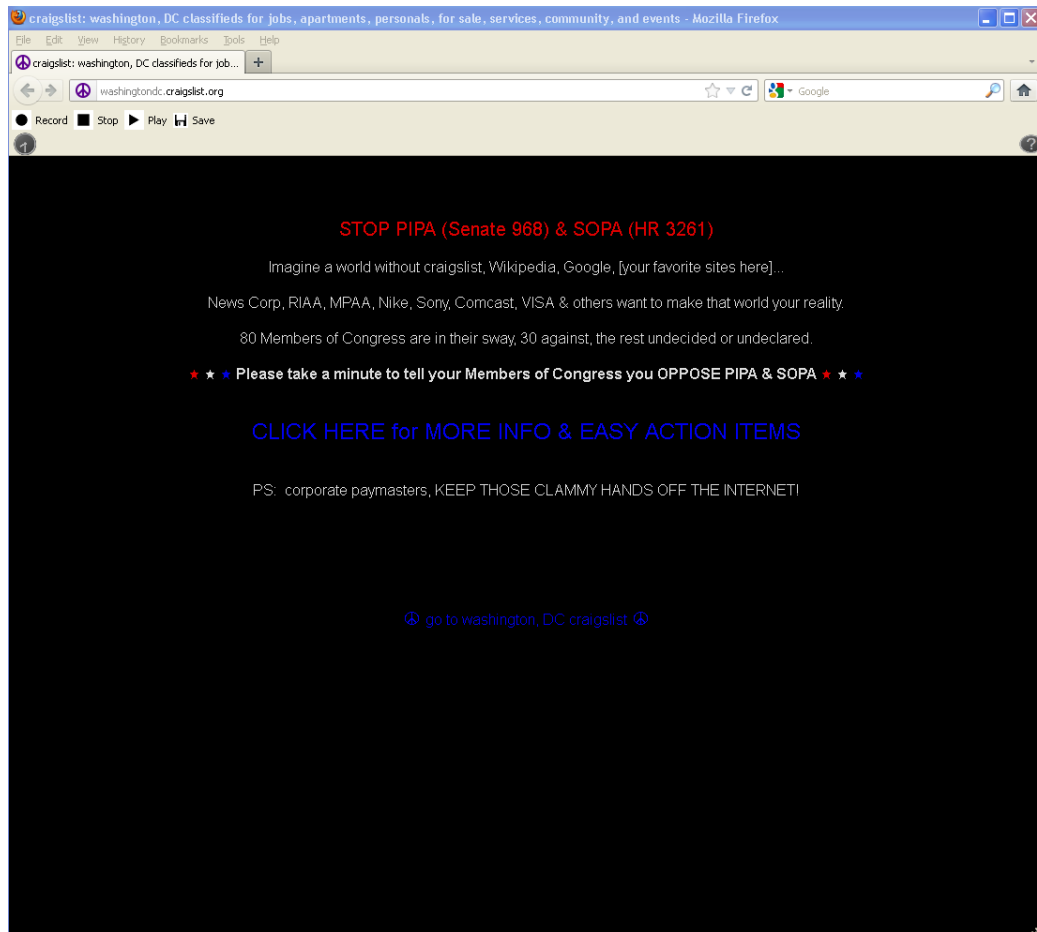
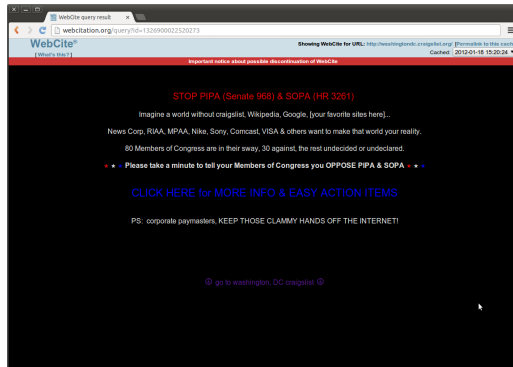
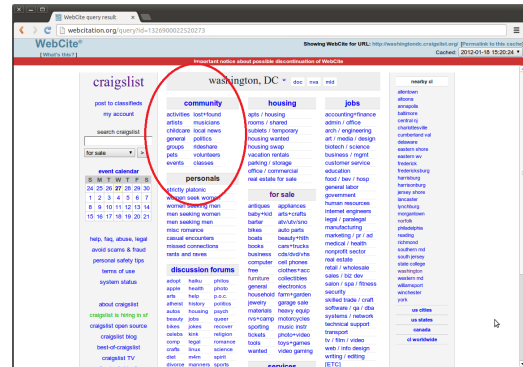


FIG. 4: A screenshot of the SOPA blackout on the Washington D.C. Craigslist site.



(a) The WebCite memento of the SOPA blackout on the Washington D.C. Craigslist site.



(b) The link presented after the countdown dereferences to a live WebCite resource.



(c) The Internet Archive memento of the SOPA blackout on the Washington D.C. Craigslist site.



(d) The link presented after the countdown dereferences to a memento of the Craigslist page during the protest.

FIG. 5: Mementos of the `craigslist.org` protest of SOPA contain evidence of the protest and their associated targets.

is a page-at-a-time archival service, it only archives the initial representation and all embedded resources, meaning the linked Craigslist protest page is missed during archiving because it did not exist until JavaScript inserted it into the representation after the 10 second countdown after the initial page load.

TABLE 1: Replay behavior of mementos of the Craigslist SOPA protest.

ID	URI-M	Memento-Datetime	Accuracy of Replay
1	http://www.webcitation.org/64momE9I1	January 18, 2012	Properly archived protest page but redirects to live Craigslist page
2	http://web.archive.org/web/20120118050348/http://washingtondc.craigslist.org/	January 18, 2012	Properly archived protest page and redirects to archived Craigslist page (URI-M 4)
3	http://wayback.archive-it.org/all/20120119183432/http://washingtondc.craigslist.org/	January 18, 2012	Properly archived protest page and redirects to archived Craigslist page (ID 4 in this table)
4	http://web.archive.org/web/20120120201008/http://washingtondc.craigslist.org/h	January 20, 2012	Properly archived Craigslist page (albeit a day later than the actual protest, but with identical content)

The Internet Archive [205, 287] contains a memento of the Craigslist page on January 18, 2012 (URI-M 2 in Table 1), as shown in Figure 5(c). The Archive-It service also has a memento of the protest at URI-M 3 in Table 1.

The Internet Archive memento has the same splash page and countdown as the WebCite memento. The link on the Internet Archive memento leads to a memento of the Craigslist page (at URI-M 4 and Figure 5(d)) rather than the live version, albeit with archival timestamps one day and 13 hours, 30 minutes, and 44 seconds apart (2012-01-20 18:34:32 vs 2012-01-18 05:03:48).

The Internet Archive converts embedded links to be relative to the archive rather than target the live Web. Because the Internet Archive also archived the linked page, the user receives the proper memento with a note embedded in the HTML protesting SOPA when clicking on the link.

The Craigslist protest was readily archived by WebCite, Archive-It, and the Internet Archive. Policies within each archival institution impacted how the Craigslist homepage (past the protest splash screen) is referenced and accessed by archive users. This differs from the Wikipedia protest, which was not readily archived.

1.2.2 WIKIPEDIA

Wikipedia⁸ displayed a splash screen protesting SOPA blocking access to all content on the site (Figure 6(a)). A URI-R replicating the protest page is still available live on Wikipedia as of October 27, 2015, at <http://en.wikipedia.org/?banner=blackout> (Figure 6(b)).

On January 18, 2012, we submitted the page to the WebCite archival service to produce a memento (URI-M 1 in Table 2) that did not capture the splash page (Figure 7(a)). Instead, the memento shows only the content meant to be hidden by the splash page.

TABLE 2: Replay behavior of mementos of the Wikipedia SOPA protest.

ID	URI-M	Memento-Datetime	Accuracy of Replay
1	http://webcitation.org/query?id=1326888962288259	January 18, 2012	No splash page captured
2	http://web.archive.org/web/20120118110520/http://en.wikipedia.org/wiki/Main_Page	January 18, 2012	Splash page captured, but not loaded in the memento
3	http://wayback.archive-it.org/all/20120118184432/http://en.wikipedia.org/wiki/Main_Page	January 18, 2012	Splash page captured, but not loaded in the memento
4	http://web.archive.org/web/20120118165255/http://upload.wikimedia.org/wikipedia/commons/9/98/WP_SOPA_Splash_Full.jpg	January 18, 2012	Memento of the splash page

The mementos captured by Heritrix and presented through the Internet Archive’s Wayback Machine (URI-M 2 in Table 2 and Figure 7(b)) and Archive-It (URI-M 3

⁸<http://www.wikipedia.org/>

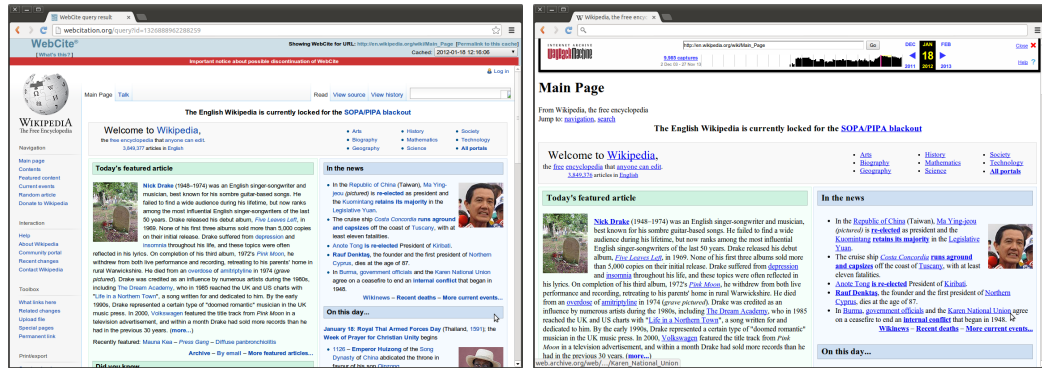


(a) A screenshot of the Wikipedia SOPA protest taken during the protest in 2012.



(b) A screenshot of the live recreation of the blackout page.

FIG. 6: Screenshots of the live versions of the Wikipedia blackout in protest of SOPA.



(a) The WebCite memento of the SOPA (b) The Internet Archive memento of the
blackout on the Wikipedia site. SOPA blackout on the Wikipedia site.



(c) The Archive-It memento of the SOPA
blackout on the Wikipedia site.

FIG. 7: Mementos of the Wikipedia.org protest of SOPA do not show the blackout splash page.

and Figure 7(c)) are also missing the SOPA splash page.

The blackout image is available in the Internet Archive, but the mementos in the Wayback Machine do not attempt to load it (URI-M 4) because the method of loading the splash page – JavaScript – causes crawlers and archival services to miss the splash page. As a result, the archives do not contain an accurate depiction of the protest by Wikipedia.

We have presented two different uses of JavaScript by two different Web sites and its impact on the archivability (or, the ease of archiving a resource) of their SOPA protests. The Craigslist mementos provide depictions of the SOPA protest, although the archives may be missing associated content (i.e., second-order expression) due to policy differences and intended use. The Wikipedia mementos do not provide an accurate memento of the protest because they rely on JavaScript to display the splash page. While the examples in Section 1.1 were adequately archived in 2001, the examples in this section from 2012 show inadequacies arising in archival attempts. This is because the archives are attempting to archive 2012 content with tools tailored for the Web of 2001. Without a change to current archival frameworks, important Web pages like these will continue to be improperly archived by archival institutions.

1.3 MODERN WEB ARCHIVING

The importance of Web archiving is increasing, as is the average Web user’s general awareness of archival efforts [178] (as evidenced by recent articles in *The New Yorker* [165], *The Christian Science Monitor* [44], NPR [306], and *The Atlantic* [158]). Users are browsing specialized archives (such as the September 11th archive [283]) and interacting with the Internet Archive. The capture and archive of Web resources is important for historical purposes and records management compliance, capturing information that would otherwise be lost due to the ephemeral nature of the Web [204, 203, 285, 165].

Archives currently hold a version of the Web from the point of view of the crawlers, not the way it is experienced by users. Crawlers are responsible for discovering and capturing Web content. As we explore in this dissertation, crawlers are not capable of uncovering *all* content because much of it is only accessible when interacting with content using methods crawlers cannot perform.

Web crawlers operate by starting with a finite set of seed URI-Rs in a frontier – or list of crawl targets – and add to the frontier by extracting embedded resources and

URI-Rs in the representations returned upon dereferencing the URI-R. This allows archival crawlers to discover embedded resources as well as new URI-Rs to crawl. Crawlers request a page and archive the response. This pattern is fundamentally different from how users interact with the Web, which goes beyond requesting a page and consuming the response to interacting with, altering, and constantly consuming the intermediate content.

The state-of-the-art crawlers, archival tools, and “major players” involved with Web archiving include the Internet Archive’s Heritrix crawler [264, 201], WebCite, the `wget` application [104], WARCcreate [142, 149], WAIL [143], Archive.is [18]⁹, Webrecorder.io [156], py-wayback [105], and many more. These tools are effective when capturing single-state Web content such as HyperText Mark-up Language (HTML). However, rapid advances in Web technologies (such as JavaScript, HTML5, Flash, SilverLight, DHTML, and other embedded media) are transitioning Web *pages* (primarily composed of HTML) toward Web *applications* or Rich Internet Applications (RIAs) (composed mainly of JavaScript or other client-side technologies delivered in an HTML container). Web *applications* are not readily archived. Web *applications* are tailored to users and depend on multi-state, personalized content, session data, and user interactions to render a final representation. This topic is discussed in depth in Section 2.3. Web *applications* are designed to provide content based on user interactions and can exist in many states based on the context or environment variables and user interactions performed.

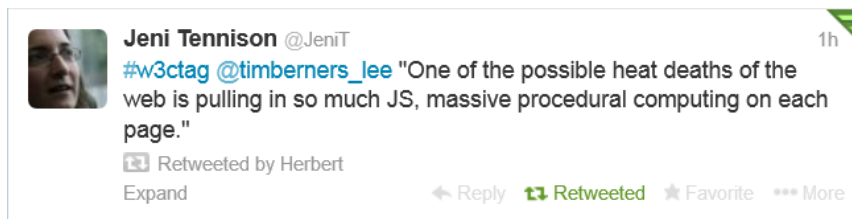
As previously mentioned (Section 1.2), current archival tools were developed to archive Web *pages* – just like the Web as it existed in 2001. These tools perform adequately on HTML-based pages. The tools have not advanced at the same pace as the current JavaScript-based pages that exist in the current Web, and technologies such as JavaScript make archiving Web *applications* in 2015 difficult with the tools of 2001. The expected advancement of the technologies enabling Web *applications* will continue to create archival challenges for crawlers without a strategy for adapting to current and expected technological challenges.

Web pages are increasingly utilizing external data and resources to generate the content provided to a user. When loaded into the page via JavaScript, such data is not accessible to crawlers. Resources that operate as Web applications (such as

⁹During the authoring of this dissertation, the name and URI of this service changed between Archive.is to Archive.today and back to Archive.is. As such there may be inconsistencies in the naming conventions in the cited resources and documents.



(a) A tweet about the importance of accessibility on the Web from multiple clients.



(b) A tweet about Tim Bernes-Lee mentioning the detrimental impact of JavaScript on the Web.

FIG. 8: Tweets showing the increasing understanding of archival challenges and need for paradigm shifts in archiving.

social media sites) that are generated by technologies such as JavaScript are typically difficult – or impossible – for crawlers to capture properly, including for archival purposes. As a result, many historically significant artifacts are being lost because of the ephemeral nature of Web data [254, 21, 52].

Even those artifacts we currently view as ephemeral and largely unimportant have significant cultural and historical value. Advertisements embedded in Web pages may seem unworthy of archival efforts, but vintage ads – by today’s standards – are considered valuable for cultural analysis as well as informational importance. For example, an analysis of cigarette ads from the 1920s shows a change in the health perception of cigarettes over time [304], and an investigation of ads from the 1800s through the 1980s shows the evolution of consumer behavior [114].

Current Web archives contain the view of the Web from the point of view of Web crawlers. Web crawlers and Web users experience the Web differently; the general use-case for crawlers involves dereferencing a URI and capturing the payload returned (e.g., HTML Document Object Model (DOM)), while Web users use Web browsers to dereference the URI, render the payload returned, and interact with a

representation¹⁰. Web crawlers experience the Web spatially (navigating between different URIs) and temporally (experienced at different times), while users experience the Web with the added dimension of environment variables, interactivity, and with stateful client-side representations in the browsing experience.

Web technologies are used to improve or enhance the user’s experience, but crawlers have not kept up with changing Web technologies to effectively archive interactive, deferred representations that users experience and have come to expect. Such Web applications rely on data and environmental variables (such as user interactions, GeoIP, or composite data) to construct state transitions on the client and reach a final representation; crawlers cannot replicate this behavior (described in Figure 8(a)). In short, Web crawlers do not have the ability to archive deferred representations (that is, representations constructed by JavaScript; we discuss this concept in depth in Chapter 2) automatically and at Web scale.

1.4 RESEARCH QUESTIONS

Our goal is to address the divide between what Web users see and what crawlers can archive by mitigating the impact of client-side technologies on archivability. In this dissertation, we define a framework for identifying, recording, and archiving the user experience and context of Web pages. We establish a foundational understanding of the archival challenges created by JavaScript¹¹ and other client-side technologies as well as the extent to which these technologies impact the archives. We propose a framework for mitigating the negative impact of JavaScript on the archives, measure the performance of the framework, and provide an analysis of the improvement that the archives can expect if implementing this framework.

Our proposed framework will improve automatic Web archiving by memory institutions like the Internet Archive, as well as improve information retrieval efforts by automatically detecting and uncovering deferred representations. This will mitigate the impact of JavaScript on the Web archives.

The research in this dissertation addresses the following questions to understand the current challenges and to better construct a framework to solve them:

¹⁰We discuss dereferencing URIs in Section 2.1 and the process of using a crawler to archive the Web in Section 3.1.

¹¹We discuss JavaScript in Section 2.2.3.

1. To what extent does JavaScript impact archival tools?

Before discussing a framework for archiving deferred representations, we measure the correlation between the increased historical adoption of JavaScript as a method of loading embedded resources and missing embedded resources in the Internet Archive. We also measure how JavaScript impacts the wget, WebCite, and Heritrix archival tools. In short, we provide evidence of the extent of archival challenges JavaScript creates as well as detail the impact it has on archival tools (Chapter 5).

2. How do we measure memento quality?

We show that the simple calculation of the proportion of missing to successfully dereferenced embedded resources in a memento does not match Web users' perception of memento quality. To resolve this discrepancy, we identified a more effective measurement of memento *damage* that more closely aligns with users' perception of quality. With this work (Chapter 6), we provide an improved method of evaluating memento quality. We also identify *leakage* and treat leaked embedded resources as missing since they are not archived and lead to temporal inconsistencies. We also show that mementos of deferred representations have lower quality than mementos of nondeferred representations.

3. How can we crawl, archive, and play back deferred representations?

We propose and measure a two-tiered approach for crawling deferred representations using Heritrix and PhantomJS (Chapter 7). We measure the performance impacts and improved crawler frontier size of the two-tiered approach. We also investigate methods of reducing crawler overhead by classifying deferred representations based on their DOM and only crawling deferred representations with slower but more accurate technologies such as PhantomJS. We propose a framework of methods for discovering and archiving client-side states in deferred representations (which we term *descendants*) by mapping interactive DOM elements to client-side state within representations, enabling crawler interaction with the representations, and understanding how to archive and replay the resulting descendants.

By answering these three research questions, we can mitigate the impact of JavaScript on archive quality and improve archival strategies to move closer to archiving the Web that users experience rather than the Web that crawlers experience.

1.5 DISSERTATION ROADMAP

Before discussing our specific work toward establishing the framework, we explore the evolution of Web technologies (e.g., from HTML to JavaScript) and the operation of the Web (Chapter 2), and discuss the work performed by prior researchers that provided the foundation for our work (Chapter 3). We also discuss our preliminary work toward understanding the challenges that arise when the deferred representations (defined in Section 2.2.3) are archived by automatic crawlers and are replayed incorrectly as mementos, and the important history that is being lost as a result of the continued adoption of JavaScript (Chapter 4).

Our work toward establishing a framework to help improve the ability of Web archives to capture and store Web pages begins with an analysis of how today’s Web crawlers are failing to archive Web pages that rely on JavaScript. We discuss the archival challenges introduced by JavaScript and measure the correlation between the completeness of Web archives and the adoption rate of JavaScript. Through this analysis, we establish and measure a major challenge facing today’s Web archives (Chapter 5), as well as answer Research Question 1 by measuring the impact of JavaScript on Web crawlers.

With the extent of the problem measured, we construct a method of quantitatively measuring the quality of an archived page and, therefore, the collections within an archive to answer Research Question 2. We compare our methodology to the qualitative and quantitative metrics used by archives during their quality assurance evaluations, and show that our measurement more closely aligns with what Web users consider to be higher quality. We also demonstrate that archived Web pages that leverage and rely on JavaScript have more *damage* (i.e., are of a lower quality) than those that do not rely on JavaScript, further establishing the need for an archival framework to crawl and archive Web pages that use JavaScript (Chapter 6).

To begin the discussion of our proposed framework, we propose incorporating new technologies called *headless browsers* into the archival workflow to mitigate the impact of technologies such as JavaScript on the automatic archival crawlers. We

measure the performance (i.e., run-time and ability to archive JavaScript-dependent Web pages) of a traditional Web crawler as-is and compare it to its performance (i.e., discovered frontier size) when incorporating the headless browser. We also propose a classification method to limit the use of the headless browser to only the archival targets for which it is needed. We also measure the amount of interaction within a Web page that is currently invisible to the archival crawlers and propose a method of interacting with and archiving the resulting discovered states of Web pages. We show that this improves archival coverage and completeness and partially answers Research Question 3 (Chapter 7).

Finally, we tie these areas of research together with a two-tiered crawling approach that mitigates the impact of JavaScript on the archives (Chapter 7.2), completing our answer to Research Question 3. We discuss its proposed operation and impact on the archives and conclude with a summary of our findings, future work, contributions, and conclusions (Chapter 8).

CHAPTER 2

BACKGROUND

In this chapter, we briefly introduce a set of topics and concepts necessary to fully understand the preliminary work and proposed framework. This section should be read as a primer to the background information needed to discuss the framework and how it was conceived.

2.1 WEB ARCHITECTURE

Tim Berners-Lee’s Web [34] revolves around client-server interactions. His contributions made global and distributed access of content possible. His contributions eventually evolved from a read/write model (similar to a file system) to the protocols that are observed today in the World Wide Web (WWW). It is necessary to discuss standard client-server interactions in order to understand how these interactions are manipulated to get static and dynamic content and why some of these resources are more easily archived and shared than others.

The overarching purpose of a client-server transaction is to allow a user to request that a server transmit data across a computer network. The most common client-server relationships exist in the context of the WWW over the HyperText Transfer Protocol (HTTP) [93]. This is not the only client-server system, but, for the purposes of this dissertation, we will almost exclusively refer to the specific communication between clients and servers over HTTP. Traditionally, human Web users interact with the Web through a Web browser such as Internet Explorer [198], `curl` [277], Firefox [95], or Chrome [109]. These interfaces provide a mechanism for graphically interacting with the Web by generating the requests and handling the responses to and from Web servers, essentially making them a wrapper for HTTP transactions and obfuscating the transaction from users. The representation is rendered from machine-readable code (such as HTML) into a form consumable by the human users. A local tool to access servers and therefore Web content is referred to as a client (or **user-agent** in HTTP). The client-server interaction is incredibly powerful due to the flexibility of the parameters – or header fields [93] – that are allowed and handled in each message.

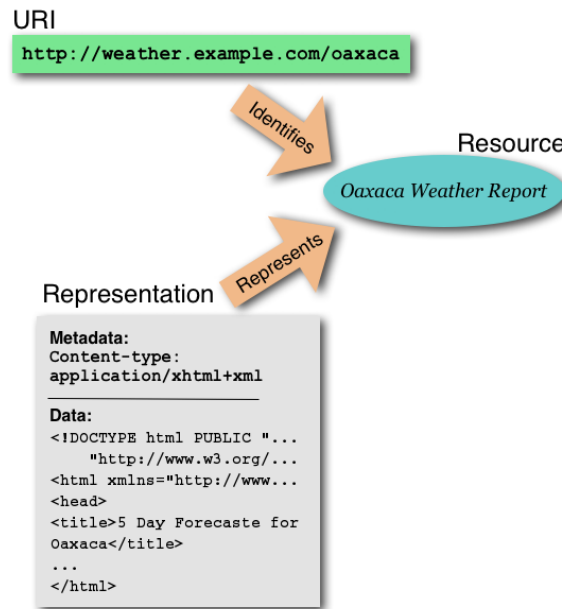


FIG. 9: An example from W3C of the URI-representation-resource relationship [135].

Universal Resource Identifiers (URIs) identify Web resources [298, 35, 38, 36]. A user-agent can dereference a URI [298] to receive a representation of the resource identified by its URI. In an example from the World Wide Web Consortium (W3C) [135] in Figure 9, we see that the Oaxaca weather report (the resource) is identified by its URI, and its state at the time of dereferencing its URI is represented by a document in XHTML markup (the representation).

When the URI is dereferenced, the representation is returned to the user-agent (e.g., a browser). This representation could be, for example, HTML, PDF, a Microsoft Office Document, or multiple formats may be available simultaneously. The user-agent conveys the user’s preferences to the server and allows the server to take these preferences into consideration to provide the most appropriate representation of a resource to the requester [297].

HTTP defines a set of “methods” for client-server interaction. The HTTP GET method is used to retrieve representations of resources. To retrieve representations of Web resources, a client issues a GET request to the server for a particular resource identified by a URI. An example HTTP Request is provided in Figure 10. As observed in this transaction, the client issued a GET request (using HTTP 1.1) to the server, and the server responded with a 200 OK response and content. The response from

```

1 GET / HTTP/1.1
2 Host: www.justinfb Brunelle.com
3 User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux i686; rv:26.0)
4             Gecko/20100101 Firefox/26.0
5 Accept: text/html,application/xhtml+xml,application/xml;
6         q=0.9,*/*;q=0.8
7 Accept-Language: en-US,en;q=0.5
8 Accept-Encoding: gzip, deflate
9 DNT: 1
10 Connection: keep-alive

```

FIG. 10: A HTTP GET Request for `http://www.justinfb Brunelle.com` as captured by Mozilla FireFox.

the server is provided in Figure 11.

There are several potential header strings that the client may provide, such as **user-agent** (which declares the browser or browser-equivalent the requester is using to perform the communication; Line 3, Figure 10), **accept-language** (which sets the preferred language for the representation; Line 5, Figure 10), and **if-modified-since** (which specifies a threshold condition under which the content is or is not returned; not shown in Figure 10). The server receives these values and determines what representation to return. The server responds to a HTTP request with a code specifying the status of the request (i.e., 200 OK for successful requests (Line 1, Figure 11) or 404 - Not Found when the requested resource is not available), accompanying headers, and returned representation. Server responses can include headers such as **last-modified** (the last time the entity was changed on the server; not shown in Figure 11). Entity headers describe the entity such as **content-length** (the size of the transferred entity; Line 5, Figure 11) or **content-encoding** (the encoding/compression method used to encode the entity; not shown in Figure 11). Entity headers can be provided in either the request (when the client is sending an entity) or response header (when the client is receiving an entity).

Berners-Lee defined *time-generic* resources as those that will evolve and change over time [35]. *Time-specific* resources are resources whose state does not change.

Representations correspond to the state of a resource. If a resource changes state, the representation changes to reflect to new state (shown in Figure 9). Dereferencing

```

1  HTTP/1.1 200 OK
2  Date: Mon, 30 Dec 2013 15:57:27 GMT
3  Server: Apache
4  Accept-Ranges: bytes
5  Content-Length: 1737
6  Keep-Alive: timeout=15, max=100
7  Connection: Keep-Alive
8  Content-Type: text/html
9
10
11 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
12   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
13 <html>
14 ....
15 </html>

```

FIG. 11: The HTTP Response for `http://www.justinfbrunelle.com` as captured by Mozilla FireFox.

a URI is a stateless transaction that returns a representation of a resource at a point in time. That is, the same resource can be accessed by different platforms (e.g., mobile, desktop) or environments (e.g., preferred English language, preferred French language, preferred MIME-type), and the appropriate representation is returned. This is a process called *content negotiation* (see Section 14 of RFC 2616 [93], in which multiple kinds of content negotiation are defined). In short, content negotiation is an agreement between a resource provider (e.g., server) and requesting user (user-agent) to return the “best” possible representation of a resource¹. In content negotiation, the server will interpret the user-agent field (and various other Accept headers) in the HTTP header and provide the best (or desired) representation, accordingly; this is a server-side activity, not a decision made by the client².

Recent technologies (described in more detail in Section 2.2) have enabled representations to change outside of the bounds of content negotiation such that representations change based on client-side events occurring after the initial HTTP GET

¹The principles of content negotiation relate well to Lanier’s notion of context.

²RFC 2616 also defines client-driven content negotiation such as prompted actions by the user (e.g., “Select preferred language”).

transaction performed when dereferencing the URI; the server no longer has complete knowledge of the representation seen by the user at the time of dereferencing. These technologies allow for personalization and for stateful client-side representations to become more like Web *applications*. The migration from Web resources to Web applications is the foundational challenge we address in this dissertation (listed as Research Question 3 in Section 1.4): how can the archives adapt their approach to archiving to better archive Web applications that leverage client-side technologies?

2.2 FROM HTML TO RICH INTERNET APPLICATIONS

The evolution from static Web *pages* to Web *applications* has made the Web resources capable of being more personalized and dynamic. Web users have come to expect Web pages to provide personalized and interactive representations [30]. It has also changed the nature of the client-server interactions discussed in Section 2.1. This section briefly describes how these Web technologies have evolved.

2.2.1 HTML

Traditional Web development is done using HTML, which was the *de facto* language of Web representations during the development and growth of the WWW. HTML, a markup language (as the name suggests), exists in a static format in which the HTML page is a representation of a resource that the browser renders. Markup in a Web page can include colors, images, text size and format. Images, Cascading Style Sheets (CSS), and other Web native resources can be included in pages to enrich the user’s experience, and we refer to these as *embedded resources*. Pages written in the same HTML result in identical DOMs. The HTML DOM is the tree representation of the tag elements in a page’s HTML. An example DOM is listed in Figure 12(b) for the code listed in Figure 12(a).

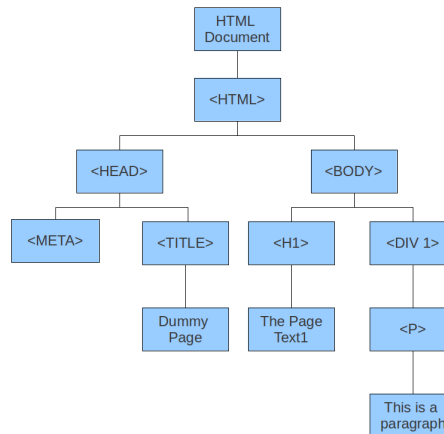
The user-agent (e.g., Mozilla Firefox and Google Chrome browsers) renders the DOM visually for the user. This DOM changes neither in value nor representation during the session. Upon calling a new page, the browser interprets the representation to display the new DOM. It is clear that the representation and content only changes when a content author changes the DOM delivered from the server. This ensures that user interactions remain constant when accessing representations by URI. Effectively, the same HTML, image, stylesheet, and other documents combine for the same representation between user accesses.

```

<HTML>
<HEAD>
<META NAME="robots" CONTENT="noarchive">
<TITLE>Dummy Page</TITLE>
</HEAD>
<BODY>
<H1>The Page Text1</H1>
<DIV ID=1>
<P>This is a paragraph</p>
</DIV>
</BODY>
</HTML>

```

(a) The HTML code for the default page.



(b) An example HTML DOM for a default page

FIG. 12: The DOM is a tree representation of the HTML.

The single-state nature of HTML lends itself well to caching, assuming the resource does not change state. Because the content is static and only requires the DOM to render the representation, the browser can faithfully cache the page contents and present the cached copy to the user; the DOM, embedded resources, and URI are necessary to provide an accurate cached reproduction of the page. Similarly, Web crawlers can easily reach and extract the content from the page, placing it on the surface Web [32]. HTML-only resources are indexable by search engines and can be archived with extremely high fidelity due to their consistent single state in time. It is (relatively) trivial to record the state of an HTML-only page because the representation does not change after it is dereferenced by the client. These points should be intuitive, but they are important to establish for reference and comparison in future sections of this dissertation.

2.2.2 REST PRINCIPLES

The concept of Representational State Transfer (REST) was proposed by Roy Fielding in his dissertation [94]. His propositions include a method of referencing resources as concepts and allow the server and client to negotiate on the representation. REST allows resources to exist independently of their representations. Essentially, a URI should identify a particular resource. Requesting and dereferencing a different URI is a stateless activity, meaning dereferencing a URI depends only on the HTTP

request (including the target URI) and not client state. As such, dereferencing the same URI with the same HTTP request and environment variables (such as those that will lead to content negotiations) at the same time from two separate user-agents *should* provide the same representation.

REST also relies on the URI to provide any information the resource needs to generate the desired state or representation. That is, the client should not need to store any state or data in order to provide a representation. Further, this assumes a representation of a resource can be cached by URI, or even stored “offline.” The state displayed is in the perpetual now. For example, we can provide an identifier of Justin F. Brunelle’s homepage `http://www.justinfbrunelle.com/` that is always current at dereference time and provides a representation of Justin’s homepage. The browser and server negotiated on the format of HTML for this representation.

REST and RPC/SOAP are two competing models. Remote Procedure Calls (RPCs) in Simple Object Access Protocol (SOAP) [299] use HTTP to perform actions on behalf of a client. However, RPCs communicate system state and environment features through the payload – or *envelope* – of the HTTP request. This could be a POST with associated XML information that is sent to the server by the requesting user [270]. In RPC, users reference resources to perform *actions* rather than using HTTP to perform actions on the resources themselves.

2.2.3 JAVASCRIPT AND AJAX

HTML alone is insufficient for providing the functionality and interactivity that modern Web users have come to expect [111]. Web documents have migrated from static, single state documents to dynamic RIAs. By incorporating JavaScript, pages are becoming more like desktop applications than their single-state, HTML-only predecessors. JavaScript is a programming language that enables client-side state transitions and multi-state representations dependent upon external resources or user interactions [98]. Client-side scripting languages run on the client (e.g., in the browser) and can change, add, and request new content after the original representation is delivered by the server. Such technologies and languages have emerged and are now essential to increase the functionality and capabilities of client-side Web pages.

JavaScript is a client-side programming language that enables scripting-style and object-oriented programming from within an HTML page. JavaScript can be embedded in HTML pages using the `<script language="JavaScript">` tag. JavaScript

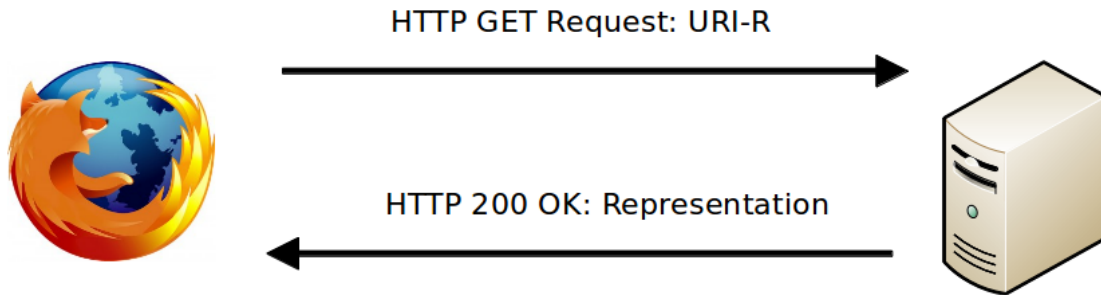


FIG. 13: Normal request-response interaction between client and server for a resource R.

can exist embedded in the page as well as be loaded from an external file. The embedded code has nearly limitless capabilities; JavaScript can communicate with servers, the browser, redirect to or call other URIs, interact with the server- and client-side data (including cookies and session information), and interact with the DOM. JavaScript uses an asynchronous, event-driven operation style; code and functions are executed at the occurrence of events. The source of these events can vary, including the invocation by user or server actions. Embedded code can also execute when a page loads or a server issues a command.

Normal browsing sessions occur as depicted in Figure 13. The browser issues an HTTP request for a resource, and the server provides the resource to the browser. The browser then renders the representation. However, when requesting an interactive resource, the browsing session will make an initial HTTP request for the main content, followed by several requests for embedded resources. The page is then rendered in the browser. Then, the JavaScript embedded in the page makes additional requests to (or passes and receives data to and from) the server for additional content using a technology called Ajax³, which is a set of interconnected technologies first introduced in 2005 and achieves what was previously known as Dynamic HTML. This content is only retrieved after the main page content is rendered. Figure 14 illustrates this alternate sequence.

JavaScript is interpreted by a JavaScript engine embedded in the Web browser.

³For the purposes of this dissertation, we refer primarily to JavaScript with the understanding that Ajax may be responsible for initiating requests for embedded resources.

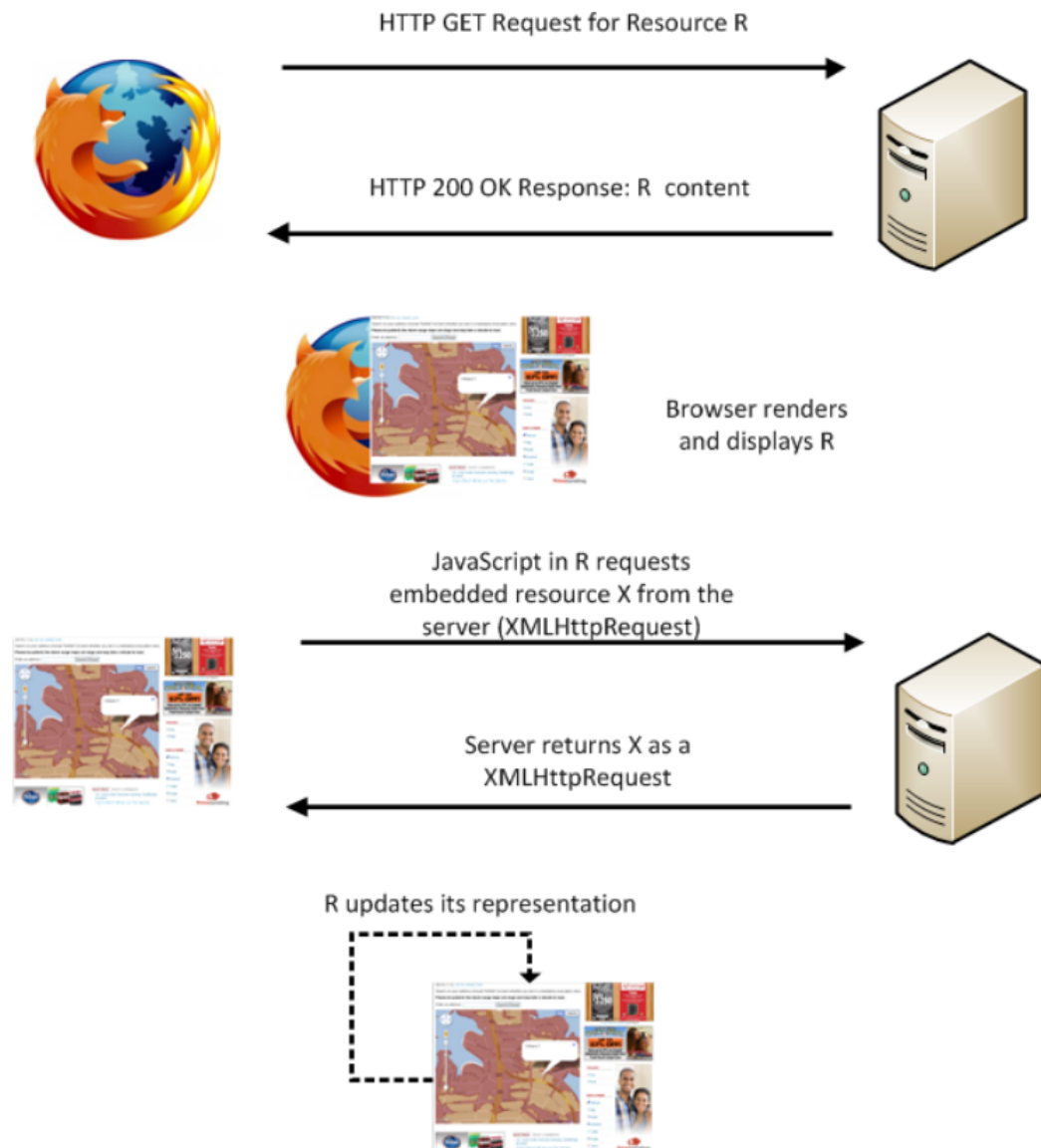


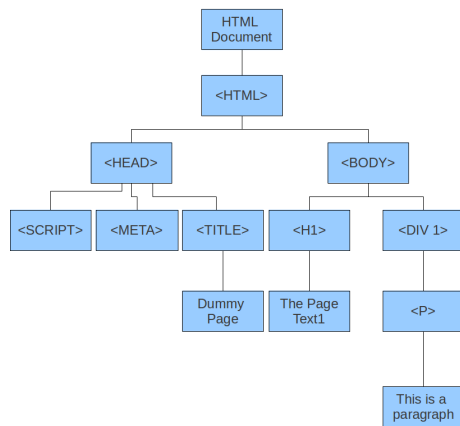
FIG. 14: Request-response interactions between client and server and page and server for resource R and embedded content.

Extending the previous example of a local page (Figure 12(b)), we provide an example DOM with embedded JavaScript (Figure 15(a)). When the JavaScript runs on the client (Figure 15(c)), an event fires on the client and can execute a change of the DOM. The event and associated changes are outlined in a red rectangle.

JavaScript can redirect the browser to new URIs, resize the window, and control the browser's appearance and capabilities (such as menus, URL bar, and printing capabilities). More importantly, JavaScript has the capability to modify the local DOM. This can include anything from reading, validating, or modifying user input into the local DOM elements and forms to adding style sheets to directly modifying the DOM. JavaScript can add, remove, and change existing HTML elements. This allows an event (triggered automatically or by user action) that will change the representation content, or meaning, of the loaded client-side page. The interaction between the user and the client-side resource is made possible by Ajax. Ajax allows client-side Web resources to interact with a server to request data after an initial load of the HTML enabling communication and data transfer to and from the server through the `HttpRequest` and `HttpResponse` objects. These objects allow the data necessary for the application to run to be delivered by the server at run-time to the client incrementally. As such, the URI for the page can lead to different representations dependent upon user interactions, data from the server, or session data.

In an Ajax application, the process of displaying content begins just as a normal Web interaction: the page is loaded and the content is printed to the screen for the user. In addition to the static content, the JavaScript code is loaded. Additional events may be fired as a result of page loads or user interactions (See Figure 16 to see the sequence of events). These events may use Ajax to request and receive additional resources from the server. The representation is returned by the server and handled by client-side JavaScript code. There are benefits and drawbacks of utilizing Ajax. The user will gain interactivity through this method – as well as the ability to stream data in the background – but will lose traditional use of the back button and URI to identify the page state. Similarly, users cannot bookmark representations that result from such events. These events create a potentially infinite number of client-side states that we refer to as *descendants* (discussed further in Chapter 7). Crawlers and archivists cannot feasibly capture each state using current Web-scale archival tools.

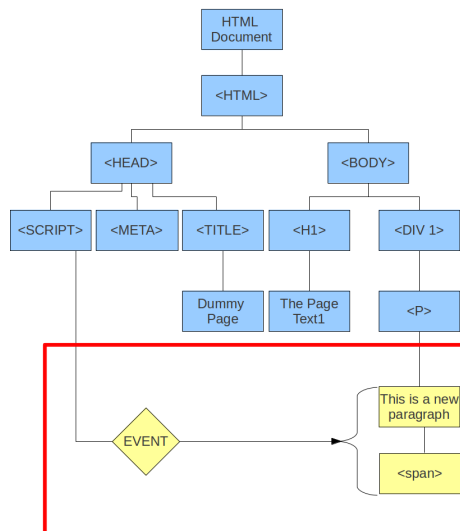
Figure 17 outlines the role Ajax plays in loading a resource like Google Maps. When a crawler fetches a Web page (1), it waits for the page and all of the embedded



(a) The example DOM, with embedded JavaScript

```
<HTML>
<HEAD>
<META NAME="robots" CONTENT="noarchive">
<TITLE>Dummy Page</TITLE>
<SCRIPT type="text/javascript">
function callWhenLoaded()
{
    document.getElementById("1").innerHTML =
        "This is a new paragraph"
        + "<span>New Span</span>";
}
</SCRIPT>
</HEAD>
<BODY><
<H1>The Page Test1</H1>
<DIV ID=1>
<P>This is a paragraph</P>
</DIV>
</BODY>
</HTML>
```

(b) The DOM of a page before the embedded client-side JavaScript code executes.



(c) The DOM for the default page after the JavaScript has executed.

```
<HTML>
<HEAD>
<META NAME="robots" CONTENT="noarchive">
<TITLE>Dummy Page</TITLE>
<SCRIPT type="text/javascript">
function callWhenLoaded()
{
    document.getElementById("1").innerHTML =
        "This is a new paragraph"
        + "<span>New Span</span>";
}
</SCRIPT>
</HEAD>
<BODY><
<H1>The Page Test1</H1>
<DIV ID=1>
<P>This is a new paragraph<span>New Span</span></P>
</DIV>
</BODY>
</HTML>
```

(d) The DOM of a page after the embedded client-side JavaScript code executes changing the text within a DIV and include a new `span` element.

FIG. 15: JavaScript can interact with the local DOM when executed.

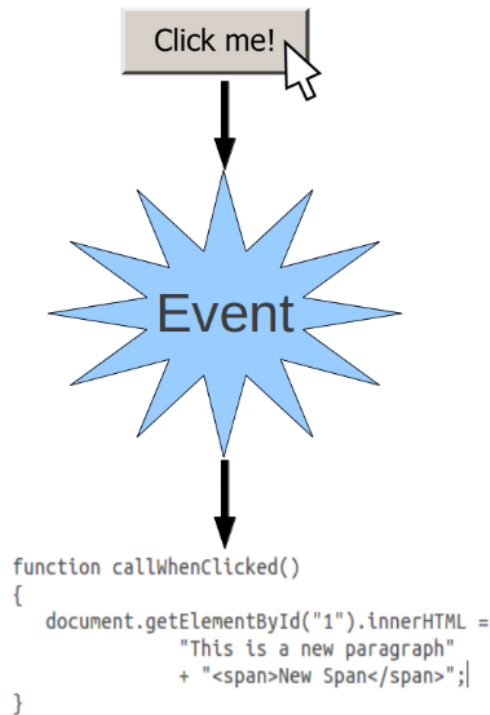


FIG. 16: Events occur on the client, and can cause JavaScript to execute and modify the client-side DOM.

resources to load to consider it fully rendered. At this point, the crawler preserves (2) all of the content on the Web page (A). After the page has loaded, the page can request further resources (3), even without user interaction. The resource is then returned to the Web page to be rendered (4), producing the final intended result (Ab). Because the crawler preserved the page prior to the page being fully loaded, the secondary content is not preserved and the archived version of the Web page is not complete.

This can result in a large number of states that the client-side representation can take, introduced below; note that the DOM is not final until rendering occurs. We take the example of Google Maps [110]. Google Maps can take on several representations (i.e., have multiple descendants) based on user interactions, such as zooming or navigating, or data from the server, such as traffic data. For example, we can modify the route from MITRE's Langley site in Hampton, VA, to the Headquarters in McLean, VA, during a client-side session. This modification is identified by the

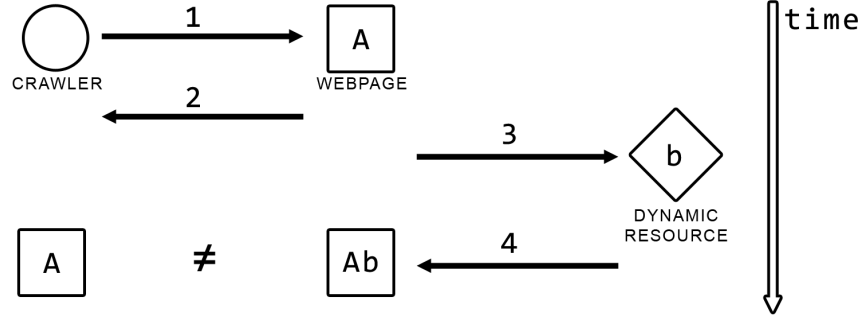
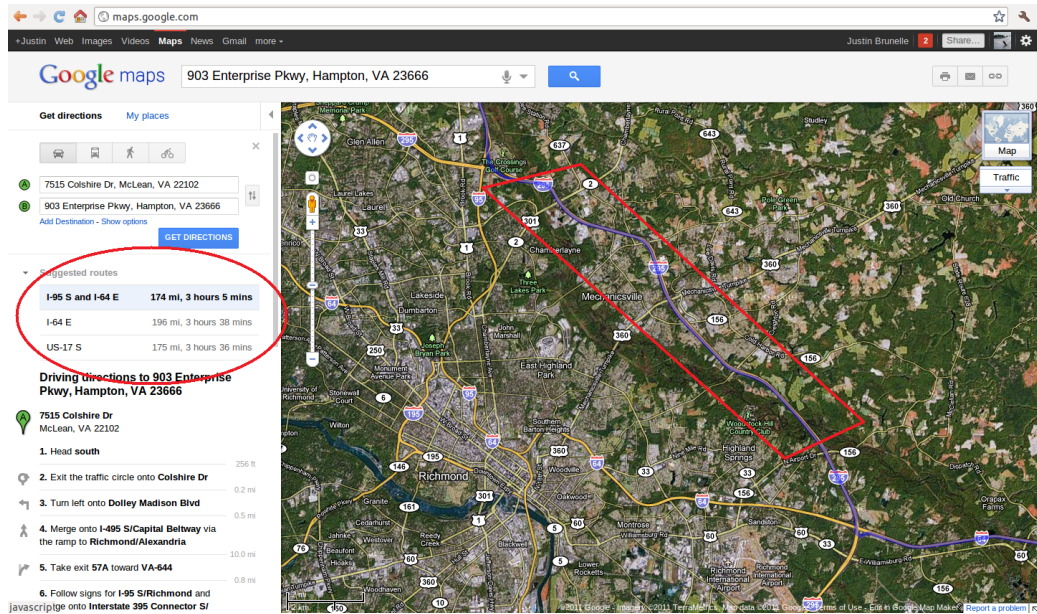


FIG. 17: Ajax interactions modify the DOM after the original page load.

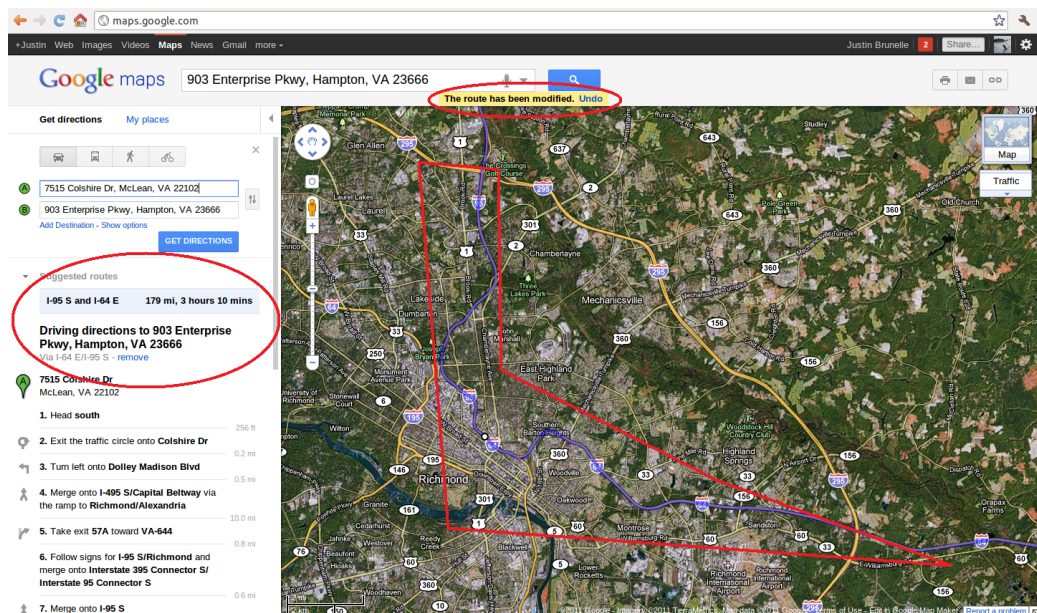
same URI for each descendant, as seen in Figures 18(a) and 18(b). Without equipping archives with the capability to differentiate between the descendants, collisions in the archives may occur [145].

Web users expect the increasing functionality from Web resources that is offered by technologies such as JavaScript and Ajax. Social networking sites, such as Facebook and Twitter, are prime examples of sites that personalize content through Ajax after an initial page load. These sites are difficult to share across users and browsing sessions due to their stateful construction and our inability to adequately represent (or index) the state using current URI parameters.

To accurately discuss the target of this proposal, we define *deferred representations* as representations of resources that use JavaScript and other client-side technologies to load embedded resources or fully construct a representation and, therefore, have low archivability. *Deferred* refers to the final representation that is not fully realized and constructed until *after* the client loads the page and executes the client-side code. The client will render the representation, which is dependent upon environment variables (e.g., the user-agent), as well as user interactions and events that occur within the representation on the client. The final representation is *deferred* until after the user-agent, environment variables, JavaScript, and user events complete their influence on the resource. From this point forward, we will refer to representations dependent upon these factors as *deferred representations*. *Descendants* are the newly constructed representations that change as a result of Ajax and JavaScript. Descendants are more formally defined in Chapter 7.



(a) The original route from Hampton to McLean, as suggested by Google Maps



(b) A modified route from Hampton to Virginia, resulting from user interactions during the same session

FIG. 18: Google Maps is a Web *application* that changes as a result of user interactions. Changes are outlined in red annotations. Note that the URI does not change as a result of the interactions.

2.2.4 PROGRESSION OF WEB LANGUAGES

Programmatically constructed Web pages are not a new concept. However, the difficulty of archiving programmatically constructed Web pages has increased with the movement of the representation construction from the server to the client.

Web servers deliver representations of Web resources to a requesting user-agent. Traditionally, the resource exists as a document on the server and the document is delivered as-is by the server to the client. Server-side programming languages (e.g., the PHP language⁴) can build a representation based on available parameters, server-side data (e.g., a database), or the current state of the server. In the case of PHP, the page is constructed on the server in its entirety before transmission to the client and, only after construction, is delivered to the client. We refer to these representations as *server-side dynamic* because they can be constructed on-demand at the server before sending the representation to the user-agent. Representations can also be delivered by the server to the client and then change or be constructed on the client using scripting languages like JavaScript. Once the client renders the representation, the scripting language will execute, potentially causing the representation to morph on the client with or without subsequent requests to a server for additional representations. We refer to these representations as *client-side dynamic* because they can be constructed at render time on the client. Both of these classifications of dynamic representations allow for personalized content (e.g., “Welcome, Justin” or layout preferences).

The two design paradigms can be embodied and compared by the LAMP⁵ (Linux Apache MySQL PHP) stack vs the MEAN⁶ (MongoDB Express AngularJS Node.js) stack. The LAMP stack places the responsibility of constructing a representation on the server using a combination of the Apache server using PHP to build a representation with the help of a MySQL database, while MEAN shifts the responsibility of constructing a representation to JavaScript and the client using a combination of Node.js on the server and AngularJS on the client. This shift in design paradigm is made possible by the adoption of client-side JavaScript and also embodies the shift in reliance from server-side to client-side technologies to build representations. The disconnect between the types of dynamism and personalization in Web resources makes discussing the nature of change and content generation difficult. However,

⁴<http://php.net/>

⁵[http://en.wikipedia.org/wiki/LAMP_\(software_bundle\)](http://en.wikipedia.org/wiki/LAMP_(software_bundle))

⁶<http://mean.io/#!/>

the client-side dynamism enabled by JavaScript is a primary archival challenge as JavaScript's adoption increases in prevalence.

As noted by David Rosenthal in his blog [245], HTML5 allows Web pages to be deployed with a programming language instead of a markup language. For example, multimedia can be delivered from containers within a page instead of having to be loaded from JavaScript. While JavaScript can load entities external to the client, HTML5 selects what representation to provide based on the entities on the client or information in local storage (this storage is inaccessible to today's archival tools). JavaScript and Ajax had previously been the only technologies available that enabled run-time loading of content or inclusion of multimedia resources; HTML5 provides these functionalities natively. As Web browsers begin to adopt HTML5, the prevalence of the unarchivable representations will increase [246], but the features of HTML5 make traditional archival methods insufficient for accurate archiving.

HTML5 is the next approved iteration of the HTML standard. HTML5 builds in dynamic behavior natively that has been traditionally provided by Flash, JavaScript, and other scripting and multimedia technologies. That is, the DOM can be designed to change upon user interactions rather than be acted upon by an external script or allowing for external content to be loaded into the client-side DOM without the use of embedded or external scripts [25]. Specifically, HTML5 can access client-side databases and even allow user interactions with 3D visualizations. HTML5 has the ability to record and read client-side state [300]. However, this can only be performed when the code understands how to do so; the client must render the state, not dereference it.

Extending the example in Figure 15, we take the same DOM in Figure 19 and show the code and associated impact on the DOM. In this example, HTML5 reads markup from local storage and modifies the DOM to reflect the storage contents.

HTML5 is also implementing features that allow client-side state to be shared and captured through the history application programming interface (API) [33]. The history API uses JavaScript to establish client-side state and save the state in local storage. The API allows the client to specify a desired state in the HTTP request header.

Just as the mobile computing field began with personal digital assistants (PDAs) and other limited-capability devices, HTML began as a very basic method of representing Web resources. As mobile computing advanced, features that were once

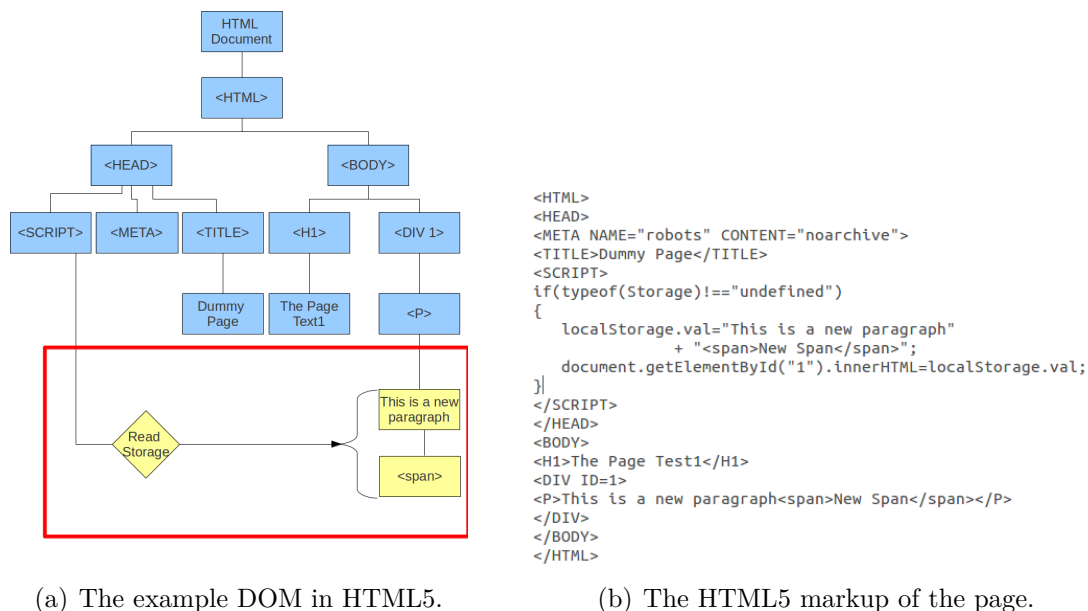


FIG. 19: Extending the example in Figure 15, HTML5 can read from load storage and modify the DOM.

provided as external and separate services (GPS, Web browsing, telecommunications and WiFi, etc.) became native to the devices as users demanded more of them [17, 269]. Similarly, as users expect more personalization of their Web content, HTML is progressing to meet those demands with native multimedia, client-side control of the DOM, and multi-state client-side representations. Mobile devices are advancing beyond the traditional PDAs, smart phones, and tablets to wearable and embedded devices. Similar to the advancement from limited functionality devices to highly functional, highly personalized devices, we can expect HTML to continue its advancement beyond HTML5, continuing the trend of personalized Web resources. As such, archival methods and frameworks must adapt with the technology to remain able to effectively archive resources.

2.3 TRENDING TOWARD THE DYNAMIC WEB

As the Web becomes more dynamic, archivists are realizing the importance of evolving to overcome this added challenge to Web archiving. In a workshop focused on current and future archiving to keep up with change with the Web, the International Internet Preservation Consortium (IIPC) discussed the evolution of the Web

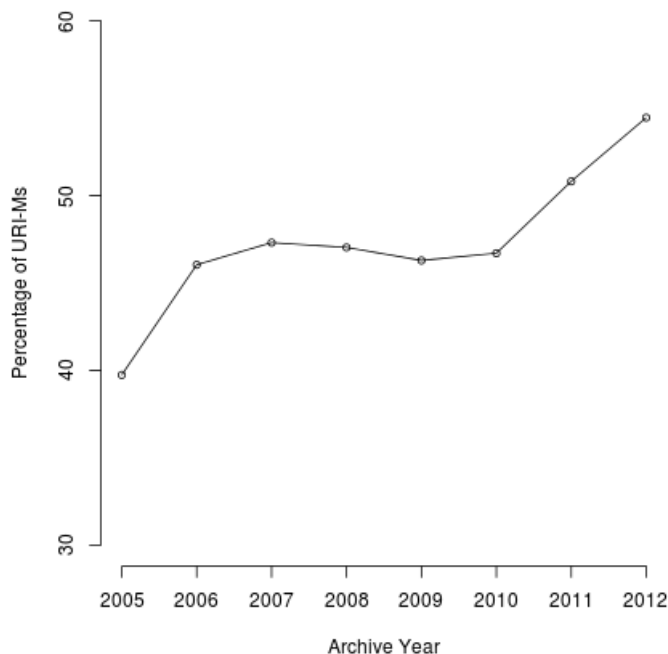


FIG. 20: Resources are using more JavaScript to load embedded resources over time (Figure from [52]).

[9]. As the Web evolves, archivists need to develop new tools and technologies to modernize the archival process and keep up with the changing Web [228]. Archivists and digital librarians have discussed the impact of the evolving Web on archiving, as well as the trends influencing the future difficulties archivists will need to overcome.

Representations that are dependent upon JavaScript and user interaction are increasing in prevalence and archival impact. In a previous work [52], we observed the embedded resources loaded from a set of mementos from 2005-2012 to determine how many resources are being loaded from JavaScript. As shown in Figure 20, more resources are using JavaScript to load embedded resources over time. In 2005, 39.7% of mementos in our collection used JavaScript to load at least one embedded resource; this trend increases to 54.5% of the collection using JavaScript to load at least one embedded memento in 2012. That is an increase of 14.7% of resources using JavaScript between 2005 and 2012.

A more fundamental change in Web language is also occurring. Rosenthal mentioned that HTML5 (discussed in detail in Section 2.2.4) is changing the native language of the Web from HTML to JavaScript [245]. This is a movement from single-state content to a program delivered in the same container. The Web is migrating toward more complex, more interactive resources reliant on their environmental context and representations with a movement away from documents constructed with HTML in favor of programmatic JavaScript-enabled applications. Similar progressions have occurred in the history of Computer Science, such as the progression from vacuum tubes to microprocessors and the reliance on mainframes or supercomputers to the prevalence of personal computers. As Rosenthal notes, the migration from documents described by HTML to applications developed with programming languages is making preservation of the Web more difficult because crawlers are becoming less effective at collecting representations of resources; a representation may no longer be discoverable by crawlers through the sole action of following links but only by initiating events and changing environmental perspective.

2.4 HASHBANG URIS

HashBang URIs are a method intended to allow client-side representations to be identifiable and sharable [281]. HashBang URIs identify client-side state in the URI. That is, they specify the information needed to dereference (and render) the target representation. However, Web pages began using Hashbang URIs in 2011 and their use has tapered over the years; they are still important to understand as part of previously popular methods of identifying client-side representation state. Traditionally, hash URIs, or fragments, are used to identify a part of a retrieved document or identify an offset to be applied by the user-agent, the nature of which is determined by the document's MIME type. To represent the state of a Web application, HashBang (#!) URIs identify a representation of a resource generated by JavaScript. An example HashBang URI used by Jeni Tennison [281] is

```
http://lifehacker.com/#!5770791/top-10-tips-and-tricks-for-making-
your-work-life-better
```

and uses the HashBang to give JavaScript a parameter

```
5770791/top-10-tips-and-tricks-for-making-
your-work-life-better
```

indicating that it should redirect the browser to the proper resource. The client will read the hash portion of the URI and either redirect the client to the proper resource or use the provided information to generate the representation by changing the client-side state. This is in direct contrast to a URI specifying a server-side state through traditional parameters:

```
http://lifehacker.com/?_
escaped_fragment_=5770791/top-10-tips-and-tricks
-for-making-your-work-life-better
```

The server reads the parameters specified after the ? and generates (or chooses) the representation during the dereferencing process. The client does not process these parameters when constructing the representation since the client can rely on the server to provide the state.

Without the HashBang URI the browser would provide the following data in a POST to `http://lifehacker.com/index.php?_actn_=ajax_post` [301]:

```
op=ajax_post
refId=5770791
formToken=d26bd943151005152e6e0991764e6c09
```

HashBang URIs are Google's convention for standardizing the mapping of client-side state in the URI. URIs follow the syntax defined in RFC 3986 [38]. Google's HashBang allows state to be communicated when dereferencing a URI with a representation dependent upon client-side code. However, this relies on the client-side code to understand how to interpret the URI when a request is received. HashBang URIs can identify subsets of Web documents, identify the state and the representation of a resource, and also capture the state generated from user interaction and streamed content. If the resource follows Google's proposal [108], the Website also commits to making that content available through an equivalent base URI with an `_escaped_fragment_` parameter (e.g.,

```
?_escaped_fragment_=5770791/
top-10-tips-and-tricks-for-making-your-work-life-better
```

as shown previously).

HashBang URIs allow servers to cache content, crawlers to index content, and users to utilize the back button, bookmark features, and represent the client-side state in the URI. Additionally, since each state can be modeled with a different URI, the back button and bookmarking becomes relevant in the discussion of navigating content in a Web composed of stateful, deferred representations. State can conceivably be shared between users using a HashBang URI. However, this assumes the data that has generated the document remains constant, as well as the client’s understanding of how to interpret the HashBang part of the URI. These assumptions restrict the usage of the documents to only the systems in which all participants understand the implementations.

Despite the HashBang URI advancement in state representation, there are still issues in the reading and archiving of content represented by HashBang URIs. The content is still JavaScript-dependent, and the resulting representations are often deferred. The browser relies on JavaScript and similar technologies to generate the representations in accordance with the HashBang URI. Additionally, HashBang URIs are not widely implemented by popular sites. Screen scraping and other methods of interpreting content are unsuitable for gathering the expected static content normally provided by Web pages.

2.5 TRENDS IN WEB ARCHIVING

Whether considering the Web viewed from mobile devices or the linked data Web, it is evident that there is more than “one Web” that needs to be archived. As we have mentioned in this dissertation, a key challenge in today’s Web archiving is replaying representations that users experience rather than the representation given to crawlers when URI-Rs are dereferenced. However, the nature of Web archiving – a crawler or other agent capturing representations and storing them out of their native, live environment (what Lanier refers to as a first-order expression (Section 1.1)) – can remove the linkages that exist on the live Web. Not only do multiple Web archives exist with varying focuses or specializations, but there are also different types of “Webs” that exist within the live Web (e.g., the mobile and linked data Webs) that have linkages that are lost during archiving. These linkages establish the context of a resource, and are important to second-order expressions.

The Memento Framework (Section 2.5.1) more closely integrates the archived and live Webs to help users discover, navigate to, and browse mementos based on their

live Web counterparts. Memento provides a tighter coupling of the live and archived Webs, as well as links the known Memento-compliant archives together.

Similarly, linkages between *desktop* versions of pages (i.e., the representations users see when visiting a page on a traditional browser) and their *mobile* versions (i.e., the representations users see when visiting a page on a smaller-form device, such as a smart phone) exist on the live Web but might be either not discoverable by crawlers or lost during archiving. Archivists are placing an increased emphasis on archiving the mobile Web and maintaining the linkages between the desktop and mobile Webs in the archives (Section 2.5.2). Linkages and relationships between information resources may change over time, and Memento helps establish temporal linkages between the resources [294, 290].

Memento and the ongoing efforts to archive the mobile and linked data Webs are indicative of a trend in Web archiving to provide better tools to Web users as well as to increase the ease-of-discovery and coverage of mementos in the archives [8].

2.5.1 MEMENTO

The Memento Framework [292, 206, 294, 293] enables Web browsing and navigation in the temporal dimension by aggregating the mementos of a resource from known archives. Live Web resources exist in the *perpetual now*, meaning changes to resources overwrite older versions. Memento bridges the gap between the live and archived Webs through content-negotiation and links. The `accept-datetime` HTTP header enables temporal content-negotiation and allows Memento-compliant servers to return a memento with the desired `memento-datetime` (the archival time). In other words, Memento allows the client to request a prior version of a resource.

In Chapter 1, we defined the terms *original resource*, *memento*, and *TimeMap* (identified by URI-R, URI-M, and URI-T, respectively). *Original resources* are the live Web versions of a resource, and a memento is the archived version of a live Web resource. A TimeMap is a machine-readable list of URI-Ms of a URI-R and are available for specific archives (e.g., all mementos of a resource from only the Internet Archive) and all archives (e.g., all mementos of a resource from all known archives).

Figure 21 illustrates the Memento Framework. Memento-compliant URI-Rs will return an HTTP Link Header to its TimeGate (identified by URI-G) when dereferenced. The TimeGate negotiates the correct memento to return if provided an `accept-datetime` header.

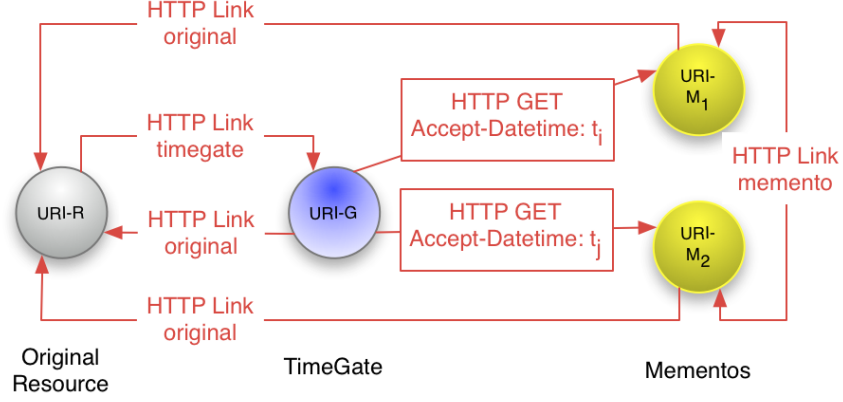


FIG. 21: The Memento Framework enables temporal browsing and aggregates the holdings of multiple archives. This image is available as part of the Memento Framework Introduction [289].

When URI-Ms are dereferenced, they return HTTP headers with links to their previous and next mementos (if they exist) as well as their URI-R. These linkages enable navigation and browsing in the temporal dimension. The TimeGate also returns a URI-T for the URI-R, providing a temporally sorted list (from oldest to newest) of URI-Ms of the URI-R. A TimeMap is useful for quickly accessing all known mementos of a URI-R.

Several tools leverage the Memento Framework to allow more readily accessible temporal navigation to users via their browsers (i.e., user-agents). MementoFox is a Mozilla Firefox add-on that uses TimeGates to allow temporal browsing from the browser [255]. A similar add-on exists for Google Chrome [260] and mobile iOS and Android devices [288]. Mink, a Google Chrome add-on, helps better integrate the live and archived Webs by using Memento to find mementos of original resources being accessed by the Chrome Browser [147]. While other memento browsers leverage TimeGates, Mink uses TimeMaps to augment live Web browsing by alerting the user of existing mementos. MobileMink provides the same functionality for Android devices but also identifies mobile and desktop URI-Rs for the resource and merges the TimeMaps of both, presenting the integrated TimeMap to the user [137]. MobileMink also allows users to push resources into the archives to improve the coverage of mobile sites. AlSum's mCURL tool provides native Memento features using the curl command [10]. Each of these tools are examples of software making use of the Memento Framework to more closely link the past and present Webs.

2.5.2 ARCHIVING MORE THAN JUST THE DESKTOP WEB

The Web that humans most commonly experience is not the only Web that exists and should be archived. As an example, we consider the mobile and linked data Webs in addition to the traditional *desktop* Web. Each of these types of Webs (linked data, mobile, and desktop) has a different coverage in the archives despite their importance to the Web user’s browsing experience.

The Personalized Web

The IIPC has held entire workshops dedicated to the discussion of problems and potential solutions to the movement towards unarchivable content [247, 211]. The “classical model of Web archiving” fails when targeting resources whose representations are dependent upon JavaScript (deferred representations) or user interaction (descendants). Archives’ policy of indexing on only memento-datetime and URI-R is also making archiving difficult because resource state on the client is not reliably identified due to the influence of environment variables and user interaction on the end representation [211, 145]. To resolve these issues, classical crawlers must improve to allow the capture of these dynamic technologies and resources. One potential solution is to capture user experiences as images under the notion that something highly representative at low-fidelity is better than missing content. However, this leaves much to be desired when information mining and session replay are of great value to the archives but come with a large amount of storage overhead and burden on the archives [11].

These trends culminate in a culture change: user-experience is no longer limited to stateless client and server interactions, but now also involves client-side state. Rosenthal mentions [245] that the user-experience is not preservable (long term) because it exists externally of the client – the representation requires client-side execution to change the representation state. For example, the Wayback Machine (the Internet Archive’s replay service, defined in Section 3.1) would need to be re-engineered to recreate a user’s browsing session on a particular resource. Bjarne Andersen from the Netarchive.dk explained further that “dynamic” sites (i.e., those that update on the client) are often missed by archival crawlers and create problems for recreating user experiences [163].

What we archive is increasingly different from what we as interactive users experience. We will explore Figure 22 – multiple snapshots of President Barack Obama’s Facebook page (live as of May 11th, 2011 and as a memento) – as an example. As a logged-in and authenticated user, I viewed Obama’s Facebook page (Figure 22(a)), but received a different representation when I have logged out of my account (Figure 22(b)). This is one example of a stateful client-side representation; my representation changed based on the parameters available (in this case, my authentication status) on the client⁷. Crawlers do not authenticate to Facebook before attempting to archive the resource, and we would suspect that the representation in Figure 22(b) would be in the archives (note that Archive-It has enlisted the help of Umbra to crawl a small subset of pre-defined domains such as Facebook that are not satisfactorily archived using Heritrix [235]). However, the Internet Archive could not capture a memento of the page due to the robots.txt protocol (Figure 22(c)) [241, 278]. The WebCite memento (Figure 22(d)) is also inadequately archived because the Facebook servers were unable (or unwilling) to return a representation suitable for the WebCite user-agent.

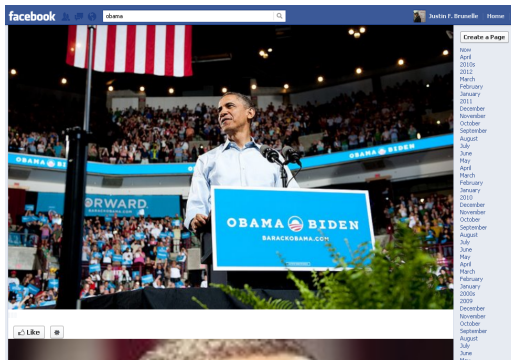
This example highlights the differences in user experiences and their lack of adequate capture by the archives. Our primary focus in this work is to provide the framework that enables user experiences to exist in the archives.

The Mobile Web

Crawlers traditionally and overwhelmingly use a *desktop* user-agent string when crawling and archiving the Web. As a result, crawlers frequently miss mobile versions of pages, and the mobile Web continues to be less prominent in the archives. This phenomenon persists even as mobile devices grow in power, use, and ubiquity and the mobile Web continues to grow and become more prevalent [302]. Because of their prevalence on the Web, it is increasingly important to archive mobile resources and representations. However, because mobile resources are not always directly linked from their desktop counterparts, it is difficult for crawlers to find pages in the mobile Web [136].

Even if the mobile resources are discoverable, crawlers may not be able to reliably archive the mobile representations; some Web resources offer different representations

⁷This behavior also exists on pre-JavaScript resources – authentication is not a new challenge for archivists. However, this behavior does illustrate the increasing personalization and difference between representations (first- and second-order expressions) Web users see and what crawlers are able to archive.



(a) The representation of Obama's facebook page when I am recognized as an authenticated user



(b) The representation of Obama's Facebook page when I have logged out of my account.



<http://www.facebook.com/barackobama>

Latest

Show All

Page cannot be crawled or displayed due to robots.txt.

See www.facebook.com/robots.txt page. [Learn more](#) about robots.txt.

(c) Facebook's robots.txt file prevented the Internet Archive from procuring a memento of Obama's Facebook page.



(d) The WebCite memento of Obama's Facebook page is missing because of a server's inability to return a representation to the provided client.

FIG. 22: President Obama's Facebook page (as observed 2011-05-11) changes based on client-side parameters and is not adequately archived.

to different users based on the **user-agent** string in the HTTP headers or other environmental factors. Web crawlers capturing content for the archives may receive different representations based on the crawl environment.

Servers often provide lighter-weight representations to mobile user-agents and the larger, full-feature versions to desktop user-agents. This allows mobile devices to more easily and more quickly browse the Web. With the increasing prevalence of mobile browsers (50% - 68% of sites have mobile versions [258, 185, 137]), it is important to capture these mobile representations of resources. Content viewed from a mobile browser is often different than content viewed from a traditional desktop browser [136, 286, 145]. Some clients use Ajax and other widget technology to create context-aware mobile services, which may modify elements of the representation to accommodate device limitations like screen size [164]. This activity further segments what content is provided to clients [302].

Mobile pages often contain links to additional resources instead of embedded text and often reduce the number of images embedded in the page [136]. For example, the mobile version of <http://espn.go.com/> contains a section on ESPN Videos, while the desktop version does not. To quantify the differences, the desktop version contains 201 URI-Rs, while the mobile version contains only 58 URI-Rs (the two sets of links are mutually exclusive). A user may view news articles or other content on a mobile device and be unable to recall the article in the archives. To capture and record the complete set of content at <http://espn.go.com>, each of these different representations, both mobile and desktop representations need to not only be stored in the archives but also linked to each other. Each of these environments will provide a different experience, and the aggregate of desktop and mobile viewing can improve the available resources for reconstruction.

We discuss our preliminary work with the mobile Web in Section 4.4 and provide examples in Figure 49.

The Linked Data Web

Linked data refers to machine-readable information resources (e.g., formatted as XML, JSON, or RDF) that connect other data and resources or enrich and annotate resources on the Web [160, 291]. These information resources traditionally operate in the background, opaquely to human users and only used by machines.

Linked data provides machine-readable metadata that is preserved along with the

```

1  $ curl -I https://en.wikipedia.org/wiki/Stop_Online_Piracy_Act
2  HTTP/1.1 200 OK
3  Server: nginx/1.9.2
4  Date: Tue, 14 Jul 2015 00:01:34 GMT
5  Content-Type: text/html; charset=UTF-8
6  Connection: keep-alive
7  X-Content-Type-Options: nosniff
8  X-Powered-By: HHVM/3.6.1
9  Content-language: en
10 X-UA-Compatible: IE=Edge
11 Vary: Accept-Encoding, Cookie
12 Last-Modified: Sun, 12 Jul 2015 14:45:11 GMT
13 X-Varnish: 753875825 749624149, 1725229408 1553107569
14 Via: 1.1 varnish, 1.1 varnish
15 Age: 73395
16 X-Cache: cp1052 hit (1), cp1053 frontend hit (28)
17 Strict-Transport-Security: max-age=31536000
18 Cache-Control: private, s-maxage=0, max-age=0, must-revalidate
19 Set-Cookie: GeoIP=US:VA:Suffolk:36.8742:-76.5613:v4; Path=/;
20             Domain=.wikipedia.org
21 X-Analytics: page_id=33629639;ns=0;https=1
22 Set-Cookie: WMF-Last-Access=14-Jul-2015;Path=/;HttpOnly;
23             Expires=Sat, 15 Aug 2015 00:00:00 GMT

```

FIG. 23: HTTP Response for the Wikipedia SOPA page.

resources to which the data links. Resources with such metadata can facilitate automatic discovery of other resources or link them within the archives. However, these information resources are not as prevalently represented in the archives, despite the important linkages they create in the live Web between resources. Without archiving the linked data Web, we lose linkages that exist between information resources on the live Web that are essential to the context and understanding of the resource.

We will take the SOPA Wikipedia page (URI-R https://en.wikipedia.org/wiki/Stop_Online_Piracy_Act) as an example (first discussed in Section 1.2). We can dereference the URI-R to get the desktop version of the SOPA Wikipedia Page (Figure 23).

DBPedia offers structured data about Wikipedia articles. If we dereference the DBPedia SOPA URI-R http://dbpedia.org/resource/Stop_Online_Piracy_

Act, we receive an HTTP 303 redirecting us to a DBPedia page (URI-R http://dbpedia.org/page/Stop_Online_Piracy_Act, an HTML page meant for humans), indicating that our original resource has no representation to return and instead we should visit the other page (Figure 24).

When we follow the redirect, we see the link header indicating an alternate Web document (URI-R http://dbpedia.org/data/Stop_Online_Piracy_Act.rdf, an RDF page meant for robots) about the DBPedia SOPA page. This page returns additional link headers with further resources (and so on until we run out of related documents)⁸ (Figures 25 and 26).

The linked nature of these resources makes discovery easier. However, the archival coverage of each layer is not equal. As an example metric, we consider the cardinality of each of these resources’ TimeMaps (i.e., the number of mementos in each TimeMap). We see an example of the discrepancy in the Internet Archive TimeMaps for Wikipedia versus DBPedia (Figure 27). We requested the TimeMaps of each of the aforementioned URI-Rs and counted the number of mementos in each TimeMap. The results are summarized in Table 3.

URI-R	TimeMap
https://en.wikipedia.org/wiki/Stop_Online_Piracy_Act	539
http://dbpedia.org/resource/Stop_Online_Piracy_Act	1
http://dbpedia.org/page/Stop_Online_Piracy_Act	1
http://dbpedia.org/data/Stop_Online_Piracy_Act.rdf	0
http://dbpedia.org/data/Stop_Online_Piracy_Act.xml	0

TABLE 3: The cardinality of the TimeMaps varies between the Desktop and Linked Data Webs (TimeMaps retrieved 2015-07-14).

While the Web that humans most frequently experience is comparatively well-archived, the mobile Web (which is increasing in popularity) and the linked data Web (which supports the semantic Web) are disproportionately not archived in regular Web archives like the Internet Archive.

⁸Note that the Link header refers to the URI-G http://mementoarchive.lanl.gov/dbpedia/timagate/http://dbpedia.org/page/Stop_Online_Piracy_Act which refers to a TimeGate for an archive that Los Alamos National Laboratory is operating on behalf of DBPedia to help address the problem with the header of the TimeGate’s HTTP response of the URI-G http://mementoarchive.lanl.gov/dbpedia/timagate/http://dbpedia.org/data/Stop_Online_Piracy_Act.xml.

```

1  $ curl -IL http://dbpedia.org/resource/Stop_Online_Piracy_Act
2  HTTP/1.1 303 See Other
3  Date: Tue, 14 Jul 2015 00:02:32 GMT
4  Content-Type: text/html; charset=UTF-8
5  Content-Length: 0
6  Connection: keep-alive
7  Server: Virtuoso/07.20.3214 (Linux) i686-generic-linux-
8      glibc212-64 VDB
9  Location: http://dbpedia.org/page/Stop_Online_Piracy_Act
10 Expires: Tue, 21 Jul 2015 00:02:32 GMT
11 Cache-Control: max-age=604800
12
13 HTTP/1.1 200 OK
14 Date: Tue, 14 Jul 2015 00:02:33 GMT
15 Content-Type: text/html; charset=UTF-8
16 Content-Length: 70502
17 Connection: keep-alive
18 Vary: Accept-Encoding
19 Server: Virtuoso/07.20.3214 (Linux) i686-generic-linux-
20      glibc212-64 VDB
21 Expires: Tue, 21 Jul 2015 00:02:33 GMT
22 Link: <http://dbpedia.org/data/Stop_Online_Piracy_Act.rdf>;
23      rel="alternate"; type="application/rdf+xml";
24      title="Structured Descriptor Document (RDF/XML format)",
25      <http://dbpedia.org/data/Stop_Online_Piracy_Act.n3>;
26      rel="alternate"; type="text/n3"; title="Structured
27      Descriptor Document (N3/Turtle format)",
28      <http://dbpedia.org/data/Stop_Online_Piracy_Act.json>;
29      ...
30      <http://mementoarchive.lanl.gov/dbpedia/timegate/
31      http://dbpedia.org/
32      page/Stop_Online_Piracy_Act>; rel="timegate"
33 Cache-Control: max-age=604800
34 Accept-Ranges: bytes

```

FIG. 24: An HTTP 303 redirect for the DBPedia resource.

```

1  $ curl -I http://dbpedia.org/data/Stop_Online_Piracy_Act.rdf
2  HTTP/1.1 200 OK
3  Date: Tue, 14 Jul 2015 00:00:59 GMT
4  Content-Type: application/rdf+xml; charset=UTF-8
5  Content-Length: 46701
6  Connection: keep-alive
7  Vary: Accept-Encoding
8  Server: Virtuoso/07.20.3214 (Linux) i686-generic-linux-
9      glibc212-64 VDB
10 Expires: Tue, 21 Jul 2015 00:00:59 GMT
11 Link: <http://dbpedia.org/data/Stop_Online_Piracy_Act.xml>;
12      rel="alternate"; type="application/rdf+xml";
13      title="Structured Descriptor Document (RDF/XML format)",
14      ...
15      <http://mementoarchive.lanl.gov/dbpedia/timegate/
16      http://dbpedia.org/
17      data/Stop_Online_Piracy_Act.rdf>; rel="timegate"
18 X-SPARQL-default-graph: http://dbpedia.org
19 Cache-Control: max-age=604800
20 Accept-Ranges: bytes

```

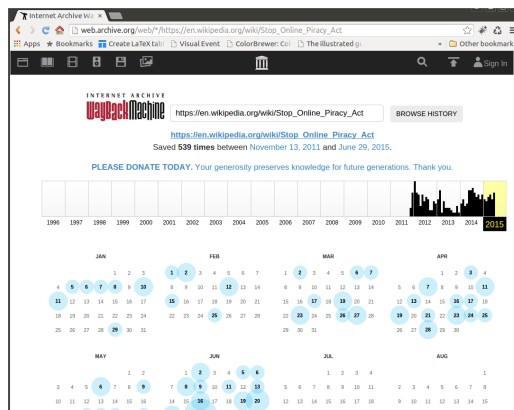
FIG. 25: Further information resources are provided in the Link header.

```

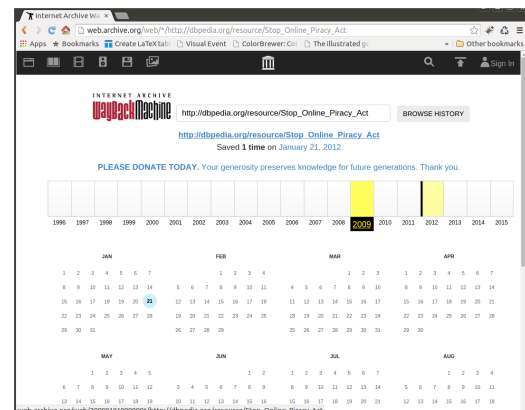
1 $ curl -I http://dbpedia.org/data/Stop_Online_Piracy_Act.xml
2 HTTP/1.1 200 OK
3 Date: Tue, 14 Jul 2015 00:03:17 GMT
4 Content-Type: application/rdf+xml; charset=UTF-8
5 Content-Length: 46701
6 Connection: keep-alive
7 Vary: Accept-Encoding
8 Server: Virtuoso/07.20.3214 (Linux) i686-generic-linux-
9      glibc212-64 VDB
10 Expires: Tue, 21 Jul 2015 00:03:17 GMT
11 Link: <http://dbpedia.org/data/Stop_Online_Piracy_Act.n3>;
12      rel="alternate"; type="text/n3"; title="Structured
13      Descriptor Document (N3/Turtle format)",
14      ...
15      <http://mementoarchive.lanl.gov/dbpedia/timegate/
16      http://dbpedia.org/
17      data/Stop_Online_Piracy_Act.xml>; rel="timegate"
18 X-SPARQL-default-graph: http://dbpedia.org
19 Cache-Control: max-age=604800
20 Accept-Ranges: bytes

```

FIG. 26: Following the Link header leads to new information resources in new Link headers.



(a) The Internet Archive TimeMap of the Wikipedia SOPA page.



(b) The Internet Archive TimeMap of the DBpedia SOPA page.

FIG. 27: The Internet Archive TimeMap cardinality is much smaller for the Linked Data Web.



FIG. 28: Smaller, focused snapshots, in aggregate, form a complete picture [75]. This Hockney joiner is called “Yosemite.”

2.6 DAVID HOCKNEY’S JOINERS

David Hockney is one of the 20th century’s most influential artists [311]. Hockney created a series of works referred to as “joiners” which are collages of photographs or images (Figures 28 and 29). The smaller pictures combined to create a larger, higher-order image. These “joiners” created a complete view of a scene from smaller, more specified views. While the individual pictures exist in a context of their own, they are not limited by their original format (e.g., Polaroid snapshot) and can be aggregated to construct a more meaningful, more substantial view.

Hockney’s joiner art is a metaphor for mementos. Archives can capture small, specified views of Web resources. These captures identify the representation of a resource at a specified time and in a specified environment and, more specifically, in the crawler’s environment. Current technologies cannot capture multiple snapshots



FIG. 29: David Hockney’s “joiners” are single images that, together, create a larger image [199]. This joiner, constructed in Hockney’s style, is by Jordan Mills.

of the same resource from a user’s perspective; they are limited to the crawler’s perspective. We need to provide crawlers the ability to discover deferred representations or context dependent representations. Combined, the snapshots of the resource taken by the archives and the snapshots from contributing users provide a full picture of the resource. Since resources with deferred representations change based on context and client-side interactions or events, the only way to gain a full view of a resource is from simulating or initiating the client-side code execution.

Archive-captured snapshots can also be considered equivalent to video game “walk-throughs” that are seen on the Web [208] (more specifically, on YouTube [314]). Each walk-through illustrates a single user’s experience with the game for a specific level, task, or accomplishment. Given enough walk-throughs, the entire gameplay can be experienced by aggregating – or *joining* – multiple user experiences captured from distributed contributions. It is important to note that this is fundamentally different than emulation which executes legacy code and is instead the recreation of user interactions and experiences with the code. In Hockney’s art, there must

be sufficient snapshots to gain a view of the entire resource⁹. To gather a complete view of a resource, multiple user experiences must be captured and entered into the archives. These user experiences, each composite mementos on their own, represent a single environmental capture – one image in a Hockney joiner. In aggregate, an entire resource experience can be realized as aggregates of snapshots in time. This can be equated to the archival equivalent of pre-caching due to the goal of discovering resources that will be needed when users interact with a memento before the user interacts with the representation.

Joiners are a metaphor for deferred representations with descendants: archived versions of a resource that are composed of different views of the same memento listed in aggregation. More specifically, we aim to construct new views (not previously captured) of the original resource from a variety of mementos by recognizing deferred representations and pre-caching all potentially embedded resources.

2.7 SUMMARY

In this chapter, we have presented a high level overview of the Web, Web archiving, and a metaphor for information construction. A common understanding of how information is represented on the Web, as well as the make-up of the Web (e.g., HTML versus JavaScript), is an essential prerequisite of our discussion on how the change from Web pages to Web applications has made archiving more difficult. We have also presented an overview of the trends in modern Web archiving to demonstrate the current successes and short-comings of the Web archiving community's efforts to archive at Web scale.

Using this common set of knowledge, we will discuss the prior research that we leveraged during our research into creating a proposed framework for archiving deferred representations.

⁹Using software to create aggregate images from smaller images is not a new concept. Adobe [4], Hugin [129], and PTgui [230] comprise a small subset of the software solutions available to create panoramic images from a set of smaller snapshots.

CHAPTER 3

RELATED WORK

In this chapter, we discuss the previous research that has established the ground-work for archiving deferred representations. We begin with an exploration and study of existing archival systems and tools to understand current archival efforts, elaborating on their purpose, strengths, and potential weaknesses (Section 3.1). We then discuss the challenges associated with archiving the Web, including change-rate studies, “deep” Web discovery and archival efforts, and the different *types* of Webs that archives target and the unique archival challenges with each (Section 3.2). We also discuss the aspects of client-side representations that may make them difficult to archive, including the role of JavaScript and Ajax in constructing representations (Section 3.3). With an understanding of current archival efforts and challenges, we discuss various approaches to assessing memento quality (Section 3.4).

Other research that has not necessarily been performed with Web archiving in mind has contributed to our work archiving deferred representations. We discuss how Web content is traditionally cached and how Ajax and JavaScript caching is performed for browser and Web application performance boosts (Section 3.5). Further work has been performed in the client-side testing and Web security domains to monitor client-side Web applications, including detecting malware, error tracking, and understanding users’ client-side interactions (Section 3.6). Other works have focused on replaying browser navigation paths and client-side sessions (Section 3.7). Web user sessions are traditionally unique and not shareable, but prior works have identified tools and methods of sharing a user’s browsing stream.

3.1 ARCHIVAL EFFORTS AND TOOLS

Traditional Web archiving efforts have used automated crawlers to find and capture content on the Web [205, 287]. For example, the Heritrix Web Crawler [201, 264] is the Internet Archive’s primary crawler tool [205]. Heritrix is an open-source tool that crawls a set of provided URI-Rs (called a seed list) and can be configured to crawl into the broader Web. The crawler captures the representation and all of the

requisite resources and stores them in Web ARChive (WARC) files which are then ingested and viewed through the Wayback Machine [287]. Mementos load embedded resources from the Wayback Machine. Using the Wayback Machine in Proxy mode can help mitigate leakage (a phenomenon that occurs when live Web representations are loaded into mementos via JavaScript and Ajax) by directing all HTTP requests to be relative to the archive [84, 236].

Prior to Heritrix3, versions of Heritrix had limited support for JavaScript. The crawler’s parsing engine attempted to detect URIs in scripts, fetch the scripts and recursively repeat this process with the intention of ensuring maximum coverage of the archive creation process. We explore using PhantomJS and Heritrix together in a two-tiered archival framework in Chapter 7 and show that incorporating PhantomJS [226], a headless WebKit (the layout engine used by Google Chrome, Apple Safari, etc.), into the Heritrix crawl workflow significantly reduces crawl speed [57]. PhantomJS has a JavaScript API that runs from the command line. It implements headless browsing, which loads the entire page using WebKit. PhantomJS also offers screen capture, network monitoring, and page automation. This advancement greatly increases the potential for accurate JavaScript processing and the likelihood that all resources required to replay a Web page are captured by a crawler [47], but it was not suitable for Web-scale crawls due to performance issues.

Due to its expansive size, the entire Web cannot be archived in its entirety by these automated crawlers [26]. With deferred representations increasing in prevalence, tools such as Heritrix are becoming ineffective [52]. As deferred representations increase in prevalence and popularity, the efforts to capture the content omitted from resources with deferred representations improve. For example, even though it does not execute JavaScript, Heritrix version 3.1.4¹ does peek into the embedded JavaScript code to extract links where possible [71, 133] (as does Google’s crawler [46]).

Archive-It [42], a subscription-based service that allows customers to build collections of pages in an archive, uses Umbra to handle the archiving of a manually defined list of URI-Rs with deferred representations [235] to mitigate the impact of the increasingly popular practice of using JavaScript and Ajax to load embedded resources.

¹The latest version of Heritrix, v3.1.4, is currently in production [249] at Internet Archive.

The `wget` application [104] downloads a target resource and offers flags for downloading representation dependencies and embedded resources. However, `wget` does not handle deferred representations at all since it does not have a JavaScript engine, causing it to miss resources that are not directly and explicitly referenced in the DOM.

PrEV is a digital library that offers a preservation explorer for Web content [73], similar to Heritrix and the Wayback Machine. It identifies the practice of archiving user-generated resources and content as especially difficult due to the inability to do so automatically and the frequent change rate of such representations and data.

Pandora, Australia’s national archive, has attempted to archive Flash resources that exist in YouTube [314] or other multimedia content [72]. When crawling to gather resources for a collection on a recent election, Pandora was able to extract the URIs for the embedded dynamic content and retrieve the embedded Flash resources for archiving. The target resource is downloaded along with the embedded videos. However, these videos are self-contained multimedia, are only played on the client, and do not change as a result of user interaction. TubeKit is another YouTube crawling toolkit [259]. Everlast [13] is a framework that focuses on improving the user-centric features of archives to increase their usability and value (e.g., adding distributed text search).

PageFreezer [223], Kapow [138], Hanzo [122], and Fetch [91] are for-profit companies that provide capture of Web resources – including those with deferred representations – as a service. These companies offer data extraction and crawling of deferred representations through human-assisted capture and data formatting. All three services advertise the ability to extract data from social media sites, as well.

Transactional archiving also offers a method of archiving resources and data needed in deferred representations. The first attempts at transactional archiving (pageVault [97] and *ttApache* [83]) captured all representations returned to users from a server equipped with the necessary software. These transactional archives operated under the premise “if a user never saw the representation, was it worth archiving?”, and similarly, “shouldn’t all versions of a representation seen by users be archived?” However, they lacked the ability to capture deferred representations and associated post-load resources. The SiteStory Transactional Web Archive [256] can capture Ajax-requested resources and associated data for replay in a deferred

representation. These transactional archives do not significantly impact the performance of a server [56]. During the SiteStory unveiling at the 2012 IIPC Future of the Web Workshop [211], the attendees concluded that a client-side transactional archive would help alleviate current archiving challenges.

WebCite [85] is an on-line, on-demand archival tool that offers real-time archiving of content. When WebCite archives a URI-R, it opaquely rewrites embedded URI-Rs to URI-Ms but does not maintain the original URI-R in the HTTP response headers or in the HTML (as does the Internet Archive). The embedded resources' URI-Rs are not required for WebCite's page-at-a-time archival practices to be successful. For example (Figure 31), when WebCite archives a URI-R, the embedded image identified by a URI-R is rewritten to point to a URI-M. For example, when WebCite archives the URI-R

`http://www.cs.odu.edu/`

the embedded link to the URI-R

`http://www.cs.odu.edu/files/logo-cs.gif`

is changed to

`http://www.webcitation.org/getfile.php?fileid=441da589db4d846925155555e687cb9aacf5627d`

in the memento at URI-M

`http://www.webcitation.org/64ta04WpM.`

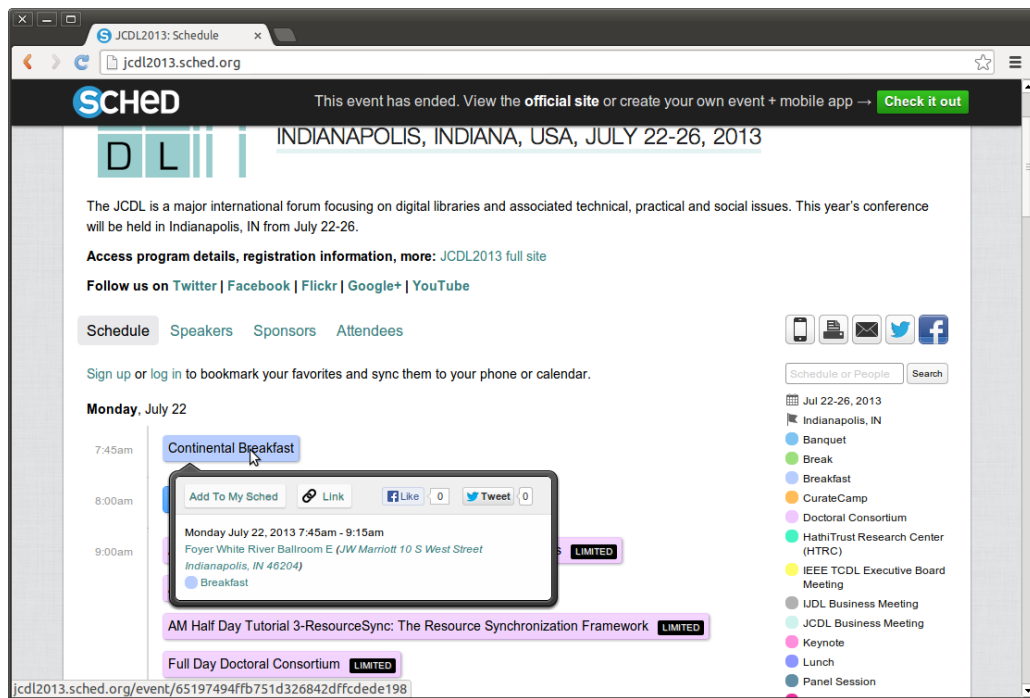
Archive.is also opaquely rewrites URI-Rs to URI-Ms. Archive.is [18], like WebCite, is one of many page-at-a-time archival services that provides on-demand snapshots of a Web page and is Memento-compliant [207]. It creates an image of the page (for permanent, low-fidelity capture) and captures the HTML and associated embedded content in their native formats. Archive.is renders the representation, including content loaded from JavaScript, using headless browsing, a method that

renders a representation and executes client-side events before capturing the final representation. More impressively, it statically archives content like live Twitter feeds instead of allowing the live Web to leak into the mementos – content that WebCite is unable to archive. Neither Flash nor client-side events are captured by Archive.is. We take the example of <http://archive.is/jL9F7> (Figure 30), a memento of <http://jcd12013.sched.org/>. The live resource displays tooltips over the schedule items, while the memento does not provide this functionality [209].

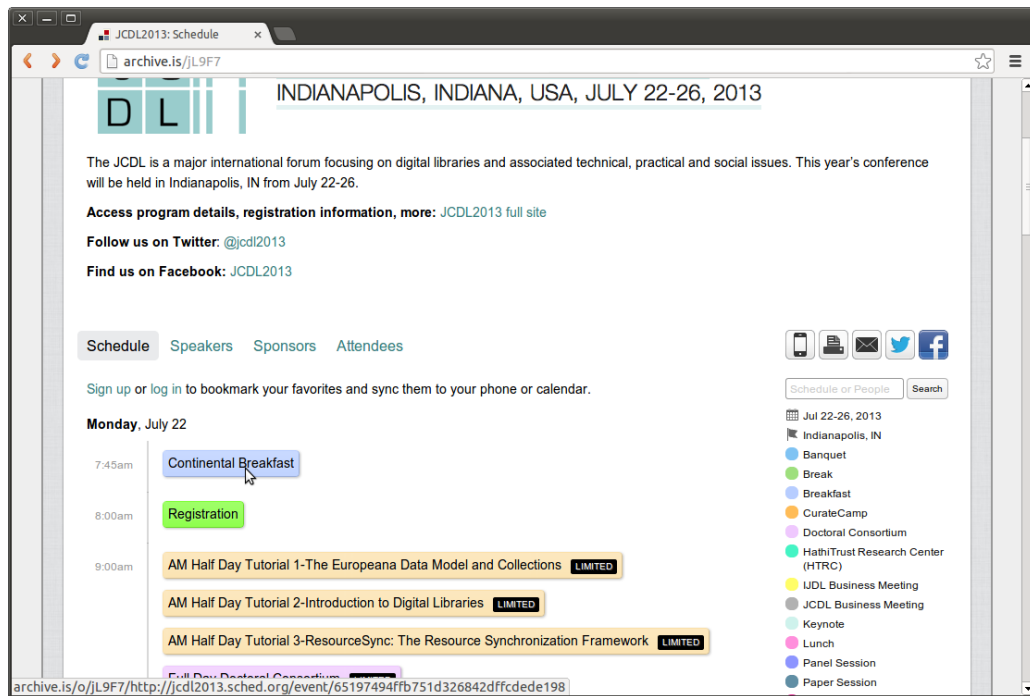
The Internet Archive also offers a “Save Page Now” feature [250] as of October 2013, which is similar to page-at-a-time archival services like WebCite and Archive.is. The Internet Archive rewrites the URI-Rs of embedded resources and links (i.e., anchor tags) to URI-Ms, but also preserves the original semantics and context of the live Web page, and the Wayback Machine provides an option to add “id_” to a URI-M after the memento datetime part of the URI-M (e.g., http://Web.archive.org/web/20090617194641id_/http://www.cnn.com/) to receive the original representation of a memento as it existed at archive time without the URIs rewritten. Additionally, the URI-Ms preserve the original URI-R and memento datetime, allowing us – as humans – to recognize the original URI-R.

Mat Kelly has created several personal archiving tools that focus on user-controlled archiving rather than the automated crawlers. WARCcreate [142, 149] allows WARC files to be created based on a representation in the browser (even those reached through technologies that normally lead to deferred representations); that is, a user can interact with a representation and create a WARC of the current state of the representation. Mink [147] is a browser add-on that uses the Memento Framework to alert the user if a URI-R does not exist in the known archives and allows the user to submit the URI-R to Archive.is and the Internet Archive’s Save Page Now service. Mobile Mink [137] is an Android application developed by Jordan et al. that implements Mink’s functionality on Android mobile devices, helping merge the Mobile and Desktop Webs and increase the archives’ coverage of mobile resources. Kelly also developed WAIL [143] as a suite of tools to make Heritrix and the Wayback Machine easy to deploy and use by individuals.

The Smithsonian’s blog offers tips on designing archivable resources and how to use ArchiveFacebook to archive a subset of one’s social Web presence [76]. ArchiveFacebook [101] is a Mozilla Firefox add-on that captures the user’s Facebook profile and information in its native format with all embedded resources and content but is



(a) The live resource provides on-hover tooltips.



(b) The Archive.is memento does not offer the tool-tip functionality.

FIG. 30: While Archive.is offers a high quality memento, some functionality is missing because the JavaScript is removed during archiving. This and URI rewriting leads to a paradox: archives have to transform HTML in order to “preserve” it.

URI-R:
`http://www.dot.state.al.us/`

```

```

URI Rewrite

URI-M:
`http://www.webcitation.org/6ItLzc9ih`

```

```

FIG. 31: URI re-writing in WebCite converts embedded URI-Rs to URI-Ms.

suitable for personal archiving only [118]; it cannot be used in larger shared archives. In addition, Kelly et al. have studied mechanisms for merging private and public archives [149, 147, 143, 142].

WebRecorder.io allows users to record a browsing session and store the recorded content in a WARC or ARC file. These files can be uploaded to an institutional archive (e.g., the Internet Archive) or played back on a user’s local Wayback Machine installation. Browsertrix [155] uses Selenium to run automation tools to load Web resources directly in the browser. This process allows embedded resources and client-side scripts to be loaded and executed organically. This places the responsibility of evolving with Web technologies on the browser rather than on the archival client. Note that the approach used by Browsertrix and WebRecorder.io (i.e., using Selenium and replaying recorded interactions) is the opposite approach we use in our framework (Chapter 7).

WARCreate, WebRecorder.io, and Browsertrix are examples of client-side and personal archival tools which are focused on individual users’ observed representations rather than a Web-scale, automatic archival framework. Our research focuses on mitigating the impact of deferred representations at Web scale using institutional archiving tools but uses the tools as templates for larger-scale archiving.

Google – an example of a large-scale crawler – is crawling deferred representations that are identified by URIs provided by developers. The developers can create a representation that they feel encompasses their resources. The URI should include the HashBang attribute (Section 2.4) so that the Google crawlers can locate and crawl the page, and also retrieve the client-side state [46, 107, 174, 119]. This effectively provides a deferred representation in static form for a crawler to discover and index. Similarly, Google has proposed a headless browsing solution to query, interact, and render the content of the page for crawling purposes [106].

The CRAWLJax software package [191] attempts to discover all possible states of an Ajax application in order to crawl and index deferred representations. Because Google does not crawl all JavaScript code (partly due to security concerns), this research is an attempt to index these deferred representations. CRAWLJax creates a list of all possible states by examining any user interaction that is possible on a page. It then stores the resulting DOM at each state to allow for crawling. A byproduct of this solution is a history for Ajax applications in which the user can move backwards through the state tree. Mesbah et al. [192] identify browser cookies and HTTP 404s

as problems they have encountered when trying to recreate states for a resource. Similarly, the crawled pages are not sharable without special software installed on the user’s machine. Duda’s work with AJAXSearch [82] takes a similar approach by generating all possible states of a deferred representation and capturing the generated content. Benjamin [30] has taken the approach of providing a canonical method of crawling RIAs for search indexing purposes.

Archival – and other Web-scale – tools have evolved over time to adapt to a changing Web. However, JavaScript and particularly representations with extensive interactivity and descendants remains a challenge for archival tools.

3.2 ARCHIVAL CHALLENGES: THE DEEP, HIDDEN, AND EPHEMERAL WEB

Web archiving is an eternal battle to capture Web content before it changes or disappears forever [43, 179, 26]. The Web not only changes frequently and at varying intervals, but also changes in different ways such as wholesale changes or appended content [3, 100, 92, 168, 240, 27, 28, 81, 280]. Many of the changes on the Web are lost [22]. Weiss [307] and Lawrence et al. [162] estimated the lifetime of a page is between 75 and 100 days. Other work from researchers at Los Alamos National Laboratory has shown that 38% of Web pages are missing or no longer on topic after 5 years [239].

Ainsworth et al. measured the archival coverage of resources sampled from a variety of sources [5] and found that archival coverage varies depending on the sample source. For example, URIs shortened by Bitly users were less frequently archived than those indexed by other services like DMOZ. In a follow-on experiment, SalahEldeen studied the decay of shared data on Twitter [252] and found that up to 11% of content shared over Twitter disappears after 1 year, and 25% disappears after 2 years. SalahEldeen also showed that mementos disappear from the archives, and content that was observed as missing can reappear in the future [254, 55]. These studies highlight the ephemeral nature of shared content and demonstrate that shared links are often not archived.

However, updates to Web resources may be missed by crawlers; the resources may not be missing as in SalahEldeen’s and Ainsworth’s studies, but instead the crawler may miss updates between visits to the URI-R. Cho studied how the Web is changing, and how crawlers can adapt crawl policies to maintain the latest versions of evolving

resources [65, 66] and found that – counter to intuition – archives can maintain the most current mementos by archiving resources that do not change frequently. Olston created a recrawl strategy for pages based on content change frequency [218]. Koehler also studied Web page change frequency over a 4-year period [153], finding that older pages change less frequently than younger pages. Radinsky and Bennett studied the change frequency of resources and predicted their future updates [233], showing that past frequency is the best method for predicting future change. Manku et al. created a fuzzy matching algorithm for near duplicate detection during Web crawls [176]. Each of these studies provides insight into how Web resources evolve and how crawlers can adapt to keep up with the changing Web.

Several works have attempted to estimate the size of the “deep Web.” In 2001, Bergman estimated the size of the deep Web is 500 times the size of the surface Web [32], which has been estimated at over 11.5 billion pages [115]. McCown et al. measured that 21% of a collection of 3.3 million URI-Rs advertised by the repositories using OAI-PMH were not indexed by crawlers [183]. This work demonstrates that search engines were not indexing URI-Rs that were known and advertised by the repositories (and therefore, not “deep” URI-Rs in the sense that they are not discoverable by Web crawlers). Ast extended Bergman’s work by proposing approaches using a conventional Web crawler (i.e., one not used for preservation) to capture content not linked to by individual URIs [19].

Past works have focused on crawling and discovering deep Web resources. He et al. worked to sample and index the deep Web [125]. Yeye utilized server query logs and URL templates to discover deep content [126]. This prototype successfully collected the RIAs and associated states. Likarish and Jung crawled the deep Web to create a collection of malicious JavaScript using Heritrix and comparing captured scripts to blacklisted content [167]. Ntoulas et al. captured the textual hidden Web with a guess-and-check method of supplying server- and client-side parameters [214].

What we have defined as *deferred representations* are classified by many as deep Web content [32, 175] due to their inability to be automatically crawled. Deferred representations are not necessarily part of the deep Web because they cannot be discovered, but instead end up classified as deep Web resources because they require interaction not supported by automated crawlers (e.g., authentication past a login wall [141]). However, even if archived, deferred representations often lead to leakage [54, 209]. Bossetta and Segesten performed a study on linking between sites and

political parties, but they concluded that their study was incorrect because some of the sites constructed links and URIs in JavaScript [40]; the Issuecrawler² tool they used does not execute JavaScript. Bossetta and Segesten have identified crawlers that do not run JavaScript as a critical limitation to Web studies and recommend a crawling approach that can execute JavaScript to increase the fidelity of studies such as theirs. Our proposed framework focuses on executing JavaScript as an answer to this recommendation.

Mesbah et al. have performed several experiments regarding crawling and indexing deferred representations of resources [195, 191, 190]. These works have focused mainly on search engine indexing and automatic testing [194, 193] rather than archiving, but serve to illustrate the pervasive problem of deferred representations. Dincturk et al. constructed a model for crawling RIAs by discovering all possible client-side states and identifying the simplest possible state machine to represent the states [80, 58, 79]. We adapt the Dincturk et al. Hypercube model for use in our two-tiered framework for crawling deferred representations (Chapter 7).

Raghavan et al. discussed the inclusion of deferred representations in the deep Web [234]. They utilize human-assisted crawling and caching to capture client-side content. Their method can differentiate between mobile content and content loaded by a browser, opening a new capability and market for archiving. However, this method does not capture the content or data passed to and from the server to generate the deferred representations, and cannot replay client-side events to reconstruct descendants.

Similarly, Fleiss performed an analysis of how client-side dynamism is impacting search engines, specifically due to the popularity and lack of crawlability [99]. His work mentions that resources such as content management systems (CMS) are producing good, sometimes viral, content, but they do not have cool URIs and are asynchronous; data is constantly flowing to and from a single, self contained resource as opposed to the traditional model of data only flowing when a resource is loaded (i.e., what we refer to as deferred representations). Search engines are not suited to crawling and caching these deferred representations. Even more problematic are resources such as Second Life [169], which is an example of applications that exist for the sole purpose of interaction. A game walk-through approach would be a more effective method of archiving an application such as Second Life, which is highly

²<http://www.govcom.org/>

dependent upon user interaction.

Our approaches in this dissertation help uncover a previously unarchivable portion of the deep Web. By understanding the challenges faced by prior efforts, we adapt our approach to mitigate the archival challenges introduced by JavaScript.

3.3 WHAT MAKES A REPRESENTATION?

As we discuss the capture and replay of client-side code and representations, it is important to discuss how these representations are constructed and the technologies that influence representations for the benefit of the user and at the expense of Web archiving.

Bucy demonstrated the relationships between page complexity and site traffic showing that popular pages are more complex [59]. Push/pull technologies were essential for a movement from HTML to a Java platform. Complexity in Bucy’s study refers to the number of links and ads within the content of a page. Bucy finds that 40% of sites have ads, 79% use frames, 33% have dynamic content, and 15% have video. Ihm and Pai showed that this trend has lasted into 2011, and that much of the newly introduced complexities such as Flash cannot be adequately cached [130]. Ha and James found that of the links in a page, most come from the host domain [117]. Our previous work (focused on using Flash movies to enhance Intelligent Tutoring Systems) shows that Flash movies can initiate client-side events or make use of JavaScript and Ajax [53, 49]. Flash is being increasingly disallowed by browsers [244].

Of particular importance to our work is a deep understanding of Ajax and its impact on client-side representations [319]. The first large-scale implementation of Ajax was in Google Maps in 2005, but Ajax was officially added as a standard in 2006 [309]. Mesbah et al. have worked to enable RESTful Ajax applications, such as the use of stateful servers [192]. Pierce et al. utilized microformats, Atom feeds, and other exploitations of the Web 2.0 to make Ajax representations more persistent [227].

There are several efforts aimed at identifying unique representations that result from modern client-side technologies. These efforts focus on HashBang URIs [301, 281, 108] (as discussed in Section 2.4), which represent client-side state. HTML5 also offers client-side storage to specify and store the client-side state [279]. Google provides client-side state storage that enables back-button functionality by keeping

a stack of state changes [212, 265, 266].

As mentioned in Section 2.2.3, Ajax has enabled the creation and popularity of personalized and deferred representations [15, 221]. Its shortcomings are well understood [70], and attempts have been made to solve issues with asynchronous communication [210]. Doloto [172], for example, introduces blocking loads to ensure synchronous communication in Ajax to prevent unnecessary load on the network infrastructure by loading unneeded content.

Other works focus on extracting and visualizing behavior on the client. Chung and Periera created representations of Timed Petri Nets using SMIL [69]. Similar to SMIL, Puerta and Eisenstein established a model for representing interaction data on the client [231]. Webquilt [128, 127, 305] is a framework for capturing user interactions during a browsing session and subsequently visualizing the user’s experience. Webquilt creates this functionality with a proxy logger, action inferencer, graph merger, graph layout, and visualization components.

Extracting the code that runs in RIAs [177] helps map functionality to code blocks. Browser fingerprinting is an emerging area of study. This is an effort to identify the **user-agent** being used by the Web user for tracking and user experience purposes [1]. Each of these efforts helps draw conclusions from browsing context.

Gyllstrom et al. studied the impact of JavaScript on representations over time with specific attention to the types of representation changes that JavaScript performs [116]. Their work measured the amount of JavaScript that resources implement over time, showing a continual increase since 1996. Their work also focuses on DOM changes and associate representation modification, while our work focuses on client-side state changes and embedded resources.

Each of these works studies the impact of JavaScript on representation and resource state and how the representations can be captured, cached, and replayed. We leverage these works to understand the types of challenges JavaScript introduces, as well as build on past attempts to mitigate these challenges.

3.4 MEASURING ARCHIVE QUALITY

Archivists use archival quality measurements to understand how well they are archiving their target resources. Many approaches to measuring archival quality are used by researchers and archivists. For example, Spaniol measured the quality of Web archives based on matching crawler strategies with resource change rates

[275, 274, 77]. Gray and Martin created a framework for high quality mementos and assessing memento quality by measuring the missing embedded resources [113]. These studies focused on memento completeness and site coverage.

Kelly et al. studied the factors influencing archivability, including accessibility standards and their impact on memento completeness [146]. Using the archives, Kelly demonstrated that movement away from accessibility standards to heavier reliance on JavaScript has led to reduced archivability. Kelly et al. also created an Acid Test utility to evaluate mementos created by archival tools and services for archival accuracy [148].

Other efforts have focused on standards as they relate to archivability. Banos et al. created Archiveready³ to evaluate expected archival success based on adherence to standards for the purpose of assigning a resource archivability score [21, 20]. Archiveready provides an interpretation and numerical measure of archivability. The Archiveready service analyzes compliance to standards, number of externally hosted resources, and format of the resource HTML and CSS to establish an archivability measure.

The PRISM project [150] assessed the practices threatening the preservation of Web resources as they relate to accessibility (such as not using modern Web archiving tools and proper replay services or having out of date software). Virtual Remote Control [186] created a framework for identifying archival targets and managing at-risk resources. Each of these efforts assesses an impact on archive quality.

However, standards and mere completeness are misleading measures of human-perceived quality. Fersini et al. studied the importance of information blocks of a rendered Web page, finding that blocks with more images are more important [90]. Singh et al. found that multimedia within a page is essential for user understanding [267]. Ye et al. found that the information blocks close to the center of the viewport contain important information, while “noise” – or unimportant content – occurs on the fringes or edges of the page [312]. Kohlschütter et al. also found that important content was located in the center of pages [154]. Centrality is a way for authors to convey importance of information to their users. For example, images in the center of the viewport are more important or contribute more to the users’ understanding of a page than those positions on the fringes or outside the viewport of a page. These works establish a relative importance of embedded resources and content based on

³<http://archiveready.com/>

where in the viewport the resource or content appears, and results in a foundation for our measurement of the quality of mementos.

Zhang et al. studied human perception and human ability to recognize differences in images effectively determining human perception limitations for images at the pixel level [316]. Similarly, Rademacher et al. used human perception to identify the visual factors that distinguished computer generated images from photographs [232]. Song et al. outlined an algorithm for determining the importance of sections of Web pages based on their content, size, and position [273]. These works establish methods of human-assessed quality; our prior works (Chapter 6) use similar methods of human evaluations of memento quality.

Ainsworth et al. used a different approach to measuring memento quality by looking at the temporal coherence of mementos and a browsing session within the archives would temporally drift (i.e., a memento-datetime would not stay rooted) during walks through the archives [7]. Additionally, Ainsworth et al. found that embedded resources would have memento datetimes different than the root memento, leading to temporal violations and – as a result – the archives provide a memento of a resource that never existed, or are *prima facie violative*. As such, Ainsworth et al. defined a memento quality measure beyond a simple boolean notion of whether an embedded resource returns an HTTP 200, but instead consider the memento datetime of the embedded resource and identify temporal violations within the archives. Rather than assigning a quality measure according to temporal drift, we assign value according to the perceived importance of the embedded resource (Chapter 6).

McCown performed several comprehensive studies involving reconstructing Web pages from the archives’ offerings [184, 182]. He created the Warrick program [181] to reconstruct missing Web sites using Web repositories. To measure Warrick’s performance, McCown created a method for evaluating the quality of a reconstructed site according to how much of the site was missed during reconstruction. At the time of McCown’s work, the archival landscape was much different than the landscape at the authoring of this dissertation. Search engine caches were available for Web site recovery, and the Internet Archive was the only major archive from which mementos could be recovered. Additionally, the Internet Archive had a six month quarantine period before the mementos would be made available, and the Memento Framework was not yet defined. McCown’s work measured quality according to the completeness of a recovery of an entire Web *site* rather than a Web *page*. McCown measured

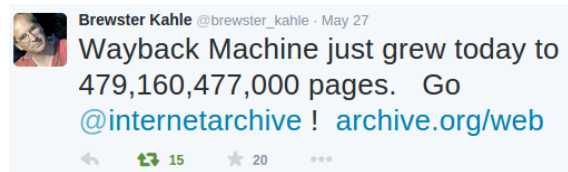


FIG. 32: Brewster Kahle tweeted that the Internet Archive’s Wayback Machine contains roughly 479 billion pages (2014-05-27).

quality by identifying recoveries that missed resources, that found new resources, and successful recoveries that were known and expected to be recovered by the service.

Of the archivists that conduct quality assurance and were surveyed by Reyes et al. [238], 100% use a manual process. The Internet Archive alone boasts 479+ billion Web pages in its archive at the time of authoring this dissertation (Figure 32), which is far larger than can be evaluated through human methods. An algorithm to automatically assess human perception of archived page quality would significantly decrease the necessary human involvement in the quality assurance process, potentially increasing the accuracy while reducing the cost of quality assurance efforts. Towards this goal, we investigated the impact of missing embedded resources on users’ perceived utility of Web resources, quantifying the impact of missing embedded resources in mementos [51, 50].

By measuring memento quality, we can better assess the success of our archival approaches and identify areas in which we should improve.

3.5 CACHING JAVASCRIPT-DEPENDENT REPRESENTATIONS

Caching information on the Web is not a new realm of study [303, 318, 268, 217, 55], but capturing client-side interactions and data transactions is more recent. Resources have been traditionally composed by reusing cached or stored content, similar to our research goals of archiving deferred representations (Research Question 3 in Chapter 1). Several research efforts have focused on caching the data that is transmitted to the client from the server by intercepting and analyzing the Ajax calls to and from the client. ActiveCache is a project that attempts to cache the server-side data that is sent to the client as a result of Ajax calls [60]. The goal of the project is to reduce the amount of traffic between the client and server. The software uses delta compression, which will add data to a cache entry if it adds data and will

only ask for the data from the server that is missing from the request. MapJAX has similar goals, but provides shared data structures with the server for the user to implement [202]. This obfuscates the process of requesting and interpreting the Ajax data and allows the developers to assume the data is all local. The data that is transmitted from the server is cached and some is even prefetched to distribute the latencies of data transfer across the entire interaction interval. MapJAX was implemented in a mapping application, similar to Google Maps [110].

AjaxScope, also developed by Kiciman and Livshits from Microsoft, is designed to reduce the overhead of transmitting data from the server [151]. AjaxScope is a proxy that rewrites the client-side JavaScript to reduce the overhead of consistently communicating with the server. It also monitors the traffic sent to and from the client by the server to allow developers to analyze the amount of performed interactions. Kiciman and Livshits propose a caching mechanism for the data transmitted to the client from the server. They hypothesize this will reduce the response time when requesting data. However, archiving the Ajax transactions is outside the scope of the AjaxScope research. Kiciman and Livshits note that the high volume of differing data may induce thrashing in the cache. This project illustrates the growing need to monitor user interactions on the client, and the ability to store the data that is sent to the client for the purpose of performance improvement.

Sivasubramanian has developed a solution centered around caching content generated dynamically on the server [268]. The major contribution of this research is a recommendation to only cache content based on data needs and load considerations, such as the caching used for distribution by Content Delivery Networks. Karri implemented a method of caching dynamically loaded DOM elements in a Firefox cache [140]. This work demonstrates that resources can be reused when composing higher-order resources. Olston investigated the synchronization of distributed caches [219]. This synchronization uses the resource change rate to establish update policies in the cache.

Squid [308] is a widely deployed application that exploits a browser's ability to cache external JavaScript pages to load User Interface (UI) elements more efficiently [140]. The project is primarily known as a cache and reverse proxy, but also transfers dynamically generated elements into JavaScript to allow the browser to cache the rendered elements for quicker load times. Similarly, HTML5 can be used by Cartagen to render data as a map on the client browser [41]. The map tiles are cached on the

client-side for quicker rendering. SyncKit has the same goals of caching client-side data transfers and interactions [31]. These interactions can be reused to generate other resources. HTML5 provides a relational database for the client, and SyncKit stores server-side data (based on last-modified HTTP headers) in the client-side database to alleviate the server processing and delivery times. However, this method must be implemented by developers. DBProxy archives only the raw data being transmitted by the server to the client [12]. These approaches to caching show the interest, need, and current capabilities to archive server-side data and dynamic content.

Another Microsoft project aimed at replaying JavaScript interactions for debugging purposes is Gulfstream [171]. Gulfstream stores content offline, allowing interactions to be replayed within the code for debugging. Further, the authors find that the updates done to the content of the page is minimal, and the error is negligible for most interactions if a cached version of the data is transmitted instead of a fresh version. Gulfstream also calculates deltas in the data to only load the necessary data when it is requested.

While these efforts focus on caching for performance and debugging, our research focuses on archiving in the context of an entire representation (first- and second-order expression). Without establishing context, the independent embedded resources cannot contribute to the quality of the memento.

3.6 MONITORING THE CLIENT FOR DEBUGGING AND SECURITY

The majority of past research in capturing deferred representations from the client has been focused on the debugging or security of Web applications. Since JavaScript and Ajax applications are *stateful* while HTTP is *stateless* (see Section 2.2.3), it is difficult for application developers to determine the failure points of applications and replicate the errors for debugging purposes. Additionally, JavaScript applications have been traditionally viewed as security threats [315, 121, 167, 78, 237, 196]. To evaluate the security of such applications, researchers have analyzed the potential states of JavaScript applications to determine if they contain a security violation.

Client-side monitoring is studied to understand user actions and tendencies with the goal of improving Web users' experiences [88, 200, 276]. Mobasher et al. [200] specifically identify the goals of client-side monitoring as: not interfering with normal

browsing; data should be collected with respect to specific users; developers should be able to determine the granularity of collected data; and a framework should be extensible enough to monitor a variety of client-side applications and activities.

Fenstermacher and Ginsburg also note that the majority of Web mined data resides on servers and that data native to the client is not captured by crawlers [89]. To begin recording user interactions with client-side representations, the authors developed an Internet Explorer framework for modeling and capturing client-side events. Their work is an early, yet limited, effort to capture data residing on the client – interactions such as Web form submissions are missed. Understanding and modeling user behavior can help uncover client-side states and events to more thoroughly test, debug, and interact with a page.

3.6.1 DETECTING MALWARE

Malware detection is done by monitoring client-side code, a shared goal of our research. Sabre is a system that monitors user interaction with Web forms – a subset of interaction missed by Fenstermacher and Ginsburg’s work – and JavaScript-enabled elements of a representation. Sabre alerts the user if there is an unsafe interaction with possibly sensitive information [78]. It analyzes the data sent to the user, and notifies the user if the data looks sensitive or malicious. This research targets mostly browser extensions and add-ons, but supports the efforts of monitoring and analyzing the data being transmitted from the server to the user.

Similar efforts have attempted alternate methods of detecting malware or corrupted data [315, 121, 167] by monitoring DOM changes, XPCOM communication, JavaScript activity and events, and all data transmitted to and from the server. Others [62] go as far as offering proxies that allow the captured interactions to replay on the client.

BrowserShield and other solutions [237, 78] shift their focus to the code being loaded by the application on the client and the information being transmitted from the client. Much of the code that loads during run-time will make or be the target of Ajax calls, access data in cookies, or perform local captures of client interactions (such as form entry). BrowserShield relies on an external server to validate the findings of the client-side application.

Ripley [157] also makes use of a server to monitor the client and all user interactions to ensure client-side applications are secure by capturing user interactions

and replaying them in a controlled environment. This helps identify the security vulnerabilities in an application. The captured representations and interactions are modified by Ripley significantly to ensure proper execution.

3.6.2 DEBUGGING AND ERROR TRACKING

There have been a plethora of efforts to automatically test and recognize bugs in client-side Web applications in an effort to test and debug Web applications similar to methods for traditionally testing standalone applications.

Choudhary and Orso monitored JavaScript code and detected failures [67, 68]. They have also experimented with monitoring worker threads in HTML5. Mesbah has been active in testing Ajax applications; Atusa is a JavaScript integrity checking tool that monitors client-side JavaScript errors [194]. Atusa establishes a test suite and monitors the DOM for validity (but has trouble with the back button on browsers). Some of the failures detected by Atusa include load failures for embedded content requested by Ajax. Zheng et al. tracked bugs that occur due to the asynchronous nature of Ajax [317].

Mugshot is a tool developed by Microsoft Research’s Mickens et al. for debugging Web applications [197]. Mugshot allows developers to replay state changes of a Web application. The author of the Web application can include an external JavaScript file that will gather and log diagnostics about the state progression and interactions of the application from the user’s machine, and replay the interaction on another machine. However, if there is embedded content (such as a frame) or other embedded media included on the page, the JavaScript must be included in those resources in order to capture user interactions with them, as well. The external JavaScript file allows the replay to run without an add-on, but with solely JavaScript. The authors describe the replay as a “VCR”-like interaction and play back. Mugshot listens for user interactions (such as button clicks) and state changes of the application, including DOM events and interrupts. This system provides for none of archiving, sharing via URI, and composing resources and is meant only as a developer tool. Ripley performs the same state monitoring task on the server [157]. The server recreates the interactions of users in order to automatically test each state of the software.

Rumadai is an Integrated Development Environment (IDE) tool that records and

replays client-side events triggered by Web sites and dynamic content [313]. JavaScript is injected into the client-side code to monitor the events and play them back for testing purposes. The Ajax requests and responses are cached at the client, meaning they are overwritten and never captured for long-term storage. Embedded content is stored as byte-streams to ensure they exist locally and are not dependent on an external server. WaRR is a Web application Recorder and Replayer for Web applications [14]. WaRR is a standalone tool that allows developers to test client-side user events. However, the tool only records a subset of user events (keystrokes, dragging, mouse clicks). It has been tested for Google Sites, GMail, Yahoo Authentication, and Google Docs.

3.6.3 USER INTERACTIONS

Simulating user interaction is essential to debugging and automatically testing Web applications. Understanding user interactions helps determine what elements of Web pages will modify the DOM or initiate Ajax requests for new, post-load resources. Like those discussed in Section 3.7, researchers have developed tools record interactions and the resulting DOM changes that occur or Ajax that is sent and requested from the client and server [24].

Bartoli et al. recorded and replayed navigations on Ajax-heavy resources [23]. The utility records interactions and replays the trace in an effort to test user interfaces. Their goal was to overcome the difficulties in automated testing and programmatic interaction with client-side interfaces introduced by Ajax.

Similarly, FireCrystal identifies the interactive portions of a page and shows the associated code [220]. It also records any events that occur within the code and interactive elements and identify any DOM changes that result from these events. This solution cannot work with Flash or other compiled languages on the client.

Rather than record interactions, Gorniak worked to predict future user actions by observing past interactions by other users [112]. This work modeled users as agents based on their interactions and clients based on potential state transitions. The agents would make the next most probable state transition, indicating what action the user is expected to take.

Leveraging user interaction patterns, Palmieri et al. generated agents to collect hidden Web pages by identifying navigation elements from current selections [159]. They represented the navigation patterns from these agents as a directed graph and

would train the agents to interact with navigation elements. This includes forms, which would be filled out based on repository attributes. Drop-down menus would be randomly selected by the agents to spawn new pages. This method generated 80% of all potential pages.

The efforts in debugging and monitoring the security of deferred representations proves particularly useful for our research, serving as the basis for our approach with identifying client-side states during our proposed archival framework. We build on these efforts to construct an approach for monitoring and replaying client-side state in an archive rather than on the live Web (Section 7.2).

3.7 SESSION REPLAY AND SHARING

Capturing and mining data from Web user sessions is important for usability studies, surf patterns, and behavior models [123, 124]. The capture of such client-side activity is increasingly important for automated crawling, capture of client-side content, and archiving of user experiences (i.e., second-order expression). In concept, these tools that enable session capture and replay are similar to the archives' goals of uncovering and storing a representation. We work to take the ideas behind these efforts and tools and adapt them to meet the archives' needs. As the Web continues to cultivate resources that change on the client and as a result of user interaction, archives need to archive the associated representations. There have been several attempts at providing solutions to subsets of the problem.

While they focus on archiving simulations rather than Web resources, Thain et al. [282, 188, 187] created tools and a framework for storing the context under which a simulation is run for reproduceable experiments. Their work stores the inputs, outputs, code, data, and context (just as we aim to do with deferred representations) to help future researchers re-run, or *replay*, simulations.

Webtour [257] is a system that records and replays multimedia annotations on Web documents. The purpose of Webtour is to provide asynchronous co-browsing of content. The recordable and replayable events on the client include mouse activities, audio/video activity, and hyperlink traversals, each of which is stored in a remote database. Special software is required to replay the interaction – the content captured is not native to the Web even though they are identified by a URI. That is, the recording sessions require specialized software before the interactions can be replayed, and the software identifies which stream of interactions to replay with a URI pointing

to the replay script.

Nino et al. have extensively investigated capture and replay of navigation sessions [213]. Their work began with capturing HTTP headers and the initial DOM using a Firefox add-on. Subsequent changes to the DOM were not captured, but instead the HTTP headers were replayed in an attempt to recreate the client-side activity of the page. This work led to CARENA, a tool to capture and replay Web navigation sessions. These replays were aimed at measuring latency for benchmarking purposes. The client-side headers were stored as XML and were replayed to simulate the real-time load of images, stylesheets, and other embedded content. Sessions exclude user activity; they are also recorded, then loaded to replay the content.

Similar to capturing and replaying client-side state, Total Recall [225] is a solution that allows for back and forward button clicks in Ajax applications. To provide this capability, the entire client-side DOM is written to cache, and custom back and forward buttons are provided as part of a Mozilla add-on to allow the user to navigate back and forth between states. However, a wholesale state change is possible in this model. This project also mentions that tabbed browsing can play a large factor in state.

ActionShot is another VCR-like add-on for browsers that automatically records user interactions for data mining, study, and replay [166]. This solution specializes in capturing client-side activity at fine-grained levels, such as click locations, targets, and resulting events.

Koala is a solution in which human-written natural language scripts can be played to automate processes in Web applications [170]. This project is meant to facilitate often-used user interactions for user convenience. However, the scripts must be downloaded and played from a wiki.

Memex is a system that records user interactions with Web pages [61]. It aims to develop a series of clickstreams for groups of similar interests. The system archives a search query from the user and also archives the sequence of pages a user visits. The system can allow for private and public browsing streams, and public streams can be shared with users. The Memex paper introduces a notion of public and private browsing, and sharing the user's activity stream.

Lowet's research [173] provides a way for distributed users to view deferred representations through a browser add-on. The events and DOM changes at each client are shared across the network to synchronize each user's view. Lowet mentions this

is like planning a trip from two different computers with Google Maps. The solution relies on a master-slave relationship between clients to participate and only stores the page's DOM, not the representation or any composite resources. Rodriguez provided a similar solution in which users can access content simultaneously on the Web [243]. The Dandelion system provides a method of collaboratively editing wiki resources [63]. These solutions are examples of live composition.

Instead of simultaneous interaction, Eyebrowse is a tool to share browsing trails – a social network for sharing resources that are visited in the pursuit of a target piece of information. This becomes important for information sharing and information transfer. However, there is no URI for sharing an interaction stream, and the browsing has no context. The interactions are simply a series of URIs that are listed by URI and title.

Rather than an interaction stream, Walden's Paths [39, 262, 139, 263, 222, 102, 74] is a research effort to establish a customized, guided tour of the Web for sharing and reuse. The tool, aimed at students in educational settings, provides a linear browsing path for Web users to follow, explore, deviate from, and ultimately return to the specified path.

The notion of spatial navigation between sites is not new. Temporal navigation is a bit harder to comprehend, and is gaining traction in the research community. Work has been done by several researchers to visualize temporal data [272, 96]. Other works dealing have modeled and recorded temporal sequences of communication with respect to wall-clock time [216, 261, 29].

Song et al. provided a service for capturing and sharing the state of a resource [271]. The model for this work begins with interacting with a page on one machine and offers the ability to move to another machine but load the state of the resource from the first machine. To establish the client-side state, the cookies, JavaScript events, DOM and other aspects are captured and recorded on the client. The state snapshots are then shared over a server and loaded on the other client. This process uses URL rewriting on external resources to ensure they remain consistent when sharing the session. There are some shortcomings with this approach. For example, JavaScript variables may contribute to the client-side state, and will be reinitialized when loading the representation on another client. Also, some features of a page, such as embedded content, may expire or be unavailable on subsequent loads. An approach focused more on archiving is necessary to achieve a fully sharable session.

Related to the idea of sharing an interactive session, our approach proposes a method of archiving the end result of an interactive session as opposed to the intermediary interactions, themselves.

3.8 SUMMARY

In this chapter, we have presented an overview of the current landscape of Web archiving, along with a discussion of the perennial challenges associated with Web-scale archiving and crawling. The principles upon which we build in later chapters are not limited to Web archiving; we leverage prior research in the areas of search engine crawling, Web user modeling, malware detection, and client-side automated testing to construct our proposed framework for archiving deferred representations. With a common knowledge of the current archival tools and the current state of archiving, we proceed with a discussion and proposed solution to the challenges of archiving deferred representations in the next chapters.

CHAPTER 4

UNDERSTANDING THE CHALLENGES OF ARCHIVING JAVASCRIPT

With a common understanding of the underlying technical foundations and an introduction to the difficulty archiving deferred representations, we describe in detail our efforts toward mitigating the challenges caused by JavaScript, and a framework for archiving deferred representations for researchers to use moving forward. In this chapter, we identify more specific and technical challenges that JavaScript introduces for the archives. In Section 4.1, we revisit the SOPA protests introduced in Chapter 1. We show that even JavaScript designed to help link rot causes archival challenges (Section 4.2). Section 4.3 discusses *leakage* of the live Web into the archives and the role JavaScript plays in the phenomenon. In Section 4.4 we describe, through example and investigation, our prior work identifying potential failures of current archival indexing systems. These sections elaborate on the challenges introduced by JavaScript and provide an initial foundation on which we build our framework (described in Chapters 5 - 7).

4.1 REVISITING THE #SOPABLACKOUT: TECHNICAL DETAILS

With an established common understanding of the background information and detailed workings of archiving, JavaScript, Ajax, and HTTP, we will revisit the SOPA Blackout Case Study introduced in Section 1.2. In this finer-grained discussion, we investigate how and why JavaScript and Ajax make the representations of the protest unarchivable.

4.1.1 CRAIGSLIST: REVISITED

As mentioned in Section 1.2.1, Craigslist protested SOPA by providing a countdown at the bottom of a splash page. This countdown used JavaScript embedded in the HTML to countdown and fade in (shown in the code in Figure 33) a link to enter the site.

```

1  <script type="text/javascript">
2      var obj1 = document.getElementById("fade1");
3      obj1.style.display = "none";
4
5      window.onload = function() {
6          startCountDown(10, 1000, myFunction);
7      }
8
9      function startCountDown(i, p, f) {
10         var pause = p;
11         var fn = f;
12         var countDownObj = document.getElementById("countdown");
13         countDownObj.count = function(i) {
14             countDownObj.innerHTML = i;
15             if (i == 0) {
16                 fn();
17                 return;
18             }
19             setTimeout(function() {
20                 countDownObj.count(i - 1);
21             }, pause);
22         }
23         countDownObj.count(i);
24     }
25
26     function myFunction() {
27         var countDownObj = document.getElementById("countdown");
28         countDownObj.style.display = "none";
29         $("div:hidden:first").fadeIn(3000, function () {
30             $("span").fadeIn(100);
31         });
32     }
33 </script>

```

FIG. 33: The JavaScript embedded in the HTML provides the countdown functionality.

The countdown behavior is archived along with the page content because the JavaScript code creating the countdown is archived by Heritrix and is available when the memento’s `onload` event fires on the client and subsequent `startCountDown` code is executed. However, the link that JavaScript embeds in the DOM and appears at the bottom of the screen dereferences to the live version of Craigslist. Notice that the live Craigslist page has no reference to the SOPA protest (Figure 5(b), Section 1.2). Since WebCite (which created the mementos in Figures 5(a) and 5(b)) is a page-at-a-time archival service, it only archives the initial representation and all embedded resources, meaning the linked Craigslist protest page is missed during archiving. While not a deferred representation, the run-time addition of the URI-R makes archiving the representation, as well as the target of the link, impossible with an automatic, web-scale archiving tool.

4.1.2 WIKIPEDIA: REVISITED

As mentioned in Section 1.2.2, Wikipedia protested SOPA by covering their content with a splash page. The mementos of the Wikipedia protest do not have an accurate representation of the protest and are missing the splash page that is loaded via JavaScript, making the Wikipedia site during the SOPA protest a deferred representation.

Recall that Wikipedia maintains a live version of the protest splash page. To investigate the cause of the missing splash page further, we requested (on November 27, 2013) that WebCite archive the current version of the Wikipedia blackout page. This memento does not capture the splash page, either.

When looking through the client-side HTML of the Wikipedia mementos we reference, there is no mention of the splash page protesting SOPA. The common Web user may recognize that clicking the browser’s “Stop” button prevents the splash page from appearing. We hypothesize (and show) that JavaScript is responsible for loading the splash page. JavaScript loads the image needed for the splash page as a result of an `onload` event on the client. Since the archiving tools have no way of executing the event, they have no way of discovering and archiving the image.

When we load the live blackout resource, we see that there are several files loaded by Wikipedia. Some of the JavaScript files return a 403 Forbidden response since they are blocked by the Wikipedia robots.txt file (Figure 34).

Specifically, the robots.txt file preventing these resources from being archived is

Name	Method	Status	Type	Initiator	Size
http://web.archive.org/web/20130824022954/http://en.wikipedia.org/?banner=blackout-41	GET	200 OK	text/css	web.archive.org/6	294 B
/static/css	GET	302 Moved Temporarily	text/html	web.archive.org/24	777 B
load.php?debug=f&startonly=scripts&skin=vector&*	GET	302 Moved Temporarily	text/html	web.archive.org/28	787 B
load.php?debug=f&startonly=scripts&skin=vector&*	GET	302 Moved Temporarily	text/html	http://web.archive.org/web/2013...	359 B
load.php?debug=f&startonly=scripts&skin=vector&*	GET	302 Moved Temporarily	text/html	web.archive.org/791	781 B
load.php?debug=f&startonly=scripts&skin=vector&*	GET	403 Forbidden	text/html	http://web.archive.org/save/_em...	305 B
load.php?debug=f&startonly=scripts&skin=vector&*	GET	403 Forbidden	text/html	http://web.archive.org/save/_em...	305 B

FIG. 34: Google Chrome's developer console showing the resources requested by `http://web.archive.org/web/20130824022954/http://en.wikipedia.org/?banner=blackout` and their associated response codes.

```

1 GET /w/index.php?title=Special%3ABannerLoader&banner=blackout
2     &campaign=none&uselang=en&
3     db=enwiki&project=wikipedia&country=US&device=desktop
4     HTTP/1.1
5 Host: meta.wikimedia.org
6 Connection: keep-alive
7 Cache-Control: max-age=0
8 Accept: */*
9 User-Agent: Mozilla/5.0 (X11; Linux i686)
10    AppleWebKit/537.36 (KHTML, like Gecko)
11    Chrome/29.0.1547.76 Safari/537.36
12 Referer: http://en.wikipedia.org/?banner=blackout
13 Accept-Encoding: gzip,deflate,sdch
14 Accept-Language: en-US,en;q=0.8

```

FIG. 35: An HTTP Request for the blackout banner.

<http://bits.wikimedia.org/robots.txt>. The robots.txt file is archived, as well, at http://web.archive.org/web/*/http://bits.wikimedia.org/robots.txt¹.

We will look at one specific HTTP request for a PHP page with embedded JavaScript in Figure 35. This JavaScript contains code defining a function that adds CSS to the page, overlaying an image as a splash page and overlays the associated text on the image, shown in Figure 36².

Without the execution of the `insertBanner` function, the archival tools will not know to archive the image of the splash page (`WP_SOPA_Splash_Full.jpg`) or the overlayed text. In this example, Wikimedia is constructing the URI of the image and using Ajax to request the splash page resource (Figure 37), making this a deferred representation.

The blackout image is available in the Internet Archive (as a memento of the currently available blackout image), but the mementos in the Wayback Machine do not attempt to load it³. Without the execution of the client-side JavaScript and subsequent capture of the splash screen, the SOPA blackout protest is neither discovered nor recorded by the archival service.

¹For reference, Sun et al. provided a study of the robots.txt protocol [278].

²I have added the line breaks for readability.

³URI-M http://web.archive.org/web/20120118165255/http://upload.wikimedia.org/wikipedia/commons/9/98/WP_SOPA_Splash_Full.jpg


```

1 mw.centralNotice.insertBanner(
2 {"bannerName":"blackout","bannerHtml":
3 \u003Cstyle\u003E
4 #mw-sopaOverlay {
5     /* Opera Mini doesn't like position
6     absolute */
7     /* iOS doesn't like position fixed */
8     top: 0;
9     left: 0;
10    width: 100%;
11    height: 100%;
12    z-index: 500;
13    color: #dedede;
14    background: black url(//upload.wikimedia.org/
15        wikipedia/commons/9/
16        98/WP_SOPA_Splash_Full.jpg)
17    ...
18    id=\"mw-sopaHeadline\" \u003EImagine a World
19        \u003Cbr / \u003EWithout          Free Knowledge
20    \u003C/div\u003E');
21    \tvar intro = $(' \u003Cdiv id=\"mw-sopaText\"
22        \u003E \u003Cp \u003EFor over a decade, we
23        have spent millions of hours building the
24        largest encyclopedia in human history.
25        Right now, the U.S.          Congress is considering
26        legislation that could fatally damage the
27        free and open Internet. For 24 hours, to
28        raise awareness, we are blacking out Wikipedia.
29    ...

```

FIG. 36: The JSON object that identifies the blackout page background image.

```

1  var scriptUrl=mw.config.get
2      ('wgCentralBannerDispatcher')+ '?' +
3      $.param(bannerDispatchQuery);
4
5  $.ajax(
6  {
7      url:scriptUrl,
8      dataType:'script',
9      cache:true
10 });
11
12 insertBanner:function(bannerJson)
13 {
14     window.insertBanner(bannerJson);
15 }
16

```

FIG. 37: The JavaScript that constructs the banner URI and uses Ajax to request the blackout banner.

We have previously demonstrated that JavaScript in mementos can lead to leakage [54]. We also showed that JavaScript can also make access to mementos from the live Web easier but ironically makes archiving those resources harder (Section 4.2), an example of how technologies intended to improve a user’s browsing experience can actually be more difficult, if not impossible, to archive.

In addition to our investigation, Jackson of the UK Web Archive has outlined difficulties and potential approaches for archiving JavaScript-focused Web applications using the Wikipedia SOPA protest as an example [134]. In his analysis of the difficulties automatic crawlers have when capturing this protest, Jackson discusses the need to archive Web applications and discusses the need to use PhantomJS (or a similar technology) to uncover run-time URIs to be added to the crawl frontier.

Wikipedia’s and Craigslist’s SOPA protests are prime examples of an historical event that has not been properly archived because the archival targets have deferred representations. To prevent further historical loss, we recommend that automatic web crawlers and specialist archives use a two-tiered crawling approach to more accurately archive deferred representations.

4.2 JAVASCRIPT AND MEMENTOS OF MEMENTOS

On February 2, 2015, Voorburg announced the JavaScript utility `robustify.js` [47, 295, 296]. The `robustify.js` code, when embedded in the HTML of a Web page, helps address the challenge of link rot by detecting when a clicked link will return an HTTP 404 and uses the Memento Time Travel Service⁴ to discover mementos of the URI-R. `Robustify.js` assigns an onclick event to each anchor tag in the HTML. When the event occurs, `robustify.js` makes an Ajax call to a service to test the HTTP response code of the target URI. When an HTTP 404 response code is detected by `robustify.js`, it uses Ajax to make a call to a remote server, uses the Memento Time Travel Service to find mementos of the URI-R, and uses a JavaScript alert to let the user know that JavaScript will redirect the user to the memento.

Our studies show that JavaScript makes preservation more difficult [52], but `robustify.js` is a useful utility that is easily implemented to solve an important challenge (link rot). Along this thought process, we wanted to see how a tool like `robustify.js` would behave when archived.

We constructed two very simple test pages, both of which include links to Voorburg's missing page⁵.

1. `http://www.cs.odu.edu/~jbrunelle/wsd1/unrobustifyTest.html`
which does not use `robustify.js`
2. `http://www.cs.odu.edu/~jbrunelle/wsd1/robustifyTest.html`
which does use `robustify.js`

In `robustifyTest.html`, when the user clicks on the link to `http://www.dds.nl/~krantb/stellingen/`, an HTTP GET request is issued by `robustify.js` to an API that returns an existing memento of the page (Figure 38).

The resulting JSON is used by `robustify.js` to redirect the user to the memento

```
http://web.archive.org/web/19990830104212/http:
//www.dds.nl/~krantb/stellingen/
```

as expected.

⁴`http://timetravel.mementoweb.org/`

⁵`http://www.dds.nl/~krantb/stellingen/`

```
1 GET /services/statuscode.php?url=
2     http%3A%2F%2Fwww.dds.nl%2F~krantb
3     %2Fstellingen%2F HTTP/1.1
4 Host: digitopia.nl
5 Connection: keep-alive
6 Origin: http://www.cs.odu.edu
7 User-Agent: Mozilla/5.0 (iPhone; CPU
8     iPhone OS 8_0 like Mac OS X)
9     AppleWebKit/600.1.3 (KHTML, like Gecko)
10    Version/8.0 Mobile/12A4345d Safari/600.1.4
11 Accept: */*
12 Referer: http://www.cs.odu.edu/~jbrunelle/
13     wsdl/robustifyTest.html
14 Accept-Encoding: gzip, deflate, sdch
15 Accept-Language: en-US,en;q=0.8
16
17 HTTP/1.1 200 OK
18 Server: nginx/1.1.19
19 Date: Fri, 06 Feb 2015 21:47:51 GMT
20 Content-Type: application/json; charset=UTF-8
21 Transfer-Encoding: chunked
22 Connection: keep-alive
23 X-Powered-By: PHP/5.3.10-1ubuntu3.15
24 Access-Control-Allow-Origin: *
```

FIG. 38: HTTP headers of the robustify JavaScript.

Given this success, we wanted to understand how our test pages would behave in the archives. We also included a link to the stellingen memento in our test page before archiving to understand how a URI-M would behave in the archives. We used the Internet Archive’s Save Page Now feature to create the mementos at URI-Ms

```
http://web.archive.org/web/20150206214019/http://www.cs.odu.edu/~jbrunelle/wsd1/robustifyTest.html
```

and

```
http://web.archive.org/web/20150206215522/http://www.cs.odu.edu/~jbrunelle/wsd1/unrobustifyTest.html
```

The Internet Archive re-wrote the embedded links to be relative to the archive in the memento during replay, converting

```
http://www.dds.nl/~krantb/stellingen/
```

to

```
http://web.archive.org/web/20150206214019/http://www.dds.nl/~krantb/stellingen/
```

Upon further investigation, we noticed that robustify.js does not issue onclick events to anchor tags linking to pages within the same TLD as the host page. An onclick event is not assigned to any embedded anchor tags because all of the links point to within the Internet Archive, the host TLD. Due to this design decision, robustify.js is never invoked when within the archive⁶.

When clicking on the URI-M, the 2015-02-06 memento does not exist, so the Internet Archive redirects the user to the closest memento

⁶This was fixed by Voorburg after our initial work (and released on the robustify GitHub, <https://github.com/renevoorburg/robustify.js?files=1>, but the original problem illustrates a potential issue with JavaScript in the archives and unintended side effects, even in tools meant to aid in preservation.

```
http://web.archive.org/web/19990830104212/http:
//www.dds.nl/~krantb/stellingen/
```

The user, when clicking the link, is redirected to the 1999 memento by the Wayback Machine, because the Wayback Machine understands how to redirect the user from the 2015 URI-M for a memento that does not exist to the 1999 URI-M for a memento that does exist. If the Internet Archive had no memento for

```
http://www.dds.nl/~krantb/stellingen/
```

the user would simply receive a 404 and not have the benefit of robustify.js using the Memento Time Travel service to search additional archives.

The robustify.js file is archived at URI-M

```
http://web.archive.org/web/20150206214020js_/http:
//digitopia.nl/js/robustify-min.js
```

but its embedded URI-Rs are re-written by the Internet Archive. The original, live Web JavaScript has URI templates embedded in the code that are completed at run time by inserting the “yyymmddhhmmss” and “url” variable strings into the URI-R:

```
archive:"http://timetravel.mementoweb.org/memento/{yyymmddhhmmss}/
{url}",statusservice:"http://digitopia.nl/services/statuscode.php
?url={url}"
```

These templates are rewritten during playback to be relative to the Internet Archive:

```
archive:"/web/20150206214020/http://timetravel.mementoweb.org/memento/
{yyymmddhhmmss}/{url}",statusservice:"/web/20150206214020/
http://digitopia.nl/services/statuscode.php?url={url}"
```

Because the robustify.js is modified during archiving, we wanted to understand the impact of including the URI-M of robustify.js

```
http://web.archive.org/web/20150206214020js_/http:
//digitopia.nl/js/robustify-min.js
```

in our test page

```
http://www.cs.odu.edu/~jbrunelle/wsd1/test-r.html
```

In this scenario, the JavaScript attempts to execute when the user clicks on the page's links, but the re-written URIs point to

```
/web/20150206214020/http://digitopia.nl/services/statuscode.php?
url=http%3A%2F%2Fwww.dds.nl%2F~krantb%2Fstellingeng%2F
```

since test-r.html exists on www.cs.odu.edu, the links are relative to www.cs.odu.edu instead of archive.org.

Instead of issuing an HTTP GET for

```
http://digitopia.nl/services/statuscode.php?url=http%3A%2F%2Fwww.dds.
nl%2F~krantb%2Fstellingeng%2F
```

robustify.js issues an HTTP GET for

```
http://www.cs.odu.edu/web/20150206214020/http:
//digitopia.nl/services/statuscode.php?url=http%3A%2F%2Fwww.dds.nl%
2F~krantb%2Fstellingeng%2F
```

which returns an HTTP 404 when dereferenced. The robustify.js script does not handle the HTTP 404 response when looking for its service, and throws an exception in this scenario. Note that the memento that references the URI-M of robustify.js does not throw an exception because the robustify.js script does not make a call to digitopia.nl/services/.

In our test mementos, the Internet Archive also re-writes the URI-M

```
http://web.archive.org/web/19990830104212/http:
//www.dds.nl/~krantb/stellingen/
```

to

`http://web.archive.org/web/20150206214019/http://web.archive.org/web/19990830104212/http://www.dds.nl/~krantb/stellingen/`.

This memento of a memento (Figure 39) does not exist. Clicking on the apparent memento of a memento link causes the user to be told by the Internet Archive that the page is available to be archived.

We also created an Archive.is memento of our `robustifyTest.html` page at URI-M

`https://archive.is/19j30`

In this memento, the functionality of the `robustify` script is removed because Archive.is strips out all embedded JavaScript in mementos, redirecting the user to

`http://www.dds.nl/~krantb/stellingen/`

which results in an HTTP 404 response from the live Web. The link to the Internet Archive memento is re-written to

`https://archive.is/o/19j30/http://www.dds.nl/~krantb/stellingen/`

which results in a redirect (via a refresh) to

`http://www.dds.nl/~krantb/stellingen/`

which results in an HTTP 404 response from the live Web, just as before. Archive.is uses this redirect approach as standard operating procedure. However, Archive.is re-writes all links to URI-Ms back to their respective URI-Rs.

This is a different path to a broken URI-M than the Internet Archive takes, but results in a broken URI-M, nonetheless. Note that Archive.is simply removes the `robustify.js` file from the memento, not only removing the functionality, but also removing any trace that it was present in the original page.

INTERNET ARCHIVE
WayBackMachine

<http://web.archive.org/web/19990830104212/http://w>

[Latest](#)
[Show All](#)


Hrm.

Wayback Machine doesn't have that page archived.

This page is available on the web!

Help make the Wayback Machine more complete!
[Save this url in the Wayback Machine](#)

Want to search for all archived pages under
<http://web.archive.org/web/19990830104212/http://www.dds.nl/~krantb/>
 ?



The Wayback Machine is an initiative of the [Internet Archive](#), a 501(c)(3) non-profit, building a digital library of Internet sites and other cultural artifacts in digital form. Other [projects](#) include [Open Library](#) & [archive-it.org](#).

Your use of the Wayback Machine is subject to the Internet Archive's [Terms of Use](#).

FIG. 39: This memento of a memento does not exist, but references a URI-M as a live-Web URI-R.

In an odd turn of events, our investigation into whether a JavaScript tool would behave properly in the archives has also identified a problem with URI-Ms in the archives. If Web content authors continue to utilize JavaScript to re-write URI-Rs to URI-Ms in an effort to mitigate link rot or utilize tools to help discover mementos of defunct links, there is a potential that the archives that utilize the Wayback Machine for memento replay may see additional challenges of this nature arising. This demonstrates that even JavaScript tools that are designed to improve preservation can cause issues with memento replay.

4.3 ZOMBIES IN THE ARCHIVES

Leakage occurs when mementos include current, live Web content. We refer to mementos that allow the live Web to “leak” into archived pages as *zombies* [54, 209]. Zombies are caused by deferred representations in the archives. When leakage occurs, a memento that should only load embedded mementos from the archives instead loads a live resource from the Web. That is to say, these resources are expected to be archived (“dead”) but still reach into the current Web, just as cinematic zombies reach out to the living world from the grave.

A zombie’s reach into the live Web is caused by URIs that are not correctly rewritten back into an archive. JavaScript is often unaware of the memento or Web archive in which it exists, resulting in JavaScript constructing URIs that point to the live Web from within a memento. When a memento is accessed by a user, embedded JavaScript will run and may attempt to dereference a URI-R constructed at run time. The archive is not able to rewrite the JavaScript-created URI-R to a URI-M because it did not discover it at crawl time. This results in the memento not pulling from the archives but “reaching out” (zombie-style) from the archive to the live Web⁷. When a zombie pulls live resources into a memento, two possible outcomes occur. First, the memento may be incomplete if the live resource is not available (missing an embedded resource). Second, the memento may be incorrect if a live Web resource with a last-modified date later than a memento’s memento-datetime is loaded into a memento, causing a temporal violation. Ainsworth et al. refer to this as *prima facie violative* [6].

We examine two humorous examples of the juxtaposition of past and present

⁷This is prevented in the Wayback Machine by using proxy mode, but can still lead to missing embedded resources.

content in the archives. While our past work has focused on leakage in general (Chapter 5), these examples focus only on the leakage of advertisements into mementos. Advertisements capture cultural norms and are important for studies of popular culture and opinion. As such, they are important to archivists and – while potentially annoying to today’s Web users – are valuable to future researchers.

In our first example, we look at a memento of CNN.com (Figure 40). We can observe a memento from the Wayback Machine at URI-M <http://web.archive.org/web/20080916123132/http://www.cnn.com/> which includes embedded JavaScript that pulls advertisements from the then live Web. This memento from September 16, 2008, includes links to articles about the 2008 presidential race between McCain-Palin and Obama-Biden. However, we observed this memento (and took the screenshot provided in Figure 41) on September 28, 2012, during the 2012 presidential race between Romney-Ryan and Obama-Biden. The advertisement in Figure 40 is a zombie resource that promotes the 2012 presidential debate between Romney and Obama. This temporal violation seems to provide a prophetic look at the 2012 presidential candidates in a 2008 memento. The live CNN.com homepage representation from September 28, 2012 gives the same advertisement as the archived version (as seen in Figure 41).

In a second example, we examine a memento of the IMDB movie database site. We observed the July 28, 2011, memento of the IMDB homepage at <http://web.archive.org/web/20110728165802/http://www.imdb.com/> (Figure 42). This memento advertises the movie *Cowboys and Aliens*. This movie is set to start “tomorrow” according to the text in our observed July 28, 2011, memento. Additionally, we see the current feature movie is *Captain America*.

According to the IMDB site as observed live on September 28, 2012 (Figure 43), *Cowboys and Aliens* was released in 2011 (Figure 43(a)), and *Captain America* was released in 2011 (Figure 43(b)), in keeping with our observed memento. However, the ad included on the IMDB memento promotes the movie “*Won’t Back Down*.” According to IMDB, this movie was not released until 2012 (Figure 43(c)). Again, we can observe a memento with reference to present-day events through leakage.

We monitored the HTTP traffic to validate that these live ads are indeed leaking into our mementos. In our investigation of the HTTP requests made when loading the mementos, we found evidence of the reach into the live Web. We stored all HTTP headers generated when a user browses the memento into a text file for analysis.



FIG. 40: A temporal inconsistency is shown in the 2008 memento of CNN.com from the Wayback Machine at URI-M <http://web.archive.org/web/20080916123132/http://www.cnn.com/>.

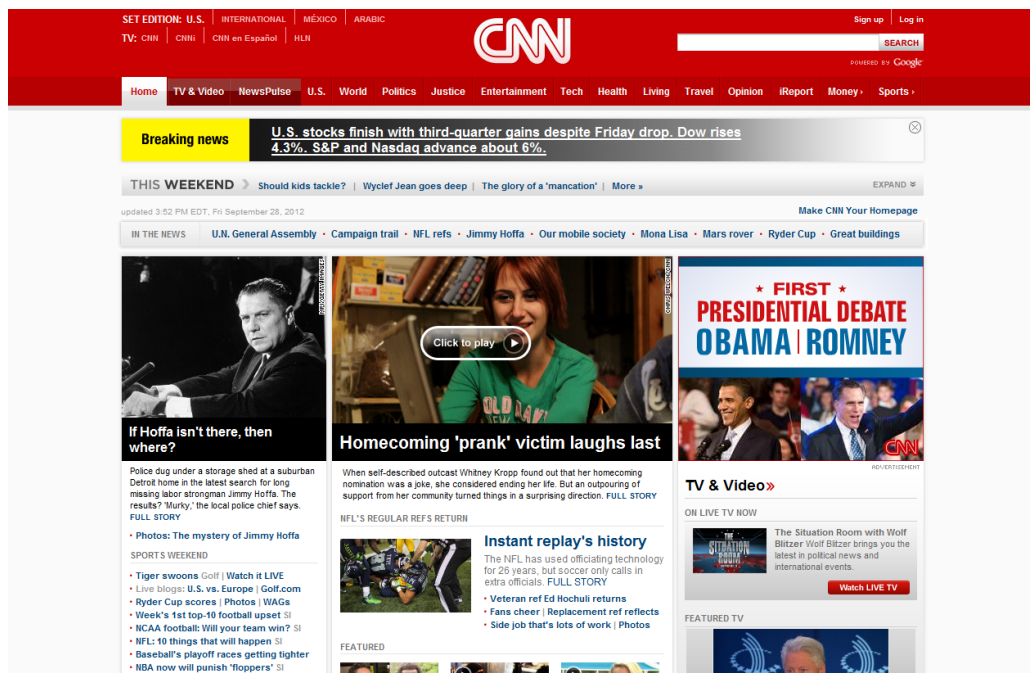


FIG. 41: A temporally correct snapshot of a live CNN.com site

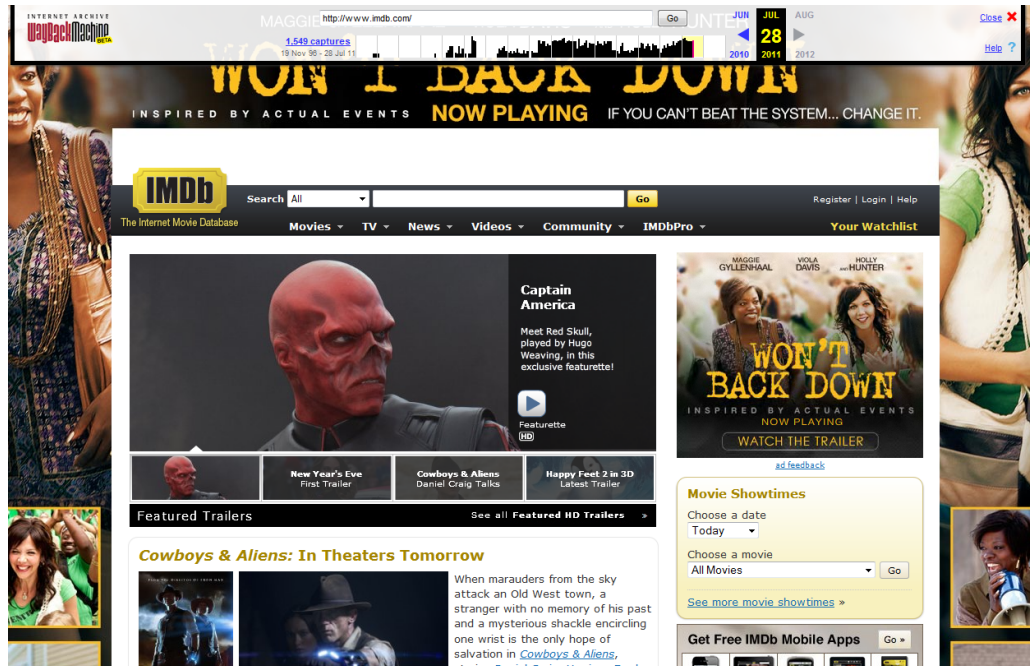
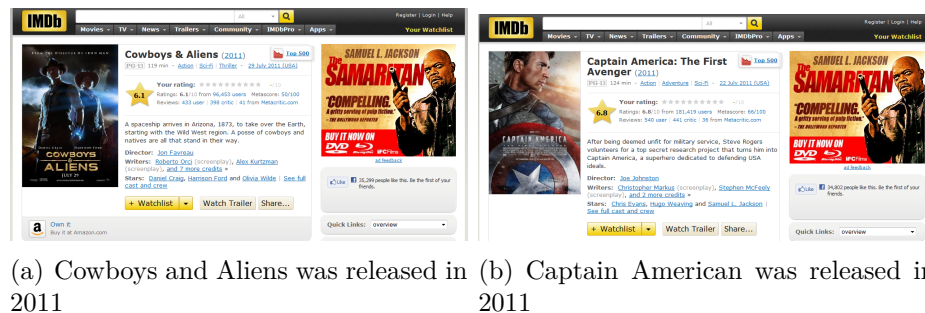


FIG. 42: A 2011 memento of IMDB.com from the Wayback Machine at URI-M <http://web.archive.org/web/20110728165802/http://www.imdb.com/>



(a) Cowboys and Aliens was released in 2011 (b) Captain American was released in 2011



(c) Won't Back Down is scheduled to be released in 2012

FIG. 43: Another example of temporal inconsistencies from leakage in IMDB.com shows that leakage occurs in the archives.

```

1 $ grep Host: headers.txt | grep -v archive.org
2 Host: ocsp.incommon.org
3 Host: ocsp.usertrust.com
4 Host: exchange.cs.odu.edu
5 Host: ad.doubleclick.net
6 Host: ad.doubleclick.net
7 Host: ad.doubleclick.net
8 Host: ia.media-imdb.com
9 Host: core.insightexpressai.com
10 Host: ad.doubleclick.net
11 Host: ia.media-imdb.com
12 Host: core.insightexpressai.com
13 Host: ad.doubleclick.net
14 Host: ad.doubleclick.net
15 Host: ia.media-imdb.com
16 Host: ad.doubleclick.net

```

FIG. 44: Live Web resources are requested when viewing a CNN.com memento from the Internet Archive.

Since the target memento is archived in the Internet Archive, the requests *should* be to other resources within the archive.org domain. However, we observe requests for live Web resources in Figure 44. These requests from archives into the live Web are initiated by embedded JavaScript (Figure 45).

The Internet Archive is not the only archive at which we find leakage. We found an example in a WebCite memento of CNN.com archive on 2012-09-09 (Figure 46 and at URI-M <http://webcitation.org/6AYT4UFVf>). The “Popular on Facebook” section of the page has activity from two of my “facebook friends” as shown in the 2012-10-01 screenshot in Figure 47. One friend shared the “10 questions for Obama to answer” page which was published on October 1, 2012 and is shown below. My “friend” should not have been able to share a page that has not yet been published (2012-09-09 occurs before 2012-10-01), demonstrating leakage in the WebCite memento of CNN.com.

Such occurrences of leakage and zombie resources are not uncommon in today’s archives. Current Web technologies such as JavaScript make an unchanging capture difficult in the modern Web. However, it is useful for us as Web users and Web scientists to understand that zombies do exist in our archives.

```

1 <iframe src="http://www.imdb.com/images/
2       SF99c7f777fc74f1d954417f99b985a4af/a/
3       ifb/doubleclick/expand.html#imdb2.consumer
4       .homepage/;
5       tile=5;
6       sz=1008x60,1008x66,7x1;
7       p=ns;
8       ct=com;
9       [PASEGMENTS] ;
10      u=[CLIENT_SIDE_ORD] ;
11      ord=[CLIENT_SIDE_ORD] ?"
12      ...
13      onload="ad_utils.on_ad_load(this)">
14 </iframe>

```

FIG. 45: Embedded JavaScript (i.e., the scripts in a frame) is responsible for the leakage.

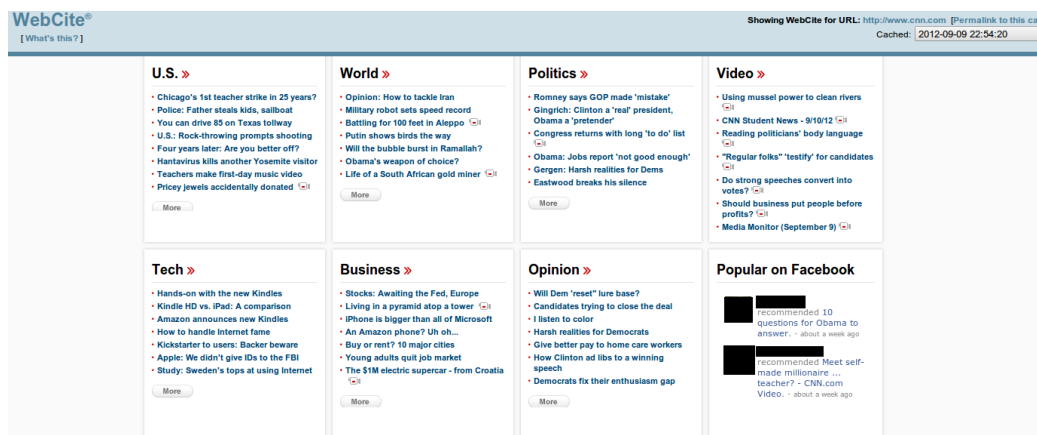


FIG. 46: WebCite memento of CNN.com with leakage.

10 questions for Obama to answer

By **David Frum**, CNN Contributor
updated 2:49 PM EDT, Mon October 1, 2012

6.8k

430

52

49

Recommend

Tweet

Share

+1

Print

Email

More sharing



President Barack Obama campaigns Sunday at a high school in Las Vegas.

STORY HIGHLIGHTS

- David Frum: President Obama has a record that should be questioned in the debates
- He says Obama should be asked about results of Afghan surge and "green-on-blue" attacks
- Frum: Ask about slow economic recovery and what level of

Editor's note: David Frum is a contributing editor at Newsweek and The Daily Beast and a CNN contributor. He is the author of seven books, including a new novel, "Patriots."

Washington (CNN) -- Mitt Romney has had a bad couple of weeks, really a bad month since the Republican National Convention in Tampa, Florida. The media spotlight has relentlessly focused on him. But there is an incumbent in the race, too, and an incumbent with a record that also reveals important disappointments, errors

With Cisco at the center, working together has never worked so well.

Learn More

CISCO

Part of complete coverage on

Opinion on the news

Mitt's foreign policy twilight zone
updated 10:36 AM EDT, Tue October 9, 2012

LZ Granderson says in Romney's foreign policy speech, he backpedals from his own stated positions as though he never held them and pivots away from reality in misrepresenting Obama's record.

For victims, 'justice' more than locking up abusers

FIG. 47: Live CNN.com resource containing the shared article.

4.4 MULTIPLE REPRESENTATIONS

Representations of resources can differ greatly depending on several factors [145]. For example, some sites have deferred representations, and others attempt to provide alternate representations by interpreting the **user-agent** portion of the HTTP GET headers through content negotiation, or provide representations based on session or user data (second-order expression based on context).

To test the behavior of these alternate representations in the archives, we archived URI-Rs using a mobile device's **user-agent** string and a desktop **user-agent** string and dereferenced the URI-Ms of the resulting mementos using both a desktop and mobile device (and, therefore, **user-agent** strings) to understand how the representations would be viewed by the devices. Additionally, we show that there is a potential for mobile-desktop memento collisions in the archives.

As an example, we study [145] the live front page resource identified by the URI-R <http://www.cnn.com/>. We ran a pair of limited crawls of the CNN.com front page with Heritrix 3.1 and then accessed the mementos captured by Heritrix via

the Wayback Machine with a desktop Mac and an Android phone. The first crawl captured the CNN.com front page and specified a desktop version of the Mozilla browser as the **user-agent** in the header string, as seen in Figure 48(a). We viewed the resulting memento in a local installation of the Wayback Machine (Figures 49(a) and 49(c)).

The second crawl captured the CNN.com front page and specified an iPhone version of the Mozilla browser as the **user-agent** string in the header, as shown in Figure 48(b). The resulting memento, as viewed in the Wayback Machine, is shown in Figures 49(b) and 49(d). The mobile and desktop representations differ in the archives, but a user of the Wayback Machine may not understand how these representations are generated since they are identified by the same URI-R. Additionally, the linkage between the mobile and desktop version exists on the live Web (e.g., via content negotiation) but disappears when archived. Similar behavior may exist for user interactions or client-side events; descendants (discussed further in Section 7.6) are identified by the same URI-R and could, therefore, lead to a URI-M collision. *Descendants* are client-side states reached through user-interactions or other client-side event that change the state of the representation. In this example, we observe mobile versus desktop user-agents, but the same archival anomalies presented in this section occur with descendants.

These examples illustrate the potential for collisions within the archives [145]. Collisions may occur when the same URI-R is archived at the same datetime but have different – potentially deferred – representations. If the mobile and desktop versions of the page are crawled by Heritrix at the same datetime, the URI-Ms in the Wayback Machine would be indistinguishable. The live Web version of CNN.com is identified by `http://www.cnn.com/` regardless of the **user-agent** string and resulting representation. Today, it is impossible to predict whether `http://web.archive.org/{MEMENTODATETIME}/http://www.cnn.com/` will dereference to the mobile or the desktop version.

These collisions are not limited to mere changes in representation. **User-agent** changes can reveal different content in resources. For example, the representations for the same MySpace page (Figure 50) differ based on the **user-agent** string used in the HTTP access. The age and location of the user are not displayed in the desktop representation in Figure 50(a), but are revealed when accessed from a mobile device, such as in Figure 50(b).

```

WARC/1.0
WARC-Type: request
WARC-Target-URI: http://www.cnn.com/
WARC-Date: 2013-03-05T16:57:00Z
WARC-Concurrent-To: <urn:uuid:d338e6e5-6854-329b-adbb-de70a62e11f0>
WARC-Record-ID: <urn:uuid:a3e3e726-97bc-3cca-af5e-d83dd1827e05>
Content-Type: application/http; msgtype=request
Content-Length: 266

GET / HTTP/1.0
User-Agent: Mozilla/5.0 (compatible; heritrix/3.1.0 +http://yourdomain.com)
Connection: close
Referer: http://cnn.com/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Host: www.cnn.com
Cookie: CG=US:--:--; CG=US:--:-- |

```

(a) HTTP GET request from Heritrix with the desktop Mozilla **user-agent**.

```

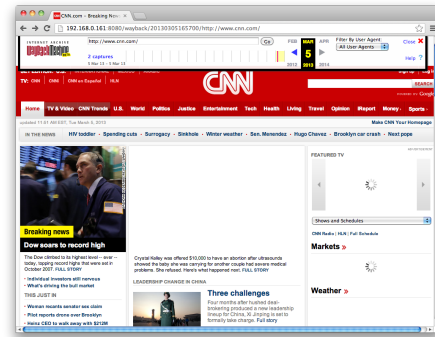
WARC/1.0
WARC-Type: request
WARC-Target-URI: http://www.cnn.com/
WARC-Date: 2013-03-05T16:38:08Z
WARC-Concurrent-To: <urn:uuid:cc7f75cc-fbaa-352a-8939-7cf5dd7792c7>
WARC-Record-ID: <urn:uuid:fcc902ba-b327-3f43-a5a8-a29861c4fa7e>
Content-Type: application/http; msgtype=request
Content-Length: 400

GET / HTTP/1.0
User-Agent: Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_0 like Mac OS X; en-us)
AppleWebKit/532.9 (KHTML, like Gecko) Version/4.0.5 Mobile/8A293 Safari/
6531.22.7 (compatible; heritrix/3.1.0 +http://yourdomain.com)
Connection: close
Referer: http://cnn.com/
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Host: www.cnn.com
Cookie: CG=US:--:--; CG=US:--:--

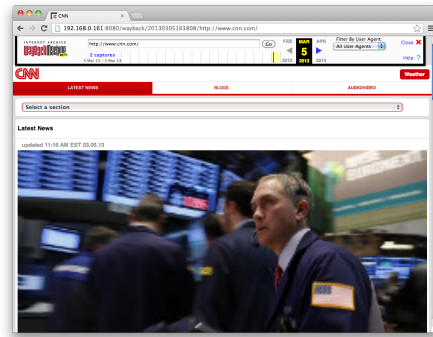
```

(b) HTTP GET request from Heritrix with the iPhone Mozilla **user-agent**.

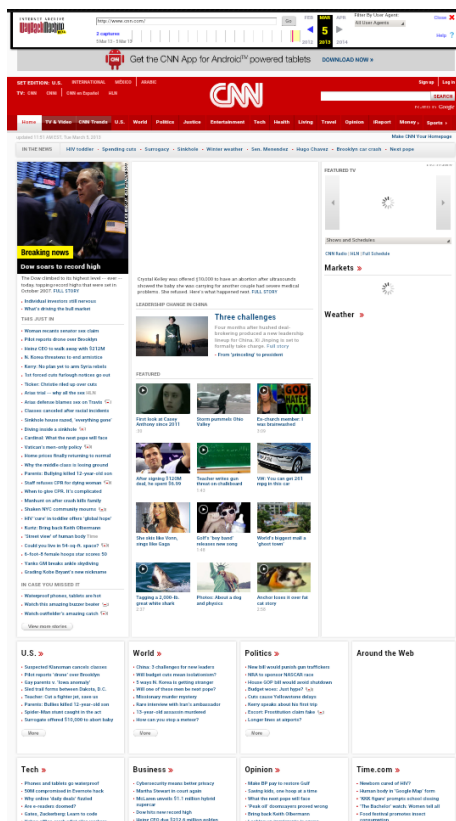
FIG. 48: WARC Record headers for the same resource accessed using different **user-agent** strings.



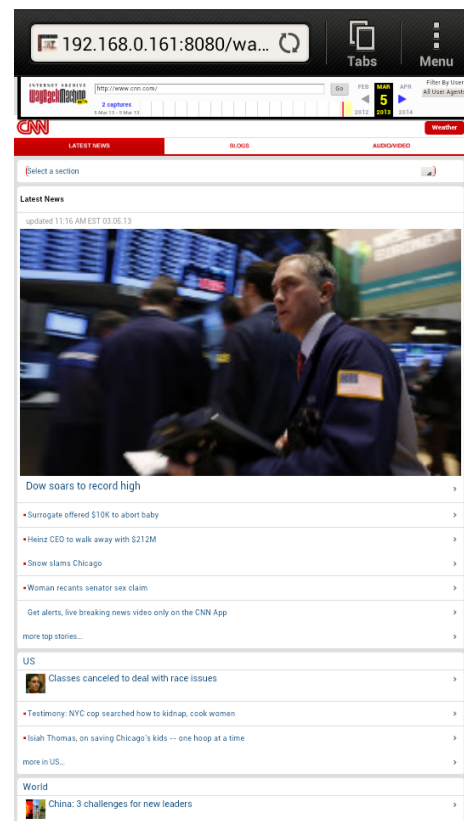
(a) The CNN.com memento when crawled by Heritrix with a desktop Mozilla user-agent accessed from a Mac.



(b) The CNN.com memento when crawled by Heritrix with an iPhone Mozilla user-agent accessed from a Mac.

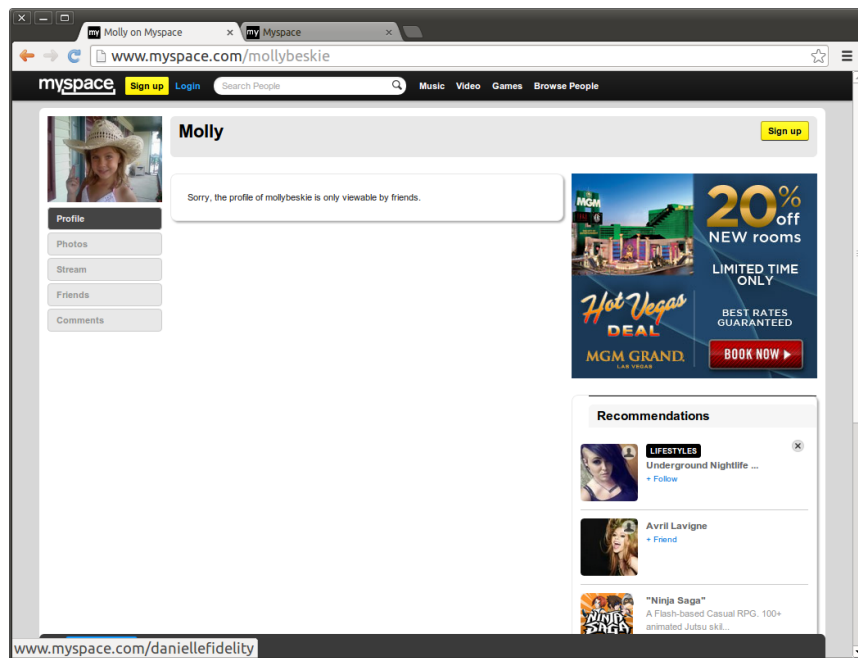


(c) The CNN.com memento when crawled by Heritrix with a desktop Mozilla user-agent accessed from an Android Phone.

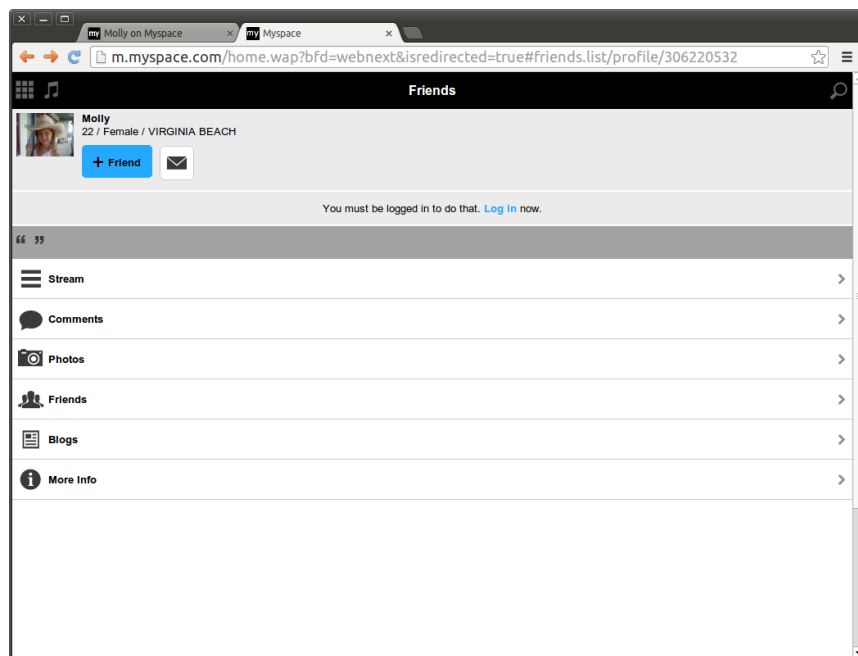


(d) The CNN.com memento when crawled by Heritrix with an iPhone Mozilla user-agent accessed from an Android phone.

FIG. 49: Mementos differ based on the parameters influencing the representations at crawl/capture time and the devices used to access the mementos.



(a) A desktop version of a private Myspace page.



(b) The same private Myspace page, viewed with a mobile browser.

FIG. 50: Resources return different representations based on the environment.

The granularity of URI-Ms makes differentiating between representations impossible with the current configurations in the archives. Additionally, users cannot reference mementos by **user-agent** or other environment variables because they are not part of the archives' indexing strategy. Temporal data alone is no longer sufficient to describe a memento; environmental variables must also be recorded and presented to the user, allowing the user to navigate between multiple dimensions or descendants of a representation (we propose using the JSON metadata in WARC_s proposed by the IIPC as presented in Chapter 7). Users can then decide whether browsing on the temporal dimension or the environmental parameters is more suitable for their goals.

In previous work [145], we proposed a set of examples that outline the increasing potential for collisions in the archives and a set of potential solutions. The results of the study demonstrated the increasing gap between the archives and live resources; live resources are increasingly impacted by the browsing paths, interactions, and environment variables, while the archives continue to capture content via singleton, HTTP-based, stateless transactions.

These collisions can potentially occur with deferred representations. If client-side code is executed, a descendant of the same resource will collide with other descendant in the archives since they are both identified by the same URI-R and potentially the same URI-M. To rectify this issue, WARC_s and WARC replay tools should adopt a method of identifying mementos with features in addition to archive datetime.

4.5 SUMMARY

In this section, we identify several examples in which JavaScript causes archival anomalies or short-comings. It is important to demonstrate why and how JavaScript creates additional archival challenges, and so we present observations of the results of JavaScript in the archives. These examples motivate the rest of this dissertation and are indicative of the importance of this research.

With a discussion of the challenges with archiving JavaScript in this chapter, we next begin to answer Research Question 1 by discussing the impact of JavaScript on the archives and measuring the increasing prevalence of JavaScript in the archives (Chapter 5). We also show that JavaScript-dependent embedded resources correlate to missing embedded resources in mementos, leading to lower quality mementos.

We address Research Question 2 in Chapter 6, discussing our work toward establishing an algorithm for automatically assessing the quality of mementos. In our work, we show that this algorithm provides a *damage* metric that is closer to human assessment of memento quality than the simple calculation of percent of embedded resources missing. We use this algorithm to evaluate the human assessed impact of JavaScript on memento quality showing that deferred representations have lower quality.

We discuss our work in answering Research Question 3 in Chapter 7. The most important contribution of this dissertation is the recommended framework for archiving the currently hard-to-archive deferred representations. Through understanding the extent and source of the challenges introduced by JavaScript and establishing that adoption is increasing, we show that a framework for archiving deferred representations is necessary to mitigate the impacts of JavaScript on the archives. First, we show that headless browsing clients and traditional crawlers have very different performance characteristics, with PhantomJS creating a 1.8 times larger frontier but running 12.12 times slower than Heritrix when comparing crawls of identical URI-Rs. After understanding the performance trade-off between PhantomJS and Heritrix, we discuss the number of client-side states (that we define as *descendants*) that may exist in a representation as a result of user interactions, and discuss the extent to which crawlers must interact with a representation to create the largest possible frontier. We show that there are two levels of descendants that lead to 15.6 times more URI-Rs in the crawl frontier, 70.9% of which are discovered via onclick events.

CHAPTER 5

MEASURING JAVASCRIPT IN THE ARCHIVES

The ease of archiving a Web page, called the *archivability* of the page, is impacted by the previously mentioned migration from Web pages to Web applications. We discuss how client-side technologies impact archivability in Section 2.2. Many culturally significant artifacts (such as Google Maps, shown in Figure 51 or the SOPA examples in Sections 1.2 and 4.1 [45]) are unable to be unarchived by today’s crawl methods. Even if resources are archived, they sometimes become *prima facie violative* because JavaScript may load content from the live Web (referred to as the live Web “leaking” into the archive, introduced in Section 4.3 [54] and discussed in more detail in Section 5.5) instead of only loading temporally appropriate content.

In this chapter, we answer Research Question 1: **To what extent does JavaScript impact archival tools?** We studied the impact of JavaScript on the archives [52] as a way to measure the extent of the problem created by the increasing adoption of JavaScript by Web pages. First, we studied live resources and mementos to provide insight into what makes a resource archivable. We used Heritrix, WebCite, and wget to capture a set of resources shared via Twitter and a set of Archive-It curated resources with the intent to determine how well each tool archives the resources¹. We compared these tools and collections by analyzing the resulting mementos and discuss the results to show that Twitter resources are less archivable than Archive-It resources, and Heritrix is the best archival tool of the three observed (note that we compare the performance of PhantomJS and Heritrix in Chapter 7).

We studied how archivability has changed over time as a result of JavaScript adoption [52]. We studied the mementos of the URI-Rs in our collections for completeness and investigated the cause of missing embedded resources. We showed that resources are becoming less archivable over time and memento incompleteness correlates to the adoption of JavaScript. In short, we provided insight into how well archival tools have performed in the past, and measured the role JavaScript has played in reducing archival coverage.

¹Dataset available at <https://github.com/jbrunelle/DataSets/blob/master/jsDataSet.txt>.

5.1 MOTIVATING EXAMPLE

Google Maps is an example of a resource with a deferred representation in which memento incompleteness is easy to observe. Figure 51(a) shows the live page, containing multiple interactive UI elements. The map in the middle is draggable, allowing the user to pan, click, and zoom. A memento from April 30, 2012, of this page in Figure 51(c) is missing UI elements and functionality (circled), and the interaction (e.g., panning and zooming) does not function. This is because the embedded resources are loaded by JavaScript as a result of user interaction which automatic crawlers such as Heritrix cannot perform and therefore do not discover. Figure 51(b) shows an archived (December 1, 2012) version that gives the façade of functionality when, in fact, resources on the live Web are leaking into the representation.

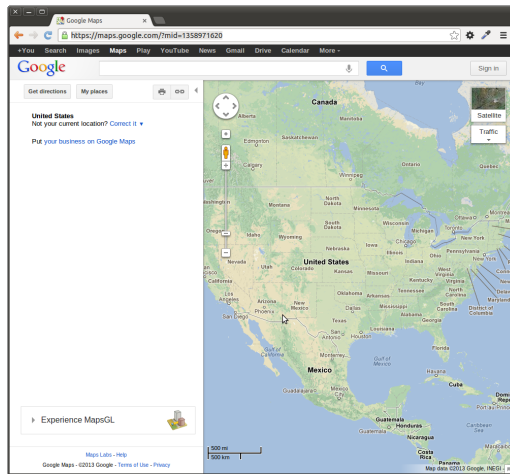
Figures 53 - 56 are examples of archival tools failing, at various levels of severity, to properly archive a resource because its representation uses JavaScript to load embedded resources. We archived versions of the resources in Figures 5.1, 53, and 55 with wget, WebCite², and Heritrix and compared the resulting archived representation to the live representation of the resource. For the wget capture, we show the replay of the memento stored on `localhost` both with and without access to the Internet. Viewing the resource without Internet connectivity ensures that only content that is locally archived is loaded into the resource (i.e., no leakage of the live Web into the memento). This simulates the absence of the live Web version of the resource.

Figure 5.1 shows a resource perfectly archived by the archival tools we examined. There is no leakage from the live Web and all requisite embedded resources are captured by each tool. There is no JavaScript in this page, and the resources loaded into the page all originate from the archive (i.e., no leakage exists). This is considered the best-case archival scenario.

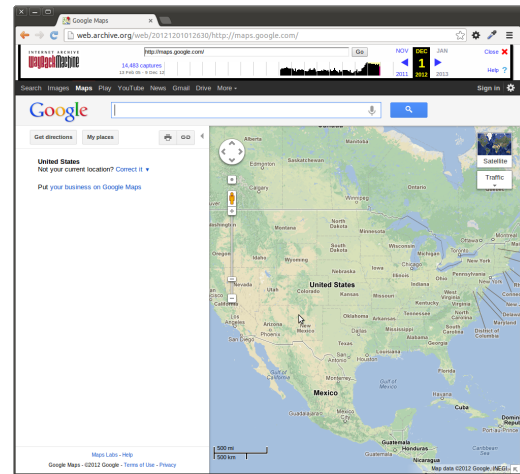
Figures 53 and 54 demonstrate a memento that is not perfectly archived but is “good enough.” All of the “important” content – the content contributing to human understanding of the page³ – was captured; however, there were slight representation and stylistic issues. As an example, we have added red circles in Figures 53 and 54 to highlight changes in content availability. The live representation (Figure 53)

²Archive.is was not available for testing at the time of this experiment, and is therefore omitted from this evaluation.

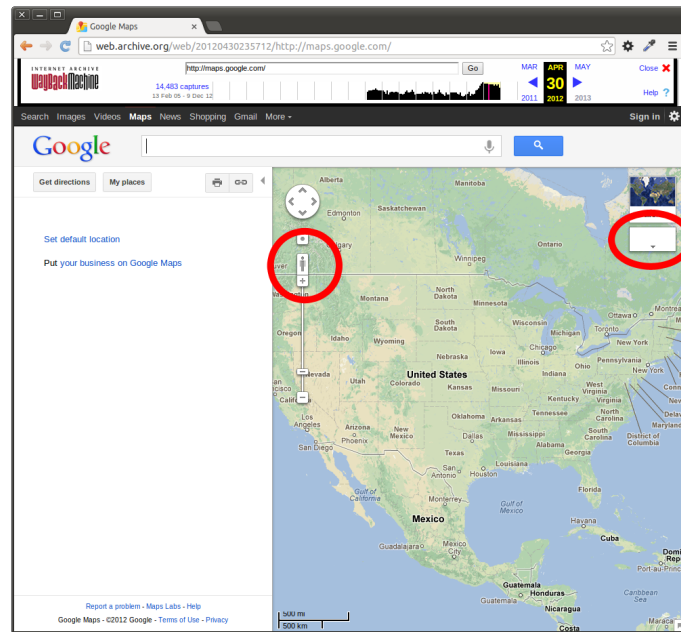
³We explore human understanding of Web page quality in Chapter 6.



(a) Live version



(b) Archived Dec. 1, 2012



(c) Archived Apr. 30, 2012

FIG. 51: Google Maps as it exists live and as a memento.



STATE OF ALABAMA
The
**Alabama State Board
Of Pharmacy**

The practice of pharmacy and the management and operation of pharmacies are hereby declared to affect the public health, safety and welfare of the people of Alabama, and thereby subject to regulation and control in the public interest.

[\[State of Alabama Privacy Statement\]](#)

LATEST ANNOUNCEMENTS
(Place mouse cursor over scrolling window to pause)

ALBOP BOARD MEETING

ALABAMA STATE BOARD OF PHARMACY
111 Village Street, Hoover, AL 35242
WEDNESDAY January 16, 2013
~ ~ ~

To suggest topics for FUTURE Work Sessions, send an e-mail to mellenburg@albop.com

All suggestions for topics will be considered by the Board for a determination as to whether they will be discussed at a future work session. Please provide as much information as possible about your desired topic.

Site Revised: December 2012

FIG. 52: The live version of <http://www.albop.com/main.html> (from the Archive-It collection) is perfectly archived using Heritrix, wget, and WebCite.

has links and icons for sharing the resource over social media services, as well as a Facebook “Like” icon. The social media links disappear in the mementos (Figure 54), and the Facebook “Like” icon disappears in the `localhost` without access to the Internet (Figure 54(a)) and WebCite (Figure 54(c)) versions. This issue with the representation is caused by leakage from the live Web into our archive. Leakage is evidenced by the memento with Internet connectivity having the “Like” icon, while the memento without Internet connectivity does not, because the JavaScript that loads the icon requests it from the live Web. This means yesterday’s HTML page has today’s “Like” icon and is *prima facie violative* [7]. In this case there is likely no problem, but this temporal inconsistency could be unacceptable in other cases (e.g., legal scenarios). Additionally, the WebCite memento (Figure 54(c)) is missing a stylesheet, causing the background to be a dark red instead of the tan color of the other mementos and live version. This URI-R and its deferred representation is an example of leakage from the live Web and of the problems introduced by Ajax in a memento.

The example in Figures 55 and 56 is poorly archived by most of the tools we



FIG. 53: Live version of <http://www.desbarresmanor.com/daytouring/history.html> from the Twitter collection.



FIG. 54: As shown by these mementos, the site <http://www.desbarresmanor.com/daytouring/history.html> is mostly archived with minor content leakage.

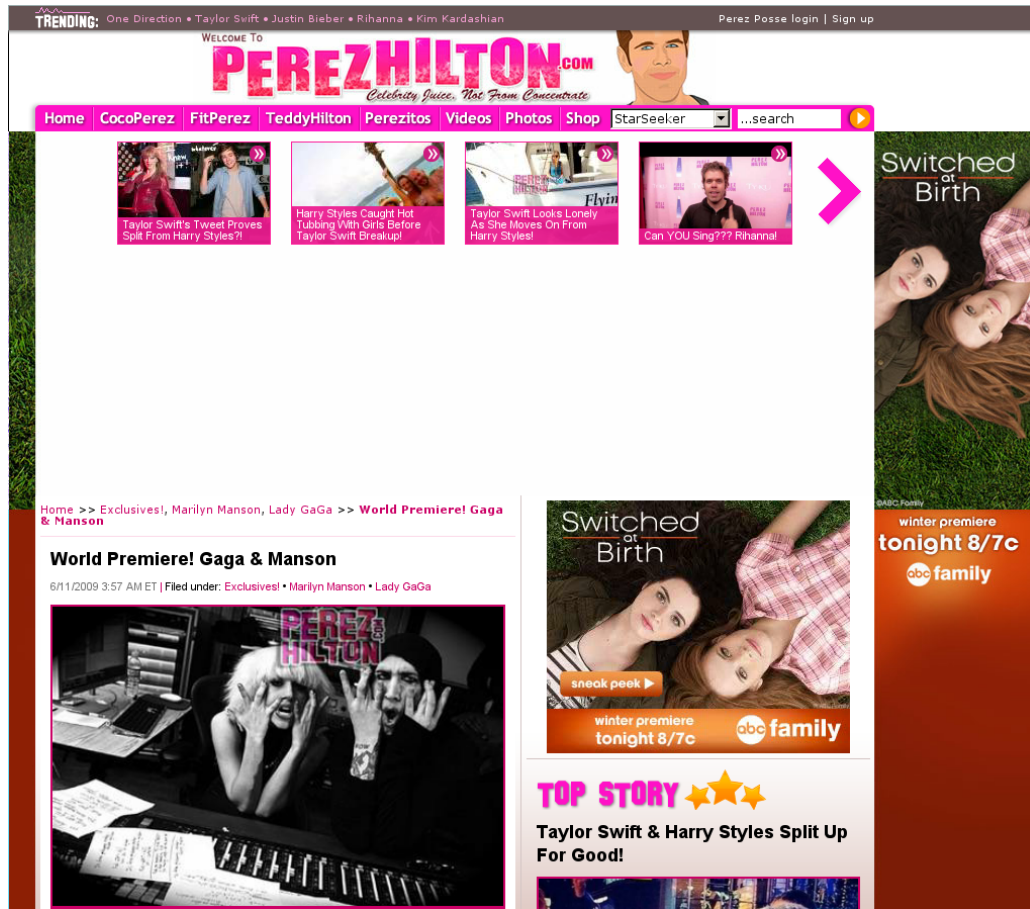


FIG. 55: Live version of <http://perezhilton.com/2009-06-11-world-premiere-gaga-manson> from the Twitter collection.

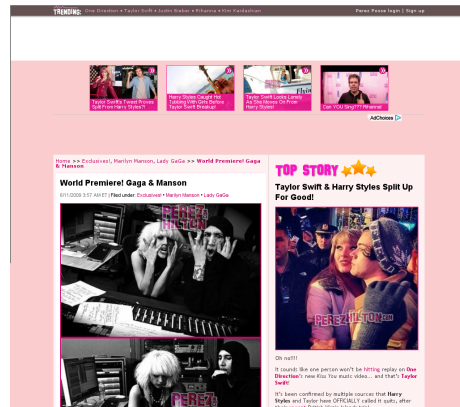
used. The reader can see that the live and the `localhost` version with access to the Internet (Figure 56(a)) are nearly identical, with the only difference being a missing banner at the top of the page. When observing the `localhost` version without access to the Internet (Figure 56(b)), there are many more missing embedded resources. The missing content highlights the fact that a large number of files are not actually archived and cannot be loaded without the live Web leaking into the memento because the archival tools cannot find them during the archival process. The WebCite (Figure 56(c)) capture seems to be loaded properly. However, there is a significant amount of leakage including many of the JavaScript files that load content and several of the image files being imported into this page. The Wayback memento (Figure 56(d)) makes several requests for content that was not captured (as noted by the “Resource not in archive” messages), and the content on the page, particularly the deferred content, was corrupted during the archival process and displays as code. The resource consists almost entirely of JavaScript that loads content into the page at run-time, resulting in the reliance on leakage to load the requisite resources.

5.2 EXPERIMENT DESIGN

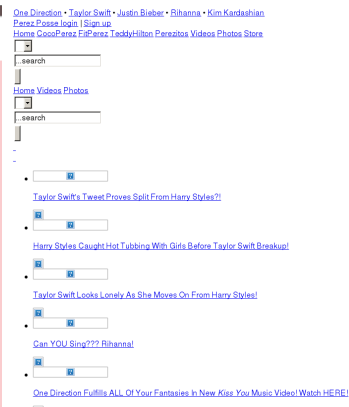
Given the aforementioned archival challenges, this experiment’s first goal is to observe how well current Web capture and archival tools can archive content and to identify characteristics of resources that are difficult to archive (i.e., have low archivability). Specifically, we study the completeness of the mementos captured by these tools and the impact that JavaScript has on archivability. We studied two sets of URI-Rs with respect to archivability; the first is a set of Bitlys (shortened URI-Rs shared in tweets) taken from Twitter, and the second is a set of human-curated URI-Rs from Archive-It.

We captured each set of resources with a set of archival tools, observed in their archived state, and each of the mementos is compared to the live *gold standard* version for completeness and to determine archivability. The Web page in its live, native environment is considered the best version possible or the best an archival tool can expect to achieve; if an archival tool replicates the live environment, it has perfectly captured and archived that resource. The first goal of our experiment is to compare each of the tools to one another, and each of the datasets are compared and contrasted.

The second goal of our experiment is to study the evolution of archivability over



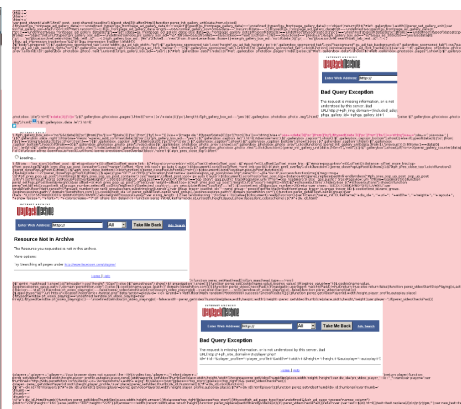
(a) Localhost version captured with wget, with access to the Internet.



(b) Localhost version captured with wget, without access to the Internet.



(c) WebCite version.



(d) Wayback version.

FIG. 56: As shown by these mementos, the site <http://perezhilton.com/2009-06-11-world-premiere-gaga-manson> is extremely difficult to archive.

time to determine the past and current trends as it relates to JavaScript’s adoption. We observed the effects of JavaScript on archivability over time by loading mementos from the Internet Archive with PhantomJS and recording the HTTP response codes of the embedded resources. We determined whether the mementos were loaded from HTML or JavaScript. From this, we analyze, in depth, the nature of the memento and its archivability over time.

5.2.1 DATASETS

This experiment utilizes two different datasets presented for comparison. The first dataset, the *Twitter* set, consists of Bitly URIs shared over Twitter. Shortened URIs are popular among social network services and the resources to which they redirect vary in expected life span [16]. The second dataset, the *Archive-It* set, was sampled from Archive-It collections. Archive-It collections are created and curated by human archivists often corresponding to a certain event (e.g., National September 11 Memorial Museum) or a specific set of Web sites with a common theme (e.g., City of San Francisco). The Twitter and Archive-It sets are made up of fundamentally different resources, as shown in Tables 5 and 6; the differences are investigated in this section. There is no overlap between the two sets.

We discarded non-HTML representations (i.e., binary MIME-types such as JPEG and PDF) from both sets. The purpose of this study is to observe how well the embedded resources are converted from live to archived. Non-HTML representations do not contribute to this study and are assumed to be perfectly archived by all tools.

Twitter

We collected the Twitter URI-Rs through the Twitter Garden Hose⁴ in October, 2012. This set consists of 1,000 URI-Rs and represents resources that Twitter users thought to be important enough to share with others on social media but which have not necessarily been actively archived. With the non-HTML representations removed, the Twitter set has 901 URIs.

⁴<https://dev.twitter.com/docs/streaming-apis/streams/public>; this service is no longer available.

Archive-It

The Archive-It set consists of the entire set of URI-Rs belonging to the collections listed on the first page of collections at Archive-It.org⁵ as of October 2012. This resulted in 2,093 URI-Rs that represent a collection of previously archived and human-curated URIs. To make the datasets equal in size, we randomly sampled 1,000 URIs from the set of 2,093. As shown in Table 5, the Archive-It set has a lower proportion of non-HTML content than the Twitter set. With the non-HTML representations removed from the randomly sampled set of 1,000 this collection has 960 URIs.

TABLE 4: Example URIs

	URI De- limiter	Example URI
URI Fragment	#	<code>http://www.example.com/ index.html#fragment</code>
URI Parameter	?	<code>http://www.example.com/ index.php?key1=value1</code>
HashBang URI	#!	<code>http://www.example.com/ index.php#!state1</code>

Collection Differences

In addition to the different collection sources, the datasets also differ in the server-side and client-side values passed to the resource via URI as parameters and fragments, respectively. URI fragments (first row of Table 4) identify a point or portion of content within the representation and are processed by the client (after the HTTP response), rather than the server when dereferencing the URI. Instead, they are an offset applied to the representation returned by the server and depend on file type, such as a tag within the HTML or a specific second in an audio file. Server-side parameters (second row of Table 4) identify values for use in constructing the representation (used on the server before the HTTP response) using key-value pairs and are used by the server when dereferencing the URI.

The Twitter set has more URIs containing fragments than the Archive-It set. Additionally, there are far more URIs with parameters in the Twitter set. Our complexity measure (to be defined in Section 5.4.1) has $\overline{UC}_{Twitter}=1.76$ and $UC_{Twitter_S}=0.312$,

⁵<http://www.archive-it.org/explore/?show=Collections>

meaning there are nearly 2 URI parameters in the Twitter data set for each URI, while the complexity of Archive-It URI-Rs has $\overline{UC}_{Archive-It}=0.16$, $UC_{Archive-It_S}=0.174$ meaning they are mostly without parameters. Note that only 3 URIs from the Twitter data set had both parameters and fragment identifiers (0.3% of the collection). Only 2 URIs from the Archive-It data set had both parameters and fragment identifiers (0.2% of the collection).

Additionally, 93.1% of URIs in the Archive-It set are the websites top level URI-R, whereas (in an aesthetically pleasing coincidence) 93.1% of the Twitter set is several levels away – leaf pages or *deep links* – from each site’s respective top level URI-R. We explore URI depth further in Section 5.4. The Twitter set contains slightly more client-side content than the Archive-It set, as suggested by the features of each set’s URIs. Twitter URIs have more client-side parameters as observed by the 0.6% more fragments than the Archive-It set. These fragments specify client-side parameters that are presumably used for client-side processing. There are also 2 hash-bang URIs (third row in Table 4) in the Twitter set and 5 in the Archive-It set.

The HTML of each set is measured and compared using several different metrics, to be discussed in Section 5.4. Each set consists mostly of representations containing JavaScript (98.7% and 97.1% of the Twitter and Archive-It sets, respectively, contain JavaScript)⁶.

TABLE 5: Content Features of Each Collection

Collection	Statistical Breakdown of Content				
	PDF	Images	Other Content	HTML	HTML Content containing JavaScript
Twitter <i>n</i> =901	0.3%	1.3%	3.7%	84.8%	98.7%
Archive-It <i>n</i> =960	4.4%	0.6%	1.3%	93.7%	97.1%

Table 6 shows the occurrences of the .com, .edu., .org, and .gov TLDs, but does not elaborate on the “Other Domains”. The “Other Domains” include .tv or .im, as in these URIs:

`http://www.geekbrief.tv/best-ad-from-microsoft-ever`

⁶To identify a representation as containing JavaScript, we did a boolean search for `<script>` tags within the representations.

TABLE 6: URI Features of Each Collection

Collection	Statistical Breakdown of URIs							
	COM	EDU	ORG	GOV	Other Do- mains	With Frag- ments	With Pa- rame- ters	TLDs
Twitter	81.6%	0.3%	3.5%	0.2%	14.1%	1.4%	30.5%	6.9%
Archive-It	47.9%	7.7%	10.9%	12.7%	20.6%	0.8%	41.2%	93.1%

`http://tr.im/lPRA#user1244749838078`

Governmental Web sites comprise 57.6% (533 URI-Rs) of the Archive-It collection. Governmental URIs are not limited to the `.gov` domain (such as `http://www.ethics.alabama.gov`) but can include other suffixes (such as `http://ali.state.al.us/`). The Twitter collection only has three government-owned sites (0.3% of the collection). Governmental Websites are mandated to conform to Web accessibility standards for content, whereas commercial Websites do not have this restriction. The government and non-government sites perform nearly identically in our content measurements – differing only by a maximum of 6% across all metrics and a Chi-square test provided a $p=0.22$ showing no statistical significance across our measurements – so we have opted to measure the collections in their entirety instead of measuring only the government and non-government URIs independently.

5.3 ARCHIVING THE RESOURCES

We archived each URI from each dataset with a suite of tools, including submitting them to WebCite, and accessed the resulting mementos on `localhost` after being captured with `wget`, archived with Heritrix 3.0.1, and, to establish a baseline, viewed in its native *live Web* environment.

We detail the methods used to archive the resources in the dataset and view the resulting mementos. We measure leakage in the mementos (Section 5.5), which only occurs when replaying the mementos in a client, and not during the archival process. Because a primary goal of Research Question 1 is to establish the impact of JavaScript on mementos created by the tools, we classify the mementos according to the environment in which they are viewed. That is, we label the mementos created by

Heritrix as “Wayback” mementos because the mementos are viewed via the Wayback Machine. Similarly, we label the WebCite mementos (viewed in a client through the WebCite Web service), live (live Web resources viewed through a client), and `localhost` mementos (archived with `wget` and viewed through a client from the `localhost` server on which the resources are stored).

We loaded the Twitter and Archive-It live Web resources using PhantomJS to establish the baseline to which all our mementos were compared. Note that the tools we investigated do not employ PhantomJS or other headless browsing techniques during archiving.

Using PhantomJS, we downloaded each resulting representation in its native format (such as HTML) and generated a PNG screenshot of the representation. We also captured and logged the HTTP response codes generated during the dereference and viewing the representation of each URI. These logs establish a set of resources required to view the page and also to identify whether or not the tool was successful in capturing all necessary resources to create a complete memento. We compared the HTTP response logs from a memento to the response logs from the live Web to establish the degree of completeness of the memento.

We installed a local version of Heritrix 3.0.1 and used it to crawl each of the URIs in this study. We installed an instance of the Wayback Machine v1.4 on `localhost` so we could view the mementos in the resulting WARC files. Mementos should load embedded resources from the local instance of the Wayback Machine. In a scenario such as this, the proxy mode is normally run with the Wayback Machine to prevent leakage but was omitted from this experiment since we were monitoring header values for leakage at load time, and this Wayback instance was limited to a target crawl.

We used the `wget` application to archive resources from the live Web on the `localhost` machine. Our goal was to make sure the entire site was captured, complete with requisite resources and converting the links to be relative to the local directory. We used the following command to download the content:

```
wget -k -p -E -H -w 10 -o ./logs/$i.log $toGet
```

This command captures the HTTP headers in the `./logs/$i.log` file (where `$i` is the identifier for the log file), and captures the target site `$toGet`. The locally

captured content can then be viewed from `localhost` and measured against the native, live Web version of the resource.

5.4 RESOURCE METRICS

We measure resource complexity based on the HTML representation and the URI identifying the resource to determine if complexity correlates with archivability.

5.4.1 URI COMPLEXITY

We measured the complexity of a URI through the depth (URI's number of levels down – or distance – from the TLD), the number of client-side parameters, and the number of server-side parameters. The client- and server-side parameters are measured by Equation 1. We take the arithmetic mean of the sum of these measures to give a URI complexity measure (UC) as noted in Equation 2.

$$F = \max(|\text{client-side parameters}|, |\text{server-side parameters}|) \quad (1)$$

$$UC = \frac{|\text{Depth}| + F}{2} \quad (2)$$

For example, a URI such as

`http://www.tennessee.gov/ecd/`

from the Archive-It collection has a complexity of 0.5. The depth is 1 (`/ecd/`), and there are no server- or client-side parameters. Alternatively, a URI such as

`http://edition.cnn.com/2009/SPORT/football/06/11/ronaldo.real.madrid.manchester/index.html?eref=edition`

from the Twitter collection has a complexity of 3.5. The depth is 6 (`/2009/SPORT/football/06/11/ronaldo.real.madrid.manchester/`), there is one server-side parameter but no client-side parameter ($\max(1, 0) = 1$), which totals to $7/2 = 3.5$. The URI

`http://www.tridentgum.com/#/tridentcares4kids/`

has a client-side parameter that does not have a corresponding anchor in the HTML ($\max(1, 0) = 1$) and has a complexity of $3/2 = 1.5$.

5.4.2 CONTENT COMPLEXITY

We measure content complexity in many different ways (e.g., number of HTML tags or presence of multimedia). This experiment focuses on JavaScript, since we have empirical evidence that JavaScript is the major contributor to missing embedded resources in mementos. Content complexity in this experiment is measured as the number of `<script>` tags loading JavaScript files or code segments, defined as CC in Equation 3. Intuition suggests that the more JavaScript embedded in a page, the more likely it will change the DOM or embedded resources, and the more difficult it is to archive. Our previous attempts at assessing content complexity focused on lines of JavaScript, but this metric was misleading (for example, due to minimized files or from including external libraries like jQuery), so we opted instead for a count of `<script>` tags.

$$CC = \sum \text{script tags} \in \text{HTML} \quad (3)$$

5.5 REQUESTS FROM HTML AND JAVASCRIPT

We compare the number of requests for resources referenced in the HTML DOM to the number of requests coming from run-time sources on the client, such as JavaScript. We compared the URI-Ms referenced in the HTML and CSS, and the requests observed by PhantomJS. Requests for URI-Ms referenced directly in the representation are classified as HTML-loaded resources, while those not referenced in the HTML have been loaded at run-time by JavaScript. We define JavaScript-loaded resources according to Equation 4 and use this method of measurement when referring to the number of resources loaded by JavaScript versus HTML.

$$\text{JS Resources} = \text{Resources Loaded} - \text{Resources in HTML and CSS} \quad (4)$$

5.6 RESOURCE SET ANALYSIS

The nature of the resources in each set is fundamentally different. We analyzed the complexities of the two datasets.

5.6.1 URI COMPLEXITY

The URI-Rs shared over Twitter tend to be *deep links*, with many layers of separation between the TLD and the shared URI-R. Additionally, the Twitter URIs tend to have client- or server-side parameters, which are assumed to identify a specific set of information or a special representation of the resource for the users' followers. Alternatively, the Archive-It collection, which has seed URIs chosen by human curators, tends to be made of more TLDs and have fewer parameters and fragments.

The \overline{UC} (Equation 2) of the Twitter collection is 0.377 with $UC_S = 0.311$. The Archive-It collection has a $\overline{UC}=0.166$ with $UC_S = 0.234$. The Archive-It collection is a lower \overline{UC} than the Twitter collection (as seen in Figure 57), supporting the theory that the human-curated Archive-It collection deals more with higher-level URI-Rs than the shared links of Twitter.

5.6.2 CONTENT COMPLEXITY

The Twitter set has a $\overline{CC}=4.78$ with $CC_S = 16.23$. The Archive-It set has $\overline{CC}=2.16$ with $CC_S = 6.87$. The Archive-It set has, on average, approximately half as many `<script>` tags as the Twitter set and a CC_S that is half that of the Twitter set (as shown in Figure 58).

We created a list of resources loaded from HTML, CSS ,and JavaScript and show the comparison of the number of JavaScript and HTML requested resources' archival success by set and archival tool in Figure 59. As expected, the CC measure correlates with the number of JavaScript requests to external resources. By taking the average across all environments, we found that the Twitter set resources load 16.3% of their embedded resources through JavaScript (presumably Ajax), whereas 18.7% of embedded resources are loaded via JavaScript in the Archive-It set. This is contrary to our hypothesis that increased CC will produce more resource requests from JavaScript – the Twitter set, which has more embedded JavaScript ($\overline{CC}=4.78$), makes fewer HTTP GET requests using JavaScript than the seemingly less complex

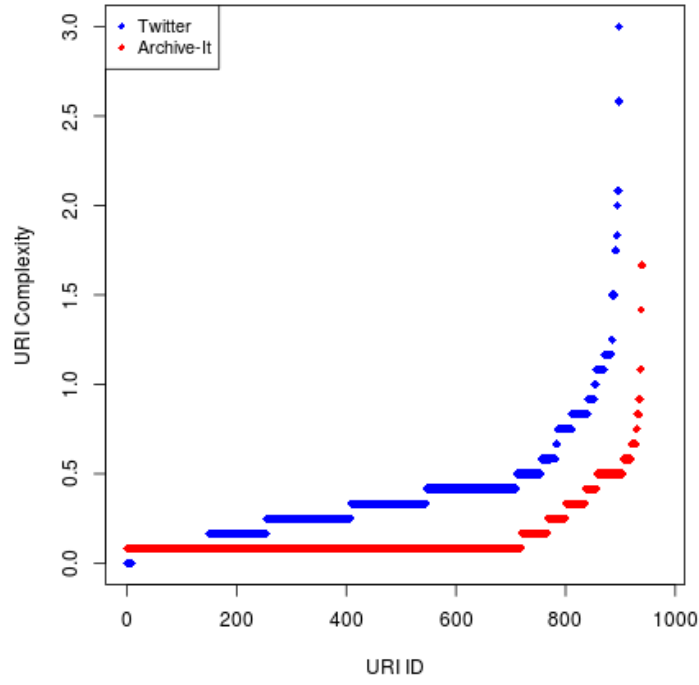


FIG. 57: URI Complexity measure (UC).

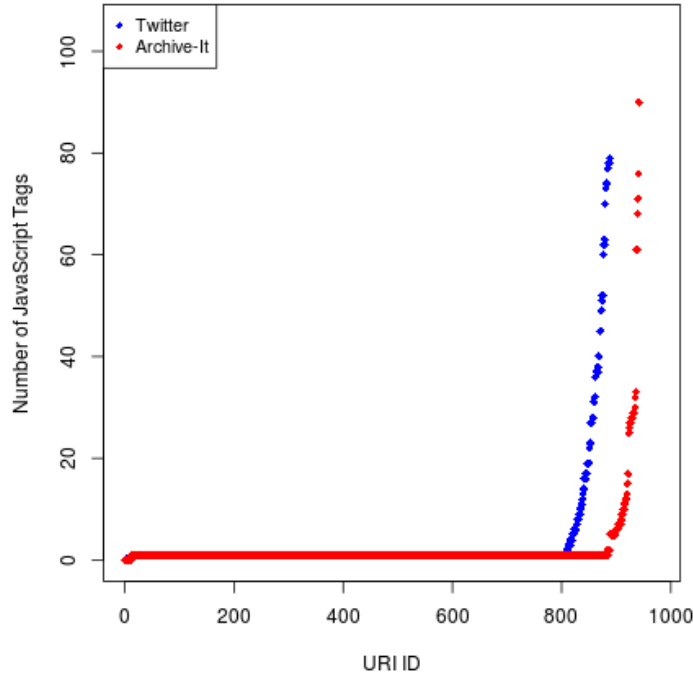


FIG. 58: CC of the HTML.

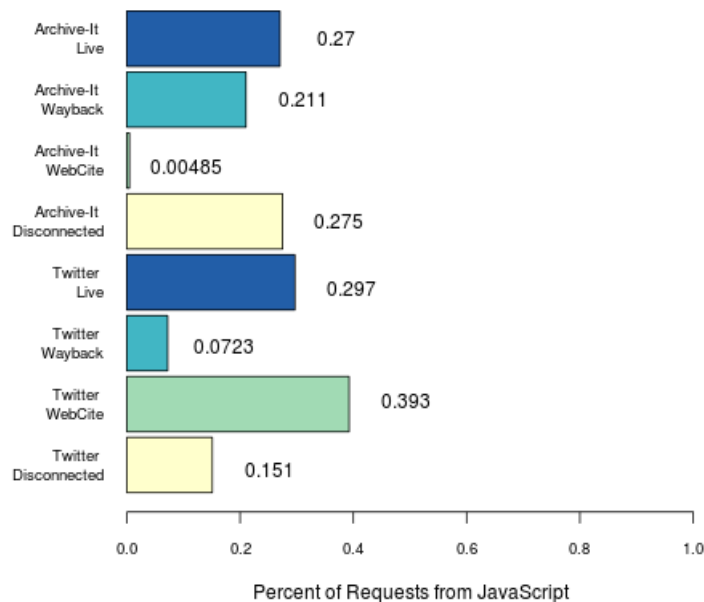


FIG. 59: Percentage of resource requests from JavaScript show patterns similar to *CC*.

Archive-It set ($\overline{CC}=2.16$). As discussed in Section 5.9, the nature of JavaScript-loaded resources reduces archivability.

5.7 ARCHIVING EXPERIMENT DISCUSSION

In this study, a perfectly archived resource has all of its embedded resources copied into the archive environment and is independent from the live environment. That is, the set of all embedded URI-Rs in the live Web matches the set of embedded URI-Ms. Failure to satisfy this capture will result in missing embedded resources (i.e., 400- and 500-classes of HTTP responses) or leakage (discussed in Section 4.3). As such, the archivability of a resource is measured by the number of embedded resources that archival tools can capture.

5.8 MISSING EMBEDDED RESOURCES

We analyzed the archival success rate of each archival tool and each collection with several metrics. Most important is the number of embedded resources missing from the mementos (i.e., the completeness). The average number of missing embedded resources is expressed as the percentage of HTTP non-200 responses returned when dereferencing the embedded URI (300- and 100-style redirects are omitted from this calculation). These averages for each collection and tool are shown in Figure 60. Each bar represents a collection and an environment. The bars representing the live environments establish the *gold standard*, or “best case,” archival goal. Live resources are not perfect – they sometimes include resources that are unavailable – and thus do not receive 200 responses for 100% of their requests. However, they do establish a ceiling of possible performance. The live resources in the Archive-It set receive an HTTP 200 response for 93.5% of all requests for embedded resources and live resources in the Twitter set receive an HTTP 200 response for 87.1% of all requests for embedded resources.

There is a clear difference between the archival tools used and the number of embedded resources successfully loaded in mementos – only 69.1% of the URI-Ms of embedded resources in mementos were successful.

We can demonstrate trends when analyzing the statistics in further depth. The resources in the Twitter collection, on average across all archived environments, produced 38.2% more HTTP non-200 response (i.e., 400- and 500- class HTTP responses) as mementos than the live state, meaning we were unable to archive 38.2% of the embedded resources in the Twitter set. The Archive-It set fares slightly better, with 37.7% more HTTP non-200 responses in their archived state. However, the WebCite results show the greatest difference between the Twitter and Archive-It sets (59.7% HTTP 200 responses vs. 42.8% 200 responses, respectively). If this difference is excluded, the Archive-It set performs better with only a reduction of 30.9% HTTP 200 responses, as compared to the Twitter collection’s 36.5% reduction.

5.9 LEAKAGE

As we have previously mentioned (Section 4.3) number of HTTP requests of mementos for embedded resources differ from that of the live environment. The WebCite, wget, and Wayback tools reduce the number of HTTP requests to different

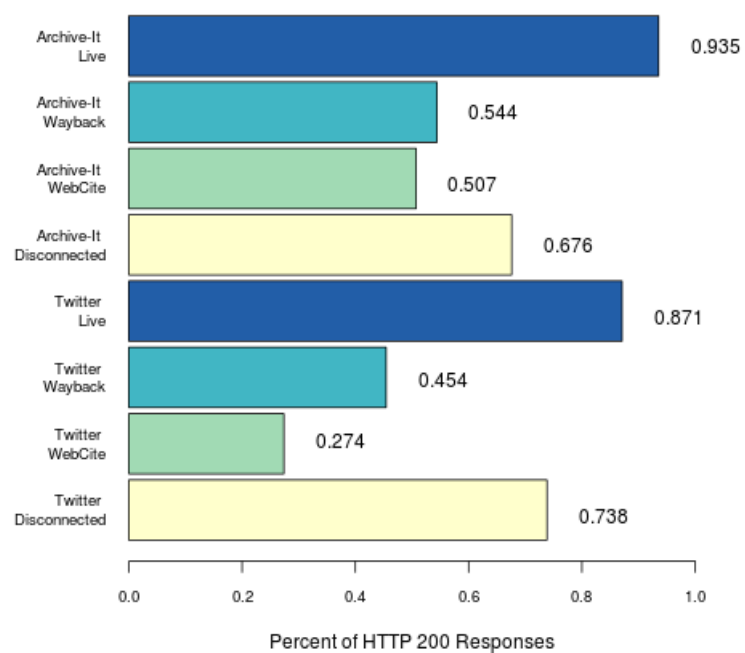


FIG. 60: The HTTP 200 response codes for embedded resources.

domains in order to change all of the requests to TLDs external to the archive (i.e., leakage) to requests for mementos within the archive. Resources referenced in HTML should be rewritten to reference a URI-M instead of a URI-R. Failure by the archive to rewrite the URI-R to a URI-M will result in a reference to an external domain when the memento is loaded (i.e., via leakage). Archival services cannot always predict at crawl time what embedded resources will be loaded (such as when loaded via JavaScript) or are unable to perform a rewrite of a URI-R constructed at run-time to a URI-M, resulting in leakage.

The proportion of requests to live TLDs (i.e., TLDs external from the archive) decreases when an archival tool captures the embedded resources and rewrites them local to the archive. Compared to the live Web, the number of requests to external domains (and therefore leakage) in the Twitter sets decreases by 43.2%, 22.5%, and 71.2% for the WebCite, wget, and Wayback tools, respectively. The Archive-It domain observes decreases of external requests of 32.3%, 1.5%, and 42.6% for the WebCite, wget, and Wayback tools, respectively. This shows that the archival tool affects the number of external HTML requests, with the wget tool doing the least to mitigate leakage and the Wayback Machine limiting leakage the most (note that this experiment did not use the Wayback Machine with proxy mode enabled). As shown by Figure 61, each archival tool impacts the target location of the requests for content the same way for each collection.

As in Section 5.4.2, we averaged across all environments shown in Figure 61 and found that 62.0% of Twitter resources loaded by HTML come from the archive's TLD, while 38.0% come from an external source. This is roughly similar to the Archive-It set, which loads 74.7% from the host TLD and 25.3% from external TLDs. This analysis of resource requests shows that JavaScript and HTML make requests for content in similar patterns, with the ratio of local versus external content following the same pattern. Additionally, the increased JavaScript requests to external content increases the opportunity for leakage and therefore decreases the archivability of a resource.

The number of JavaScript requests to archived vs. external (i.e., archived versus live) resources are similar to that of the HTML requests (Figure 59). The Twitter set gets 52.2% of its JavaScript-requested resources from external domains, while the Archive-It set gets 70.1% of its JavaScript-requested resources from external Web domains, suggesting that JavaScript is a source of leakage. As compared to the live

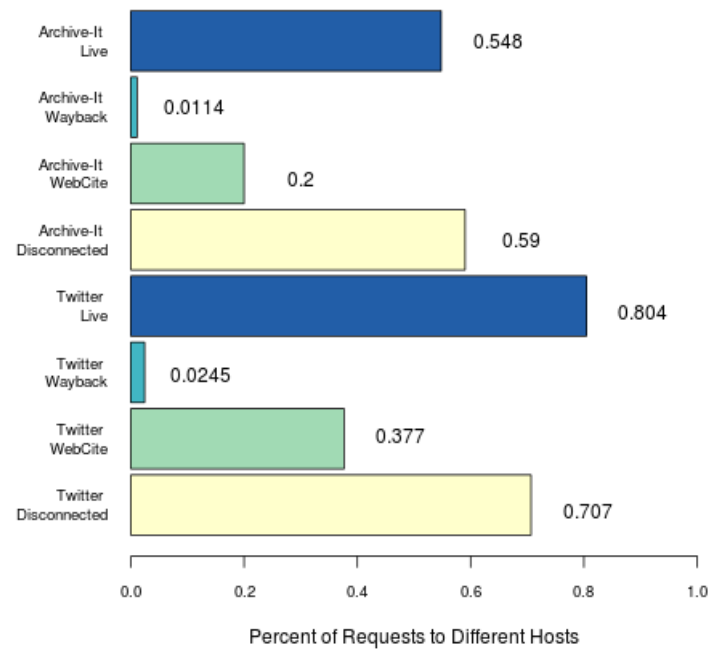


FIG. 61: Percentage of resource requests going to remote hosts from both HTML and JavaScript.

Web, we observed a reduction of external requests for resources in the Twitter set by 42.8%, 9.8%, and 78.0%, respectively. The Archive-It set's capture by the WebCite, wget, and Wayback tools reduced the number of JavaScript requests to external Web domains by 33.5%, 0.6%, and 48.0%, respectively. Again, this shows that the wget tool does little to mitigate leakage, and the Wayback Machine performs best by reducing requests to the live Web. Heritrix provides the most complete collection and performs the best in this analysis. Since Heritrix peeks into the JavaScript to extract URI-Rs, it can limit leakage by anticipating JavaScript requests from mementos.

The percentage of external requests is highest in mementos captured using WebCite with 14.6% of all resource requests resulting in leakage. This holds true regardless of dataset. The wget mementos are second highest with 14.2% of all resource requests resulting in leakage. The Wayback mementos experience the least leakage with only 7.1% of all resources requests resulting in leakage.

The Twitter set observes the most leakage (as seen in Figure 61). When the Twitter resources were captured with the wget, WebCite, and Heritrix tools, the leakage observed is (on average) 49.9%, 8.6%, and 1.3%, respectively. The Archive-It set only sees an average leakage rate of 44.1%, 4.7%, and 1.9% for each tool.

As shown, the Archive-It set is more easily archived, results in more complete mementos, and has fewer instances of leakage than the Twitter set. The Archive-It set has many resources that are perfectly archived. The Twitter set only had 4.2% of the resources that were perfectly archived by each tool, while 34.2% of the Archive-It set was perfectly archived by each tool. This shows that the Archive-It set is much more archivable than the Twitter set. The resources shared over Twitter were much more difficult to capture than the content human curators identify as historically or socially important. Twitter-shared resources are important enough to share with online acquaintances but cannot be effectively captured with current technologies. As shown, the Twitter set contained more JavaScript than the Archive-It set. As such, we conclude that resources relying on JavaScript to render a final representation are less archivable than representations made of purely HTML.

5.10 IMPACT OF ACCESSIBILITY

Many factors can impact archivability. As we have shown, current tools can capture Archive-It resources more completely than Twitter resources. The composition of the collections may impact this behavior. For example, the Archive-It set contains

more government-owned URI-Rs. Government URIs are often perfectly archived, with 85 of the 124 government URIs in our set being perfectly archived (68.5%). This shows that government URIs are more frequently perfectly archived than the rest of the mementos measured (the entire Archive-It collection was archived perfectly 34.2% of the time).

Section 508 [2] is a set of recommendations to be followed by government site owners and provides suggestions on how websites should be designed by content authors to be considered accessible. Also, the W3C through the Web Content Accessibility Guidelines (WCAG) gives concrete ways for developers to evaluate their pages to make them more accessible [64]. Hackett et al. detail the accessibility of archives on a pre-Ajax corpus (from 2002) and enumerate specific features that make a page accessible in the legal sense [118]. Much of this information is beyond the scope of this study (e.g., animated GIFs). Other efforts were made by Parmanto and Zheng following the document’s release on a metric to evaluate Web accessibility [224], but also fail to consider an asynchronous Web, which was less prevalent at the time.

All United States government sites are advised to comply with Section 508. These mandates guide content authors by limiting embedded programming languages and deferred representations. It stands to reason that pages adhering to Section 508 are easier for tools like Heritrix to capture in their entirety, which may influence the archivability of the Archive-It set.

In much of the same way that accessibility guidelines encourage content to be accessible to the end-user, complying with the guidelines also facilitates accessibility of the content (displayed by default or as the result of a JavaScript invocation) to the user-agent. Conventional Web browsers (e.g., Google Chrome, Mozilla Firefox) are capable of retaining this content in memory, even if not displayed. Contrary to modern browsers, archival crawlers like Heritrix are not equipped with the capability to execute client-side code to access, process, and capture deferred representations that use potentially inaccessible (according to WCAG) features.

Conventional Web browsers are usually the first to adopt new technologies that may introduce features that reduce archivability. Today, these features often are implemented using JavaScript and often disregard accessibility standards in favor of enticing users with more personalized and interactive representations. Because deferred representations are not accessible in their entirety to archival crawlers, requiring content on the Web to be more accessible would limit the impact of deferred

representations and newer browser features on the archives. Additionally, avoiding deferred representations will reduce the ability for Web resources to provide personalized and interactive content to users, presumably resulting in a less appealing representation.

5.11 MEASURING ARCHIVABILITY IN THE INTERNET ARCHIVE

We investigate the archivability of resources in our Twitter and Archive-It datasets over time by looking at the mementos in the Internet Archive. In effect, this is a black-box assessment of archiving by examining the output of the archival process over time. Because application technologies have increased in prevalence over time, we expect to see memento completeness decrease as time progresses. (We explore the value of measuring archive quality using memento completeness versus a more robust weighted metric in Chapter 6, but rely on the more simplistic measure of memento completeness for the purposes of this discussion.)

When considering the age and longevity of resources in the archives, it is useful to understand how the collections differ in age. We used the Carbon Dating service [251, 253, 215] to estimate the age of the URI-Rs in our collections. The results are presented in Figure 62. The carbon dating service was unable to reliably determine a creation date for 35.5% of our collection, so the sample size in this graph has been reduced to include only the URI-Rs with reliably determined creation dates. Note that the Archive-It set is evenly distributed over time, while the Twitter resources are younger. This is intuitively satisfying given the different purposes of the Archive-It and Twitter services.

To increase the heterogeneity of our dataset (mitigating the impact of accessibility standards and the age of URI-Rs), from here on we will combine the Twitter and Archive-It data sets.

To empirically observe the change in archivability over time of each of the URI-Rs in our collections, we acquired the TimeMap of each URI-R from the Internet Archive to produce output like Figure 63. We captured screen shots and HTTP requests for one memento per year of each URI-R in our collections in an effort to analyze the results of archival efforts performed in the past.

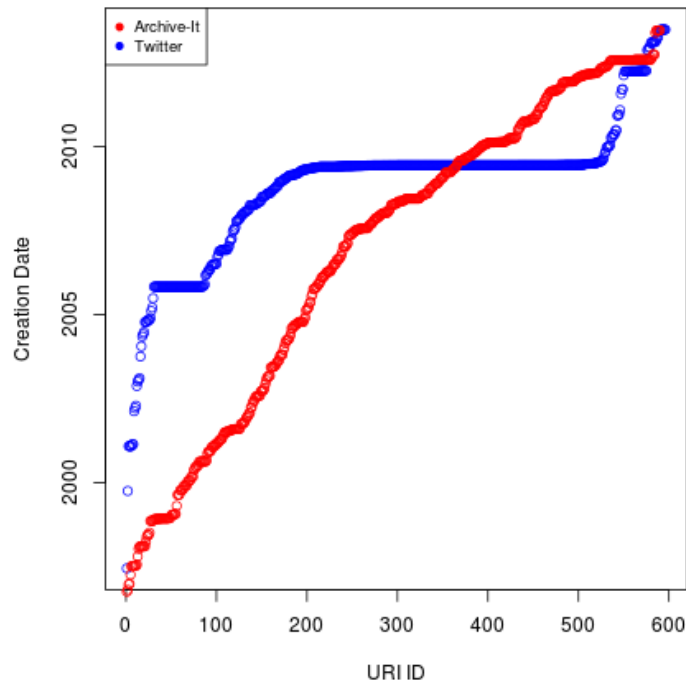


FIG. 62: The Twitter collection ($n=596$) is, on average, younger than the Archive-It collection ($n=590$).

The `http://www.doc.alabama.gov/` resource ($CC = 0.43$) appears to be perfectly archived throughout time in the Internet Archive. This holds true when accessing the mementos with and without JavaScript enabled. The mementos are visible in Figure 64. All mementos from 2007-2013 neither request a resource nor receive a non-200 HTTP response through JavaScript with the exception of a single memento with an archival date time of October 21, 2011, that requests and misses six resources via JavaScript.

The `http://www.cmt.com/` resource ($CC = 0.87$) varies in archivability (seen in Figure 65), with an increase in missing resources over time. The mementos from 2009-2013 (Figures 65(l)-65(p)) are missing almost all central article and image content; this is when CMT.com implemented content loading through jQuery and Ajax, resulting in an average of 9.7% of all URI-Ms requested to be missed because of JavaScript. CMT.com's addition of Ajax and associated technologies drastically increased the feature gap between the resource and the crawlers performing the archiving, while the `doc.alabama.gov` site limits its use of JavaScript and is more complete over time.


```

<http://www.doc.alabama.gov/>; rel="original",
<http://web.archive.org/web/timemap/link/
http://www.doc.alabama.gov/>;
  rel="self"; type="application/link-format";
  from="Mon, 08 Jan 2007 17:48:19 GMT";
  until="Sun, 28 Jul 2013 15:16:29 GMT",
<http://web.archive.org/web/http://www.doc.alabama.gov/>;
  rel="timegate",
<http://web.archive.org/web/20070108174819/
http://www.doc.alabama.gov/>;
  rel="first memento"; datetime="Mon, 08 Jan 2007 17:48:19 GMT",
<http://web.archive.org/web/20070113182156/
http://www.doc.alabama.gov/>;
  rel="memento"; datetime="Sat, 13 Jan 2007 18:21:56 GMT",
<http://web.archive.org/web/20070118175605/
http://www.doc.alabama.gov/>;
  rel="memento"; datetime="Thu, 18 Jan 2007 17:56:05 GMT",
<http://web.archive.org/web/20070123202638/
http://www.doc.alabama.gov/>;
  rel="memento"; datetime="Tue, 23 Jan 2007 20:26:38 GMT",
<http://web.archive.org/web/20070519200310/
http://www.doc.alabama.gov/>;
  rel="memento"; datetime="Sat, 19 May 2007 20:03:10 GMT",
<http://web.archive.org/web/20070617053244/
http://www.doc.alabama.gov/>;
  rel="memento"; datetime="Sun, 17 Jun 2007 05:32:44 GMT",
<http://web.archive.org/web/20070621140945/
http://www.doc.alabama.gov/>;
  rel="memento"; datetime="Thu, 21 Jun 2007 14:09:45 GMT",

```

FIG. 63: An abbreviated TimeMap for <http://www.doc.alabama.gov/>.

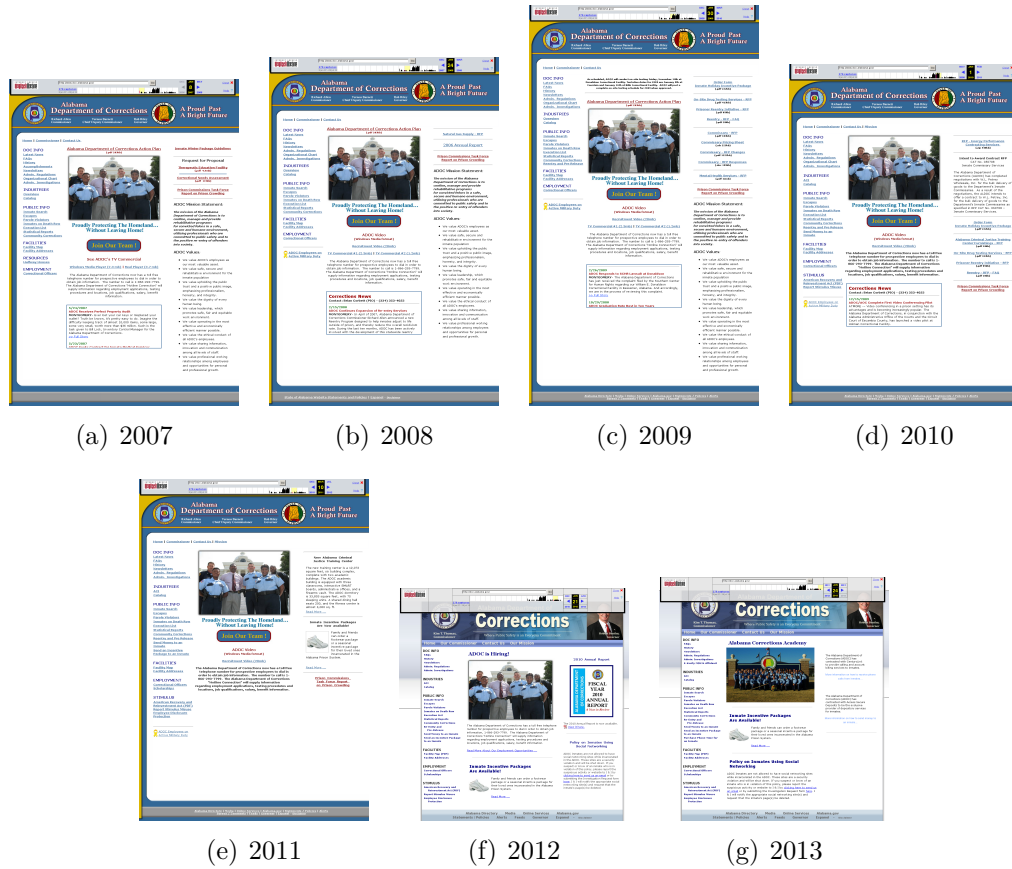


FIG. 64: The <http://www.doc.alabama.gov/> mementos are perfectly archived through time since they limit their reliance on JavaScript to load embedded resources.

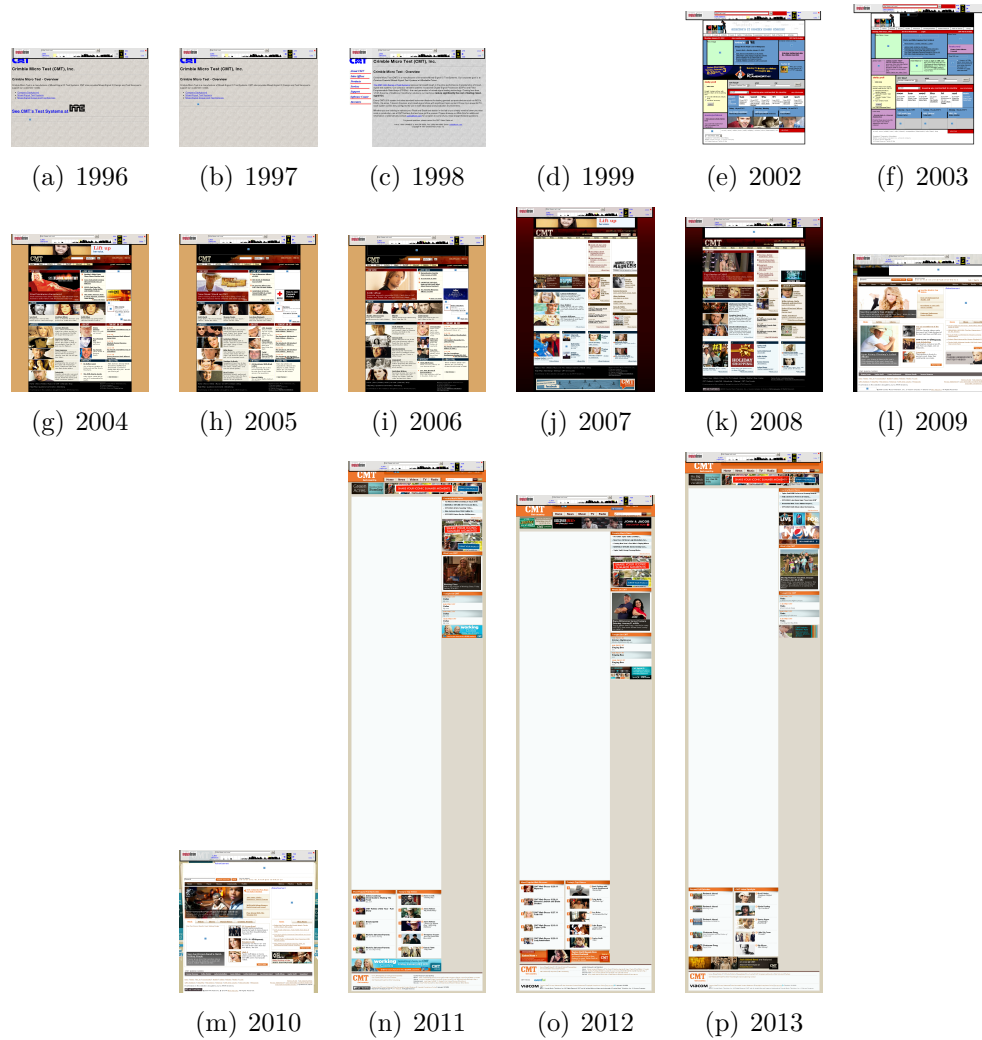


FIG. 65: CMT.com over time. Changes in design and thus the technologies used are easily observable after mementos archived in 2009-2013 (Figures 65(l)-65(p)), which is when jQuery is introduced into the page and used to load embedded resources.

5.12 CHALLENGES IN MEASURING PAST PERFORMANCE

In our research, we have omitted calculating CC and other archivability metrics from mementos in our collections because we do not have the corresponding URI-R from which to compute a baseline. Several scenarios could take place that impact our archivability metrics:

- mementos may have been damaged (missing embedded resources) after the archival process due to implementations of the robots protocol (eliminating mementos from external domains) or limited by policy (such as the case with Archive-It);
- inavailability of embedded resources due to server-side failures or transient errors in the archives [55];
- embedded resources may not have been available to the crawler at archive time.

Without an understanding of the state of the live resource at archive time, we cannot accurately assign CC to the mementos.

While calculating archivability metrics, we must also consider the practices used by each archival tool. For example, the Wayback Machine inserts additional JavaScript and other content to generate the banner and headers to appear in the mementos. Other archival services have similar practices for which we must account. It becomes difficult to measure the original resource when HTML is modified after archiving and during the dereferencing of mementos. This after-archiving injection and loading of scripts and other embedded resources by the archives will impact the calculations we have performed in this research (e.g., CC). The live resources may have also been bound to different standards, and we have no *a priori* knowledge of these standards or their implementations. As such, we have omitted these metrics and measurements.

We instead investigate the availability of embedded resources within our collection over time. The completeness of the memento as it existed at observation time and the source of the embedded resources (either from the HTML or JavaScript) provide a measurement of how JavaScript impacts memento archivability. We loaded all URI-Ms listed in the Internet Archive TimeMaps of URI-Rs in our collection and recorded the resources loaded along with their HTTP response codes. We also recorded the origin of the request; specifically, we note whether the resources were

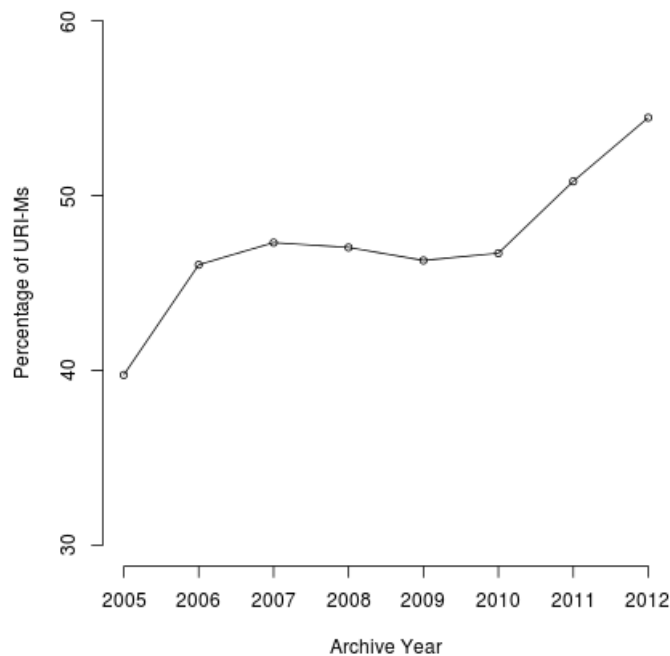


FIG. 66: Resources are using more JavaScript to load embedded resources over time.

loaded because of inclusion in the HTML or as a request from JavaScript. We did not track availability of each embedded resource within a URI-M over time during this experiment, but rather considered just the response codes generated from individual URI-M accesses.

5.13 THE IMPACT OF JAVASCRIPT ON MEMENTO COMPLETENESS

We observed the embedded resources loaded from each memento in our collection from 2005-2012 to determine how many resources are being loaded from JavaScript. As shown in Figure 66, more resources are using JavaScript to load embedded resources over time. In 2005, 39.7% of our collection uses JavaScript to load at least one embedded resource, and continues to increase to 54.5% of the collection using JavaScript to load at least one embedded resource in 2012. That is an increase of 14.7% of resources using JavaScript between 2005 and 2012.

These resources are not only increasingly using JavaScript, but also rely more

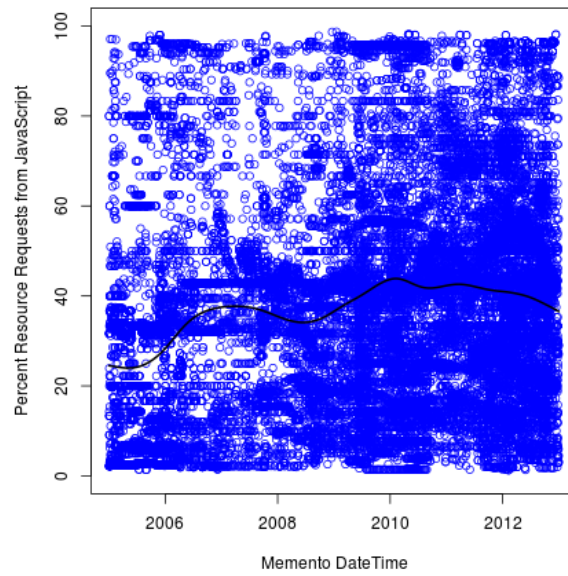
heavily relying on JavaScript to load embedded resources. In Figure 67(a), we plot the percent of requests that come from JavaScript from each URI-M in blue as well as a fitted curve in black. Mementos load 24.5% of all embedded resources through JavaScript in 2005, and 36.5% in 2012, for an increase of 12.0% in seven years. As shown in Figure 67(b), yearly averages also increase from 2005 to 2012. Over our seven year window, mementos load 37.7% of all resources via JavaScript ($\sigma = 23.6\%$).

These increases show that resources are loading a higher proportion of their embedded resources by JavaScript instead of HTML (potentially to provide a more personalized experience to the users). These embedded resources are loaded at run-time by JavaScript instead of when viewed on the client. Embedded resources loaded by JavaScript are harder for Heritrix to archive (since their URI-Rs may not be known during crawl-time), and result in missing content when accessing the mementos from the Wayback Machine.

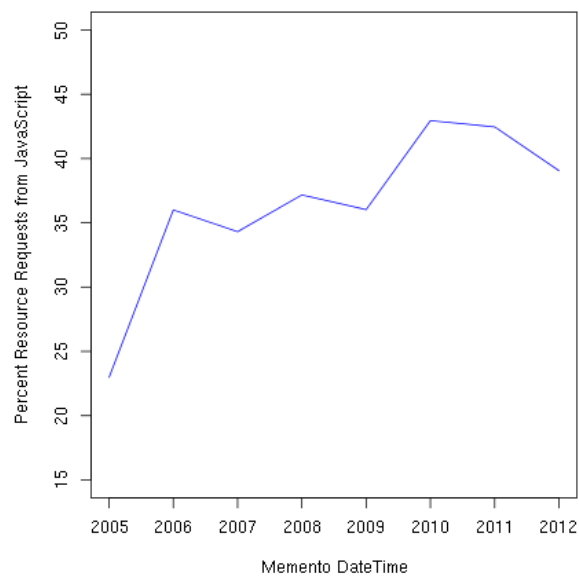
As we discussed in Section 5.8, not all resources are loaded properly when archived. JavaScript is responsible for an increasing proportion of missing embedded resources (Figure 68). We observed that JavaScript accounts for 39.9% of all missing embedded resources in 2005, but is responsible for 73.1% of all missing embedded resources in 2012. A more detailed analysis is provided in the box plots of Figure 68(b) with the missing resources binned according the memento archive year. Box plots provide the calculated 25th percentile as the bottom of the box, the 50th percentile as the horizontal line within the box, and the 75th percentile as the top line of the box. The “whiskers” establish the lower and upper bounds of the data. The dots (for example, those in the box plot of 2008 in Figure 68(b)) in a box plot are outliers in the data.

JavaScript is responsible for 52.7% of all missing embedded resources from 2005 to 2012 ($\sigma = 28.9$) which is 33.2% more missing resources in 2012 than in 2005, showing JavaScript is responsible for an increasing proportion of the embedded resources unsuccessfully loaded by mementos. JavaScript is increasingly requesting mementos that are not in the archives and is responsible for attempting to load over half of all missing resources in our collection. This suggests that an increasing utilization of mementos to load embedded resources via JavaScript, and increasing failures to archive those resources will result in further reduction in memento completeness.

Over time, mementos are increasingly constructed with embedded resources and are requesting and missing an increasing number of embedded resources to render a

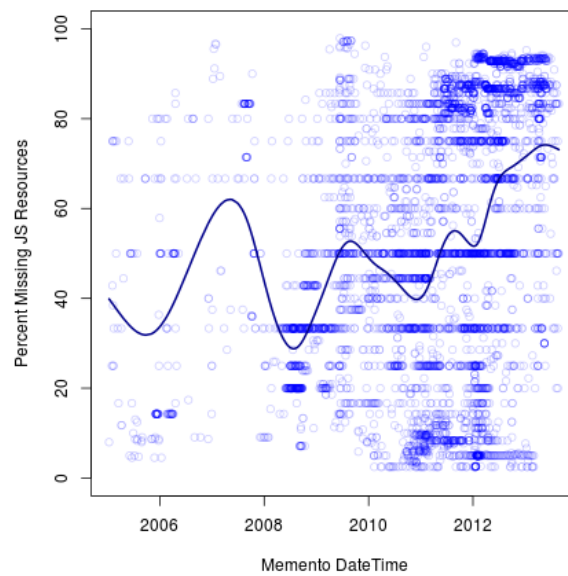


(a) More embedded resources are being loaded via JavaScript over time.

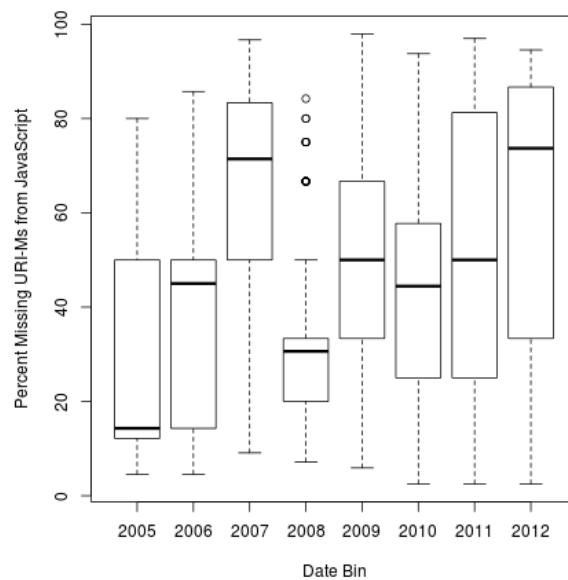


(b) On average, resources are increasingly being loaded by JavaScript per year.

FIG. 67: JavaScript is responsible for loading an increasing proportion of mementos over time.



(a) JavaScript is responsible for an increasing proportion of missing embedded resources over time.



(b) Mementos are binned by archive year to provide a yearly summary of archive performance.

FIG. 68: JavaScript is responsible for an increasing proportion of missing resources.

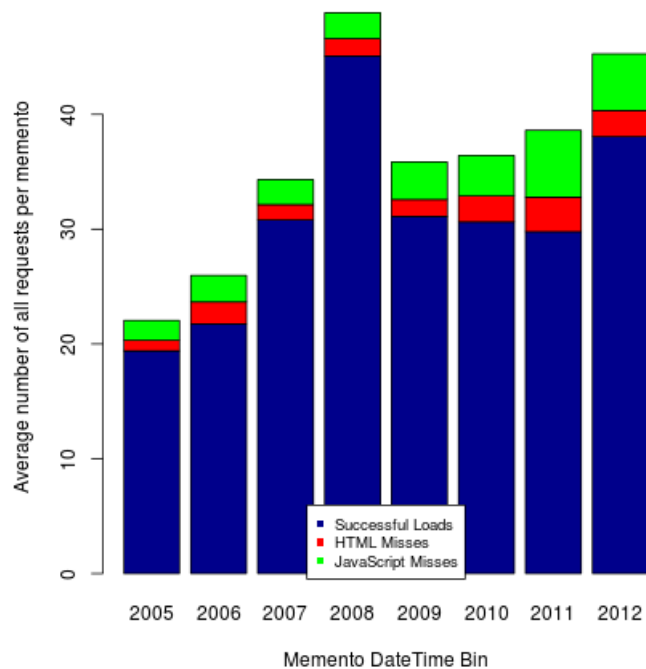


FIG. 69: Number of requests per memento by archive year.

final representation. Table 7 and Figure 69 provide the number of successfully dereferenced URI-Ms, unsuccessful dereferences originating in HTML, and unsuccessful dereferences originating from JavaScript for each memento for each year.

TABLE 7: Number of requests per memento by archive year.

Request Type	2005	2006	2007	2008	2009	2010	2011	2012
JavaScript Misses:	1.7	2.3	2.1	2.3	3.3	3.5	5.8	4.9
HTML Misses:	0.9	1.9	1.3	1.5	1.5	2.3	3.0	2.3
All HTTP 200s:	19.4	21.7	30.8	45.1	31.1	30.6	29.8	38.1

While it is uninteresting that more total attempted dereferences result in more successful and more failed dereferences, the breakdown of how the failed dereferences were loaded provides insight into the source of reduced memento completeness. As the number of requested resources increases, the number of resources unsuccessfully loaded by HTML increases, and the number of unsuccessfully loaded JavaScript resources increases, as well. However, the JavaScript requested resources increase

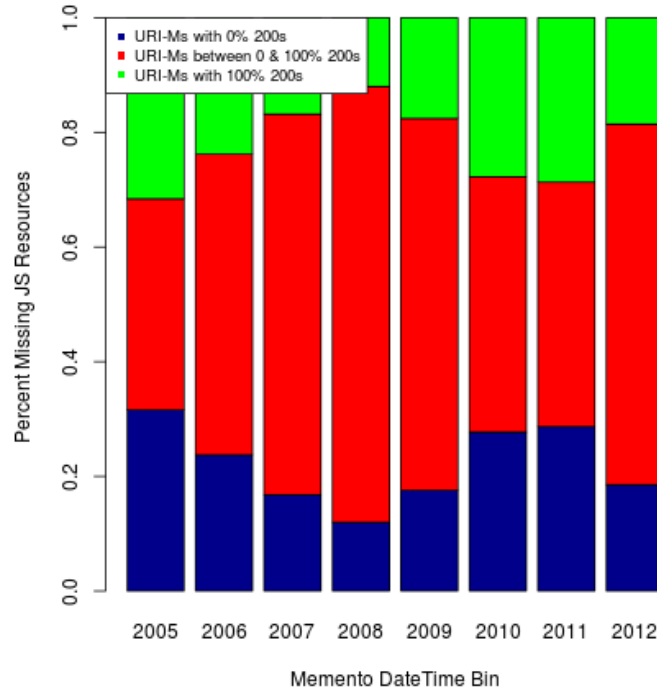


FIG. 70: Percent of missing resources from JavaScript by year.

at a higher rate ($\Delta_{HTML} = 1.4$ requests vs $\Delta_{JavaScript} = 3.2$ requests from 2005 to 2012). This suggests that resources will continue to unsuccessfully load embedded resources via JavaScript as time progresses.

The collections are not dominated by either perfect (0% missing resources from JavaScript) mementos or completely JavaScript reliant (100% missing resources from JavaScript) as shown in Figure 70. The 0% and 100% mementos are approximately equal to each other for every year, meaning the statistics reflect the performance of those resources that load at least (but limited to) a portion of their embedded resources from JavaScript.

As we have hypothesized, an increasing number of requests originating from JavaScript will result in an increased proportion of missing embedded resources. We show the number of requests from JavaScript and the resulting proportion of missing mementos for each URI-M in our collection in Figure 71. The fitted line in black shows the correlation between the two statistics. As the number of requests from JavaScript increases, the number of missing embedded resources increases. This trend is supported by a moderate Kendall Tau-b correlation with $\tau = 0.36$ and $p < 0.01$.

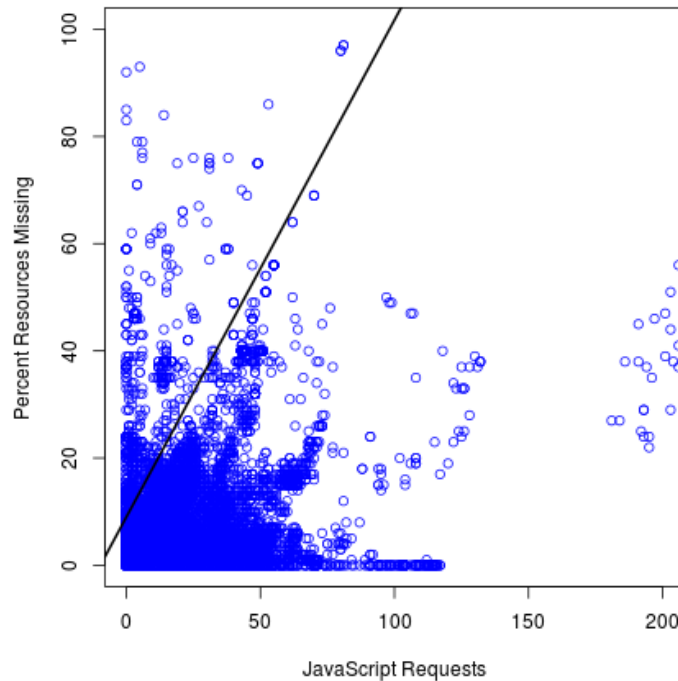


FIG. 71: As JavaScript is relied on more heavily to load embedded resources, more resources are missed.

This correlation suggests that the current trend of relying increasingly on JavaScript to load embedded resources will result in a higher proportion of missing embedded resources and reduced memento completeness.

5.14 SUMMARY OF FINDINGS

We began this chapter with the hypothesis that increasing reliance on JavaScript results in an increase in missing embedded resources in our archives. From the graphs in Figures 66-71 we present four findings en route to confirming our hypothesis.

1. More resources are using JavaScript over time, and mementos use JavaScript more heavily to load embedded resources as time progresses;
2. JavaScript is responsible for an increasing proportion of missing embedded resources in the archives;
3. The number of missing resources loaded by JavaScript is increasing at a higher rate than the number of missing embedded resources loaded by HTML over

time; and

4. The proportion of missing embedded resources is moderately correlated to the number of resources loaded from JavaScript.

These findings support our hypothesis and suggest that increased JavaScript in the archives will result in decreased memento completeness. As time progresses, an increasing number of resources will rely more heavily on JavaScript, resulting in more missed content in the archives. Based on these findings, we recommend that crawlers adapt to archive embedded JavaScript to increase the completeness of mementos. We recommend a two-tiered crawling approach to mitigate this trend, as presented in Chapter 7.

5.15 CONTRIBUTION TO RESEARCH QUESTION 1

The answer to Research Question 1: **To what extent does JavaScript impact archival tools?** is predicated upon a measurement of the impact of JavaScript on the current state-of-the-art archival tools, and therefore the impact of JavaScript on our archives.

This work observed a set of URI-Rs shared over Twitter and a set of URI-Rs previously archived by Archive-It. The Twitter URI-Rs are twice as complex as the Archive-It URI-Rs; the Archive-It URI-Rs are closer to TLDs ($\overline{UC}_{Archive-It}=0.166$, $\overline{UC}_{Twitter}=0.377$). The representations of Twitter resources contain twice as much JavaScript as the Archive-It set ($\overline{CC}_{Archive-It}=2.16$, $\overline{CC}_{Twitter}=4.78$). The shared Twitter resources are more difficult to archive and experience much more leakage (between 0.6% if using Heritrix to 5.8% more if using wget) than their Archive-It counterparts. Only 4.2% of the Twitter collection was perfectly archived by the archival tools in our experiment, while 34.2% of the Archive-It set was perfectly archived by each tool. This shows that the resources that human archivists actively curate and archive are easier to capture with today's tools than URI-Rs being shared over Twitter.

The archivability of websites is changing over time because of an increasing reliance on JavaScript to load resources. JavaScript is responsible for 33.2% more missing resources in 2012 than in 2005, meaning that JavaScript is responsible for an increasing proportion of the embedded resources unsuccessfully loaded by mementos. JavaScript is also responsible for 52.7% of all missing content in our collection.

This trend is expected to increase as time progresses since the number of embedded resources loaded via JavaScript is moderately correlated to the proportion of missing embedded resources. This supports our theory that as resources continue to more heavily utilize and rely on JavaScript to load resources, the completeness of mementos will decrease.

With this work, we show that JavaScript negatively impacts the archives. Deferred representations are hard to archive, have increased leakage, and the problem is trending toward an increasingly prevalent challenge. Archival tools would benefit from mitigating the impact of JavaScript through new crawl techniques or policies. We explore metrics, including completeness, for quantifying how well a memento is archived in the next chapter to better quantify the perceived archivability of a memento.

CHAPTER 6

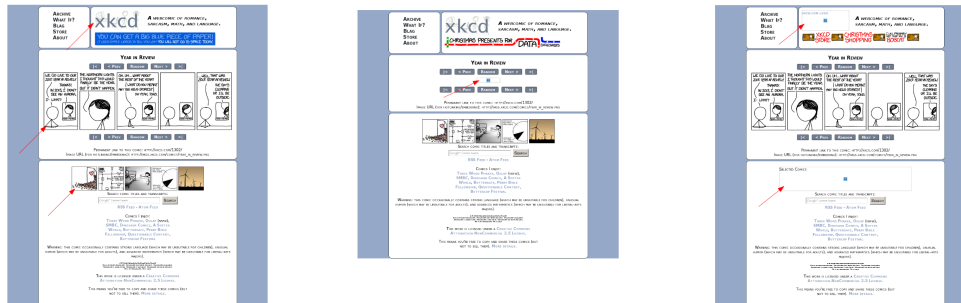
MEASURING ARCHIVE QUALITY

In the previous chapter, we showed that JavaScript has a negative impact on the archives, causing the archives to miss many embedded resources when archiving resources. Research Question 2 asks how we can automatically determine the quality of a memento given the missing embedded resources.

Embedded resources have varying importance to the utility of a memento to Web users. Large images are often more important to a memento’s utility than small images. Similarly, style sheets that format visible content are more important to the representation of the page than style sheets without significant formatting responsibilities. In our work to answer Research Question 2, we provide a mechanism to assess the impact of missing embedded resources in the archives based on a weighted value, improving upon the existing, naïve metric of counting missing embedded resources [50, 51]. A simple measure of the percentage of missing embedded resources is misleading because embedded resources have varying utility and importance to the user’s understanding of the page.

With archives growing to very large scales, automatic assessment of archive quality is essential; humans would require a massive amount of time and labor to complete an assessment. However, current automated methods are not accurate reflections of user interpretation of quality.

In order to sufficiently answer Research Question 2: **How do we measure memento quality?**, we achieved three goals. First, we wanted to understand how missing embedded resources impact Web user perception of memento quality or satisfaction (i.e., the utility of mementos). Using an algorithm to measure embedded resource importance, we determined whether an important embedded resource of the memento is missing (e.g., a main image or video essential to the user’s understanding of the page), or if the missing embedded resource contributes little to the memento’s utility for the user (such as a spacer image or small logo). We proposed a method of weighting embedded resources in a memento according to importance. We showed that this is an improved damage rating over an unweighted count of missing embedded resources. We used Amazon’s Mechanical Turk [152] to compare our algorithm



(a) All three of the embedded images are included in m_0 and identified by the red arrows ($M_m=0.17$).

(b) We removed the large, central image (that is the main content of the page) from m_1 , identified by the red arrow ($M_m=0.24$).

(c) We removed the XKCD logo and banner of comics from m_2 , identified by the red arrows ($M_m=0.29$).

FIG. 72: The XKCD example demonstrates that embedded resources have varying human-perceived importance to their page.

to Web users' notion of damage and to show an improvement over the unweighted count of missing embedded resources.

Second, we used our algorithm to assess the damage of mementos in the Internet Archive and WebCite. We compared the proportion of missing embedded resources to all requested resources (M_m) with our weighted calculation of damage (D_m).

Third, we measured damage in the Internet Archive and WebCite over time using D_m . We describe how this algorithm can be used for future enhancements of the Heritrix crawler and Internet Archive's archival processes. We also discuss the impacts of JavaScript on archive quality, using WebCite as the target of our discussion and compare WebCite's memento quality to mementos created by Archive.is.

6.1 MOTIVATING EXAMPLES

We use the XKCD Web page as an example of a resource with embedded resources of differing importance. We captured the XKCD URI-R using the wget command¹ and manually inflicted damage on a local memento of <http://www.xkcd.com/> by

¹We executed the wget command with parameters as follows: `wget -E -H -k -K -p http://www.xkcd.com/`



www.cityofmoorhead.com/flood/

[Back to City of Moorhead](#)

The City of Moorhead's Official Flood Information Web Site

[Skip to content](#)

- [Home](#)
- [Announcements](#)
 - [News Releases](#)
 - [Dike Status](#)
- [Slideshow](#)
- [Get Involved](#)
 - [Volunteers](#)
 - [Donations](#)
- [City Services](#)
 - [Garbage/Recycling](#)
 - [Utilities](#)
 - [Mass Transit](#)
- [FAQs](#)



(a) This memento (URI-M <http://web.archive.org/web/20110116022653/http://www.cityofmoorhead.com/flood/>) is missing two style sheets which changes the entire appearance and utility of the memento ($M_m=0.38$).



(b) Meanwhile, this memento (URI-M <http://web.archive.org/web/20060102083228/http://www.ascc.edu/>) is missing two style sheets (along with two images) but does not appear damaged ($M_m=0.20$).

FIG. 73: Mementos have different meanings and usefulness depending on which embedded resources are missing from the memento (and the proportion of missing resources, M_m).

removing embedded images. We used PhantomJS to dereference the local URI-M and take a PNG snapshot of the representation and recorded the resulting HTTP response headers of the embedded resources. We created three mementos of the URI-R: one duplicating its live Web counterpart (m_0), one with the central comic image removed (m_1), and one with two logo images removed (m_2). The snapshots taken by PhantomJS are provided in Figures 72(a), 72(b), and 72(c). As shown in the captions, the proportion of embedded missing resources (M_m) varies among the mementos.

When we performed this test, the live XKCD site was missing two embedded cascading style sheets (or simply “style sheets”), as are m_0 , m_1 , and m_2 since they are copies of the live site. We verified that our memento m_0 has a M_m value identical to its live Web counterpart – the live resource and m_0 are both missing the same embedded resources ($M_m=0.17$). In Figure 72(a), m_0 has multiple embedded resources, but we focus on the three identified by the red arrows: the XKCD logo, the main comic image, and the banner of comics. The central image is most important to the utility of the page – without the main comic image, the user does not obtain the information from the page that the author intended (Figure 72(b)). The logo and banner are not essential to the user’s understanding of the XKCD content (Figure 72(c)).

CSS files also differ in importance. Some style sheets are responsible for formatting small portions of a page, while others are responsible for placing images and other content or even organizing the entire page for the user. Figure 73(a) shows a memento of a URI-R that is missing a single style sheet. This style sheet is responsible for a large amount of information in the representation and without it, the meaning and utility of the memento changes. Figure 73(b) shows a memento that is properly styled but is missing two style sheets that are not responsible for the majority of the content organization.

As we have discussed, the percentage of successfully dereferenced embedded resources is not the only factor in determining memento quality. In support of that principle, we refer to Figure 73(b) in which $M_m=0.2$ (6/30). However, it appears to be well-preserved. In our XKCD example, Figure 72(c) is missing two images ($M_m=0.24$) yet maintains more important embedded mementos than Figure 72(b) ($M_m=0.29$). These examples support the motivation of our research by demonstrating that unweighted percentages (i.e., M_m) are insufficient to assess perceived memento damage.

6.2 USERS' PERCEPTION OF DAMAGE

As archivists, our perception of damage differs from that of more “traditional” Web users. As such, we needed to enlist the help of more traditional Web users to assess damage. To determine if M_m (percent missing) is a good estimate of human perception of damage, we used Amazon’s Mechanical Turk to measure Web user agreement with M_m .

To ensure that Mechanical Turk workers (or more colloquially, “turkers”) could evaluate damage, we presented turkers with pairs of mementos – one of which was not damaged and the other that had a varying level of damage – and asked them to select the memento they preferred to keep if given a choice between the two.

We captured 11 hand-selected URI-Rs (Table 1) on a local server (similar to the process in Section 5) and created five versions of the mementos for each URI-R. We manually damaged the mementos to create the five categories of damage by removing an embedded resource. For the category *missing image*, we removed a prominent image (empirically identified as important) from the memento. For the category *missing css*, we removed a prominent CSS file to cause formatting issues in the memento; we empirically selected the CSS file to remove based on the greatest human-perceived detrimental impact to the page layout. We also created the categories *missing all images* (we removed every embedded image), *missing all resources* (we removed all embedded resources), and *original* (the URI-M was a direct copy of the live resource) and measured the M_m of each URI-M in each category. We refer to a memento from any of these categories as m_1 and the *original* as m_0 . These categories created several degrees of damage through a variety of missing embedded resources for identical URI-Rs at an identical time point to provide a wide spectrum of mementos to be evaluated by turkers.

With the goal of determining whether or not turkers can recognize damage in a memento, we presented the turkers with a m_1 and its m_0 counterpart (that is, a “damaged” and its *ground-truth* memento) and asked the turkers, “We saved two pages for you. For which page did we do a better job?” (Figure 74). For each URI-R, a pair of mementos consisting of m_0 and one of the four categories of m_1 were evaluated by five turkers for a total of 280 evaluations.

We show the judgement splits from the turker evaluations in Table 9. The judgement splits refer to the number of turkers that selected the correct-incorrect version. For example, a 0-5 split means all five turkers selected the m_1 memento (an incorrect

URI-R	M_m				
	m_0	missing image	missing css	missing all images	missing all
http://www.cs.odu.edu/~mln/	0.14	0.43	0.29	0.43	0.43
http://activehistory.ca/2013/06/myspace-is-cool-again-too-bad-they-destroyed-history-along-the-way/comment-page-1/	0.0	0.32	0.32	0.57	0.85
http://www.albop.com/	0.0	0.13	0.0	0.50	0.50
http://www.cs.odu.edu/	0.10	0.13	0.11	0.82	0.81
http://ws-dl.blogspot.com/2013/08/2013-07-26-web-archiving-and-digital.html	0.07	0.08	0.08	0.13	0.14
http://www.cnn.com/2013/08/19/tech/social-media/zuckerberg-facebook-hack/	0.19	0.22	0.28	0.46	0.57
http://xkcd.com/	0.14	0.38	0.31	0.53	0.54
http://www.mozilla.org/	0.80	0.80	0.80	0.877	0.89
http://www.ehow.com/	0.05	0.05	0.06	0.11	0.33
http://google.com/	0.0	0.0	0.0	0.0	1.0
http://php.net/	0.32	0.33	0.33	0.37	0.37

TABLE 8: The 11 URI-Rs used to create the manually damaged dataset. M_m values are provided for each m_1 .

You are using the Mechanical Turk Developer Sandbox. This site is for test and development only. [Learn more.](#)

amazon mechanical turk Artificial Intelligence Sign In

Your Account | **HITS** | Qualifications **244,928 HITS** available now

All HITS | HITS Available To You | HITS Assigned To You

Find **HITS** containing that pay at least \$ **0.00** ☐ for which you are qualified ☐ provide Master Qualification

Timer: 00:00:00 of 20 minutes Want to work on this HIT? Total Earned: Unavailable
Total HITS Submitted: 0

Which version of the website is better?
Requester: Hury
Qualifications Required: Masters has been granted
Reward: \$0.01 per HIT
HITS Available: 1
Duration: 20 minutes

Which version of the website is better?

We saved two versions of the same website and are presenting them to you. These two versions have some differences.

Which version did we do a better job saving? A version of a webpage is "Better" only if it is more complete, usable, and provides the expected information.

Instructions:

- Wait until the two different versions load completely.
- Look at each version and scroll through.
- Click the radio button for the version you think is better.

Version 1 of the Website: ☐ Version 2 of the Website: ☐



☐ **Version 1 is better** *(we did a better job in saving version 1 than version 2)*

☐ **Version 2 is better** *(we did a better job in saving version 2 than version 1)*

Provide feedback on our work.

You must ACCEPT the HIT before you can submit the results.

Want to work on this HIT?

[FAQ](#) | [Contact Us](#) | [Careers at Mechanical Turk](#) | [Developers](#) | [Press](#) | [Policies](#) | [Blog](#)
©2005-2014 Amazon.com, Inc. or its Affiliates

An **amazon** company

FIG. 74: We asked the turkers to select the less damaged of two mementos. The two versions of the page are accessible in separate tabs.

ΔM_m	Splits						Total
	5-0	4-1	3-2	2-3	1-4	0-5	
1.0							0.00
0.9							0.00
0.8	4						0.07
0.7							0.00
0.6							0.00
0.5	1	1					0.04
0.4							0.00
0.3	15	5					0.36
0.2	2						0.04
0.1	5	4	4	2		1	0.29
0.0	5	3	1	3			0.22
Total	0.58	0.23	0.09	0.09	0.00	0.02	1.0

TABLE 9: The turkers selected m_0 as the preferred memento 81% of the time, and more consistently for larger ΔM_m values.

Turker Assesment	M_m	
	Select m_0	Select m_1
m_0	44	0
m_1	11	0

TABLE 10: Confusion matrix of the turker assessments of the m_0 vs m_1 comparison test.

selection), a 5-0 split means all five turkers selected the m_0 memento (the correct selection), and a 3-2 split means three turkers selected the m_0 memento and two selected the m_1 memento (a correct selection by the majority, but still a split decision among the turkers). For the purposes of Research Question 2, we consider only 5-0 and 4-1 splits as agreement with M_m and all other splits as disagreement. ΔM_m refers to the delta between M_{m_0} and M_{m_1} .

The turkers selected m_0 as the preferred option (less damaged memento) 81% of the time (226/280). As ΔM_m grows, turker agreement is more consistent.

Regardless of ΔM_m , 81% of the evaluations agreed with M_m as a suitable damage metric (5-0 and 4-1 splits). Turkers were unsure about the damage (3-2 and 2-3 splits) 18% of the time and incorrectly identified damage only once. The average ΔM_m for the unsure selections was < 0.01 , and the only 0-5 split had a ΔM_m of

0.014, suggesting that confusion or disagreement occurs more often when the damage delta is smaller (i.e., the mementos are too similar to differentiate between).

Confusion matrices provide a consolidated view of an algorithm’s performance. The top left quadrant shows the number of true positives, the top right shows the number of false negatives, the bottom left shows false positives, and the bottom right shows true negatives. The algorithm’s accuracy (Equation 5) and harmonic mean (also called *F-measure*, or F_1 Score, Equation 6) are calculated using a confusion matrix. A harmonic mean provides an average (in this case, of the algorithm’s success rate) and is sensitive to small values and outliers. Accuracy is defined as the number of correctly classified instances divided by the test set size (Equation 5). F-Measure goes beyond accuracy to consider the harmonic mean of precision and recall (defined in Equation 6).

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Number of Classifications}} \quad (5)$$

$$\text{F-Measure} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \quad (6)$$

From the confusion matrix (Table 10), we can calculate $F_1=0.88$ and accuracy=0.80 for m_0 vs m_1 . Turker agreement does not match M_m 100% of the time with the m_0 vs m_1 test because of phenomena with aesthetics and human perception. Also, m_0 is often incomplete (with $M_{m_0} > 0$ e.g., php.net in Table 8).

6.3 EVALUATING ORGANIC DAMAGE

Turkers identified m_0 in the m_0 vs m_1 in a large majority (81%) of the comparisons. As a result, the turkers have shown that they can identify a damaged memento when presented a damaged and undamaged memento. Because they can identify damage in mementos, we felt comfortable using turkers to evaluate our measured damage of mementos found in the Internet Archive.

This experiment uses the same set of 2,000 URI-Rs as in Section 5 and our previous work [52] as sampled from Twitter and Archive-It. Using these two sets of URI-Rs, we measured the damage of one memento per year from the Internet Archive TimeMap of each of the 1,861 URI-Rs, resulting in 45,341 URI-Ms. We randomly selected a subset of 100 URI-Ms from this set. Similar to the evaluation in Section 6.2, we gave turkers two mementos (we will generalize these to m_2 and m_3) from

ΔM_m	Splits						Total
	5-0	4-1	3-2	2-3	1-4	0-5	
1.0					1		0.01
0.9							0.00
0.8							0.00
0.7		1					0.01
0.6					1		0.01
0.5							0.00
0.4		1					0.01
0.3	1		3	4	1	2	0.11
0.2		5	6	5	12	9	0.37
0.1	4	5	10	11	15	3	0.48
0.0							0.00
Total	0.05	0.12	0.19	0.20	0.30	0.14	1.0

TABLE 11: The turker evaluations of the m_2 vs m_3 comparisons when using M_m as a damage measurement.

consecutive years from the same TimeMap and asked the turkers to select the less damaged memento (“We saved two pages for you. For which page did we do a better job?”) as shown in Figure 74. Because m_2 and m_3 are observed from the Internet Archive, neither is considered a *ground-truth*. We measured M_m of mementos in the Internet Archive and compared it to the turker perception of the utility of the mementos.

Contrary to the test in Section 6.2, as ΔM_m grows, the turkers are not as overwhelmingly effective at selecting the less damaged memento (the splits are shown in Table 11). The turkers only agree with M_m 12% of the time (5-0 and 4-1 splits) and completely disagree with M_m (1-4 and 0-5 splits) 44% of the time. This discrepancy demonstrates that turker assessment of damage does not match M_m . Additionally, we see that the turkers performed well when comparing m_0 vs m_1 (original vs damaged) but struggle to compare m_2 vs m_3 (damaged vs damaged).

From the confusion matrix (Table 12), we can calculate the accuracy of turker selections of m_2 vs m_3 agreement with M_m is 0.46 with $F_1=0.55$. In a Receiver Operating Characteristic (ROC) curve [87], we calculated the Area Under the ROC Curve (AUC) for the results of the turker evaluations of m_2 vs m_3 against M_m and the results of the manually damaged m_0 vs m_1 test (as the optimal performance). The AUC of M_m is lower (AUC=0.472) than random (AUC=0.500) as shown in Table

Turker Assesment	M_m	
	Select m_2	Select m_3
m_2	29	24
m_3	23	24

TABLE 12: Confusion matrix of the turker assessments of the m_2 vs m_3 comparison test against M_m .

Damage Calculation	AUC	F_1	Accuracy
M_m	0.472	0.55	0.46
M_{m_0}	0.789	0.88	0.80

TABLE 13: When compared to random, M_m performs worse than random selection and is worse than the optimal performance of m_0 vs m_1 .

13, meaning that M_m performed worse than random for matching turker perception of damage and far worse than the optimal performance (AUC=0.789), a further indicator that M_m is not a suitable metric for measuring memento damage.

6.4 CALCULATING MEMENTO DAMAGE

With M_m not matching Web users' perception of damage, we propose a new algorithm for assessing memento damage. Our proposed algorithm is based on the file type, size, and location of the embedded resource.

6.4.1 DEFINING D_M

We define D_m as the damage rating, or cumulative damage, of a memento m in Equation 7. D_m is a normalized value ranging from $[0, 1]$. We calculate the potential damage of a memento and the actual damage of a memento and express the damage rating as the ratio of actual to potential damage. Notionally, potential damage is the cumulative importance of all embedded resources in the memento (i.e., those that could potentially be missing), while actual damage is only the importance of those embedded resources that are unsuccessfully dereferenced, or missing. That is, *actual* damage is the cumulative weighted importance of the embedded resources missing from a memento m , while the *potential* damage is the weighted importance of embedded resources both present and missing in a memento m . Actual damage is

the damage done to a page while the potential damage is the damage that would be measured if all embedded resources were missing.

$$D_m = \frac{D_{m_{actual}}}{D_{m_{potential}}} \quad (7)$$

To determine potential and actual damage, we first define the set of all embedded resources R and the set of all missing resources R_r in Equation 8.

$$\begin{aligned} R &= \{\text{All embedded resources requested}\} \\ R_r &= \{\text{All missing embedded resources}\} \\ R_r &\subseteq R \end{aligned} \quad (8)$$

6.4.2 WEIGHTING EMBEDDED RESOURCES

We calculate the importance of each embedded resource in the set R . The sum of each embedded resource is the potential damage $D_{m_{potential}}$ (Equation 9). Important resources are assigned additional weights to increase their relative value over unimportant resources (Equations 11 - 12).

$$\begin{aligned} D_{m_{potential}} &= \frac{\sum_{i=1}^{n_{[I,MM]}} D_{[I,MM]}(i)}{n_{[I,MM]}} + \frac{\sum_{i=1}^{n_C} D_C(i)}{n_C} \\ &\forall \{I=\text{Images}, MM=\text{Multimedia}, C=\text{CSS}\} \\ &n \in R \end{aligned} \quad (9)$$

Actual damage ($D_{m_{actual}}$, defined in Equation 10) is identical to $D_{m_{potential}}$ except it is computed using only the missing embedded resource set R_r .

$$\begin{aligned} D_{m_{actual}} &= \frac{\sum_{i=1}^{n_{[I,MM]}} D_{[I,MM]}(i)}{n_{[I,MM]}} + \frac{\sum_{i=1}^{n_C} D_C(i)}{n_C} \\ &\forall \{I=\text{Images}, MM=\text{Multimedia}, C=\text{CSS}\} \\ &n \in R_r \end{aligned} \quad (10)$$

In M_m , all embedded resources are treated as equal. The potential damage is therefore the number of embedded resources, and the actual damage is the number of missing embedded resources. M_m is the unweighted ratio of missing embedded resources to total embedded resources.

We assign additional weights to important embedded resources at the expense of less important mementos. When a weight w is given to an embedded resource, all n embedded resources lose $\frac{w}{n}$ importance, which redistributes the importance between embedded resources while keeping the sum of all importance constant.

6.4.3 IMAGE DAMAGE CALCULATION

Images receive weights for image size and centrality (Equation 11). We use the pixel area (width \times height) of the image as specified in the HTML and the page size along with a weight for horizontal and vertical central dividing line overlap by the image. This gives a weighted measure of the proportion of a page the image occupies and how central the image is in the viewport. For example, we can extract the width and height of the missing embedded resource “IMAGE.png” from this HTML

```

```

but not this HTML

```
.
```

$$\begin{aligned}
 D_{[I|MM]} &= 1 + \frac{width \times height}{\text{Page Size (pixels)}} + w_{\text{horizontal}} \\
 &\quad + w_{\text{vertical}} \\
 w_{\text{horizontal}} &= \begin{cases} 0.25 & \text{image overlaps horizontal center} \\ 0 & \text{otherwise} \end{cases} \\
 w_{\text{vertical}} &= \begin{cases} 0.25 & \text{image overlaps vertical center} \\ 0 & \text{otherwise} \end{cases}
 \end{aligned} \tag{11}$$

Embedded multimedia importance (D_{MM}) is calculated identically to image importance D_I , and we represent both in the same equation $D_{[I|MM]}$. Because size and centrality determine multimedia importance, we omit audio and other non-visual multimedia resources. We also classify Flash movies as multimedia.

6.4.4 STYLE SHEET DAMAGE CALCULATION

Equation 12 outlines the damage from missing style sheets, including a factor for a style threshold w_{style} and a threshold for non-matching CSS tags in the DOM w_{tags} .

$$\begin{aligned}
D_C &= 1 + w_{style} + w_{tags} \\
w_{style} &= \begin{cases} 0.50 & >75\% \text{ content in left two thirds} \\ 0 & \text{otherwise} \end{cases} \\
w_{tags} &= \begin{cases} 0.50 & \text{tags in DOM but not CSS} \\ 0 & \text{otherwise} \end{cases}
\end{aligned} \tag{12}$$

Traditional Web design (and particularly design supported by style sheets) evenly distributes content across each of the vertical thirds of a page. Our intuition is that an important style sheet, when missing, will shift content to the left of the page rather than center content in the viewport. To identify this phenomenon, we divide the PNG snapshot of a memento into vertical thirds and measure the amount of content in each third. If a style sheet is missing *and* content appears to be shifted to primarily the left two-thirds, we assume the missing style sheet was important to the distribution of content on the page.

When detecting content in the PNG snapshot, we use remaining CSS markup and files and the HTML to determine the background color of the page. For each pixel in the PNG, we measure the number of background and non-background colored pixels, with content being the number of non-background colored pixels. The proportion of non-background colored pixels in each vertical third gives us the amount of content in each partition.

The style threshold to identify an important style sheet is determined as follows:

1. Determine the background color
2. Render a PNG snapshot of the page
3. Divide the PNG into equal vertical third partitions
4. Calculate the number of pixels of the non-background color in each third for the viewport only (we used a 1024x768 viewport) and entire page
5. If $\leq 75\%$ of the non-background colored pixels are in the left two thirds of the viewport, set $w_{style} = 0$ in Equation 12 (CSS file does not receive a weight)
6. If $> 75\%$ of the non-background colored pixels are in the left two thirds of the viewport and a style sheet is missing, set $w_{style} = 0.5$ in Equation 12 (CSS file does receive a weight)

For example, we created two mementos of the URI-R <http://www.pilotonline.com/> on a local server, one as it appears live (with all style sheets – Figure 75(a)) and the other with its style sheets removed (Figure 75(b)). The vertical partitions extend from the top of the PNG snapshot to the bottom. The percent of non-background color pixels in the viewports of our mementos are shown in their respective thirds in Figure 74. Notice that the non-background pixels (text, images, etc.) shift left when the CSS is missing. Intuitively, information is not meant to be displayed like the representation in Figure 75(b).

When we consider content outside of the viewport (Figures 76(a) and 76(b)), we see the same shift of content to the left when style sheets are missing. However, the distribution of content in Figure 76(b) is more evenly distributed because the content has shifted down and fills out the middle and right vertical partitions more than in Figure 75(b). This is an indicator that the style sheets missing in Figures 75(b) and 76(b) were important.

Along with the style threshold, the presence of tags on the page without a matching style suggests that the missing CSS contained the referenced formatting. If such tags exist without a matching style, $w_{tags} = 0.5$ in Equation 12.

6.4.5 THE D_M ALGORITHM

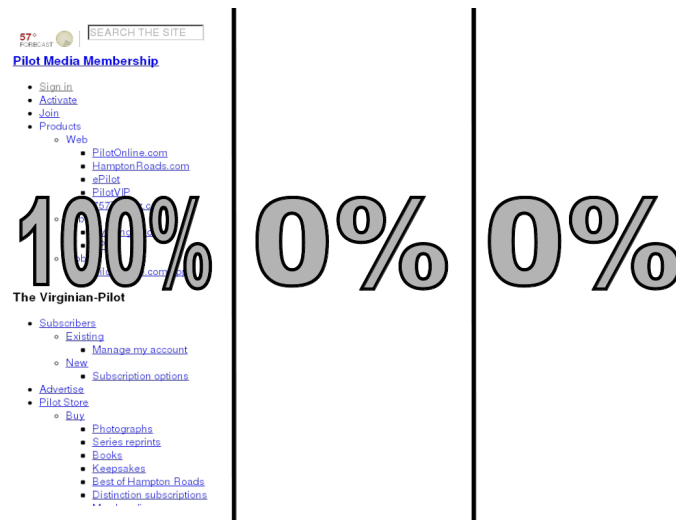
Embedded multimedia, images, and style sheets do not account for the entirety of a page's importance and usefulness. We assume that text, as defined by the DOM and included on the page, is available regardless of archival success (M_m) and therefore does not contribute to the damage calculation.

In summary, Equations 7 - 12 are used to compute D_m :

1. Load URI-M with PhantomJS
2. Find Potential Damage $D_{m_{potential}}$ (Equation 9)
 - (a) Determine CSS importance D_C (Equation 12)
 - (b) Determine Multimedia importance D_{MM} (Equation 11)
 - (c) Determine Image importance D_I (Equation 11)
3. Determine proportion of unsuccessfully dereferenced embedded resources M_m (Equation 8)



(a) We calculated that the non-background color is more evenly distributed between the three vertical partitions of the Pilot Online page with its style sheet included than when it is missing.



(b) We calculated that the non-background color is most prevalent in the left-most vertical partition of the viewport of the Pilot Online page when it is missing its style sheet.

FIG. 75: Missing style sheets causes content to shift left. We show the percent of content in the vertical partitions of the viewport.



(a) When considering the entire page, the content of the page is distributed 33% in the left, 26% in the middle, and 41% in the right partitions when the style sheet is present.



(b) When considering the entire page, the content of the page is distributed 84% in the left, 15% in the middle, and 1% in the right partitions when the style sheet is missing.

FIG. 76: Missing style sheets causes content to shift left. We show the percent of content in the vertical partitions of the page.

Figure	D_m	M_m	Human Assessment
72(a)	0.09	0.17	Acceptable
72(b)	0.41	0.24	Damaged
72(c)	0.36	0.29	Acceptable
73(a)	0.59	0.38	Damaged
73(b)	<0.01	0.20	Acceptable

TABLE 14: D_m vs M_m for the images in Figures 72 and 73. Note $M_m > D_m$ in 2 of 5 cases.

4. Find Actual Damage $D_{m_{actual}}$ (same as Step 3, but with only those URI-Ms unsuccessfully dereferenced)
5. Determine total damage $D_m = [0, 1]$ (Equation 10)

With D_m defined, we revisit the examples presented in Section 6.1. The values for D_m and M_m are listed in Table 14. Note that the D_m ratings are closer to our empirical human assessment of memento quality than the proportion of the embedded resources that are missing.

6.4.6 LIMITATIONS OF D_M CALCULATION

Not all pages and page construction methods can be evaluated by D_m . An edge case not handled by this algorithm is any page constructed with iframes. Our algorithm uses JavaScript to determine the rendered location of embedded multimedia and images. When the embedded media is in a page embedded within another page, our algorithm does not provide the accurate rendered location. For this reason, we exclude iframes from our algorithm. We also exclude missing audio-only multimedia.

While D_m includes multimedia calculations, multimedia resources are rarely embedded in our mementos (only observed twice in our entire set of 45,341 URI-Ms). We observed that multimedia is often loaded by JavaScript files embedded in the DOM; this prevents the multimedia files from being archived because archival crawlers (at the time of this experiment) do not execute client-side JavaScript and therefore do not archive deferred representations.

Further, the JavaScript files may not operate properly when archived [47] and may not issue a request for the target multimedia files. If the JavaScript operates properly and makes an HTTP GET request, the multimedia file would be missing

(since it is not archived) and we would observe more missing embedded multimedia files.

The D_m measurement (and its constituent weights) was constructed by archivists as an improvement to the metric M_m currently used for archive quality assurance. We do not assert that D_m is a perfect measure, but rather an improvement that will require additional investigation and re-weighting to reach perfect agreement with turker evaluation. We recognize that D_m should be more finely tuned to more accurately reflect turker opinion of damage. We also avoid defining a threshold for damage acceptance; this is left to the discretion of the archivist utilizing D_m to measure damage in an archive.

6.5 DAMAGE IN THE ARCHIVES

We measured D_m values for each of the 45,341 URI-Ms from Section 6.3. We used these measurements to assess D_m 's performance relative to turker assessment and to perform damage measurements in the Internet Archive.

6.5.1 TURKER ASSESSMENT OF D_M

We compared D_m to turker assessment and to M_m . As shown in Table 15, D_m aligns with turker assessment of damage 32% of the time (5-0 and 4-1 splits, as indicated in **bold**), an increase of 18% over M_m . Additionally, 49% tie with a 3-2 or 2-3 split and only 18% of the turker evaluations disagreed with the D_m measure. Turkers agree more consistently when ΔD_m is larger. If we only consider $\Delta D_m > 0.30$, the turkers agree with D_m 71% of the time. However with $\Delta M_m > 0.30$, the turkers agree only 20% of the time.

From the confusion matrix in Table 16, we determined that the accuracy of D_m when comparing m_2 vs m_3 is 0.60, and $F_1 = 0.69$. This is an improvement of 0.14 over the accuracy of M_m and an improvement over the harmonic mean of M_m by 0.14, showing that D_m measures damage closer to turker perception. We also calculated the AUC in a ROC curve for D_m and compared it to M_m and the optimal performance of the m_0 vs m_1 test. As shown in Table 17, D_m has an AUC of 0.584, an increase in 0.108 over M_m , showing that D_m outperforms M_m and is closer to the optimal performance of m_0 vs m_1 (AUC=0.789).

ΔD_m	Splits						Total
	5-0	4-1	3-2	2-3	1-4	0-5	
1.0							0.00
0.9		1					0.01
0.8							0.00
0.7							0.00
0.6			1				0.01
0.5							0.00
0.4	4	1					0.05
0.3	2	2	3				0.07
0.2		2	1	2	2	1	0.08
0.1	4	16	27	15	12	3	0.77
0.0							0.00
Total	0.10	0.22	0.32	0.17	0.14	0.04	1.0

TABLE 15: The turker evaluations of the m_2 vs m_3 (sampled mementos) comparisons when using D_m as a damage measurement.

Turker Assesment	D_m	
	Select m_2	Select m_3
m_2	45	32
m_3	8	14

TABLE 16: Confusion matrix of the turker assessments of the m_2 vs m_3 comparison test against D_m .

Damage Calculation	AUC	F_1	Accuracy
M_m	0.472	0.55	0.46
D_m	0.584	0.69	0.60
M_{m_0}	0.789	0.88	0.80

TABLE 17: D_m provides a closer estimate of turker perception of damage and our optimal performance of m_0 vs m_1 than M_m .

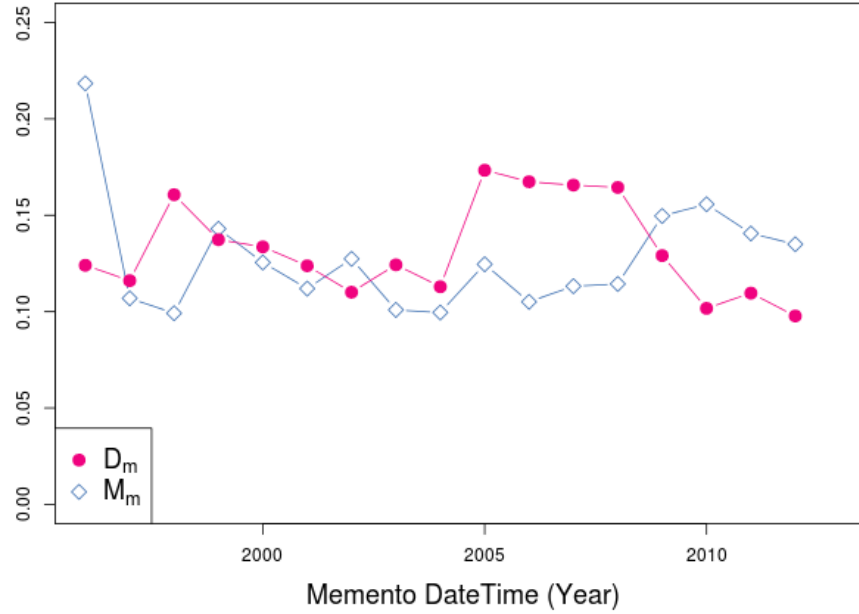


FIG. 77: The average percentage of embedded resources missed per memento per year in the Internet Archive as compared to damage per memento per year ($\overline{D_m}=0.128$, $\overline{M_m}=0.132$).

6.5.2 MEASURING THE INTERNET ARCHIVE

With D_m validated as aligning closer to turker evaluations than M_m , we used D_m to evaluate the Internet Archive’s damage rating over time. Our measurement shows that only 46% of the 45,341 URI-Ms listed in the 1,861 TimeMaps are complete – that is, 54% of all URI-Ms listed in the Internet Archive TimeMaps we studied are missing at least one embedded resource². In Figure 77, we show the average number of missing embedded resources M_m along with the average calculated damage D_m per URI-M per year.

Because the number of missed mementos is relevant to M_m and D_m , we investigated the occurrence of missing and successfully dereferenced embedded resources. Mementos in our dataset are missing very few embedded resources with most missing 1-10 embedded resources (shown as a histogram and Cumulative Distribution

²The Internet Archive performs URI canonicalization very well, and is assumed to not be a source of missing resources.

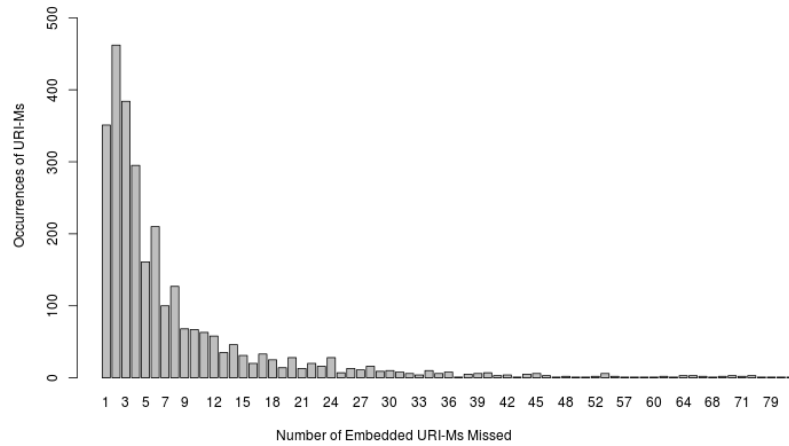
Function (CDF) in Figures 78(a) and 78(b)) ($\overline{\text{Missing Resources}} = 1.7$, $s = 4.6$, $Md = 3$)³. We note the long tail on this distribution; a few mementos are missing a very large number of embedded resources (maximum is 116). We calculate that 61% of mementos are missing 3 or fewer embedded resources, and 85% of mementos are missing 6 or fewer embedded resources. Most mementos have very few embedded resources ($\overline{\text{Embedded Resources}} = 17.6$, $s = 86$, $Md = 7$), as shown in Figures 79(a) and 79(b). A few mementos have a very large number of embedded resources (maximum is 552).

In aggregate, we observed that 45,009 of 292,192 embedded resources were missing, meaning 15% of the embedded resources in the dataset are missing. Of these, 25,848 (57% of the missing URI-Ms) were important, meaning they were assigned an additional weight by D_m (Equations 5 and 6). The average damage of all measured mementos was 0.132.

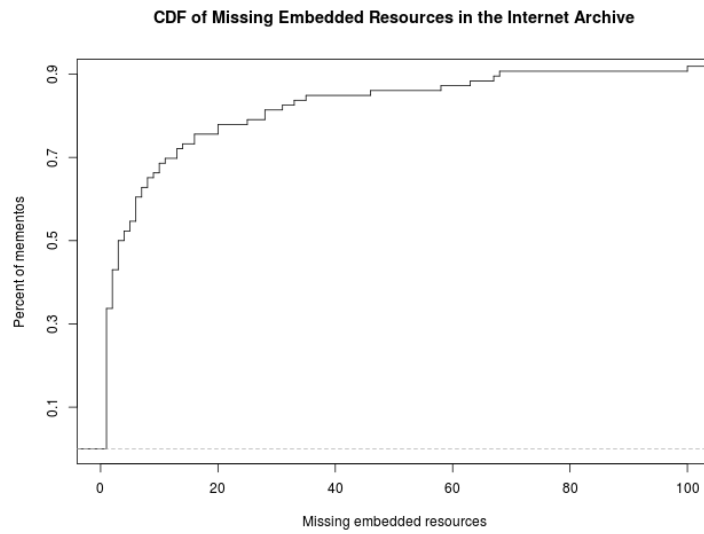
The yearly $\overline{D_m}$ goes from 0.16 in 1998 to 0.13 in 2013. That means the Internet Archive is doing a better job (over time) reducing the total memento damage in its collection. However, the number of missing *important* resources (resources with an importance > 1 due to added weights) is increasing, going from an average of 1.30 important resources per memento in 1997 to 2.38 important resources per memento in 2013 for an average of 2.05 missing per memento. Meanwhile, the number of unimportant missing embedded resources (damage rating weight ≤ 1) per memento is increasing at a lesser rate, going from 1.35 in 1997 to 1.64 in 2013. This suggests that while the Internet Archive is getting better overall at mitigating damage as much as possible, the archive is missing an increasing number of embedded resources deemed important.

The distribution of file types missing per memento (Figure 80) shows that most URI-Ms are missing ≥ 1 embedded resource and that style sheets and JavaScript files are missing at higher rates over time. Missing JavaScript may lead to additional missing files (such as multimedia). Images are missing at varying rates per memento over time.

³We measure the sample mean using \overline{X} , sample standard deviation using s , and sample median using Md . Our results are specific to a sample rather than the population (which would be the entire WWW), and we use the sample symbols to match.

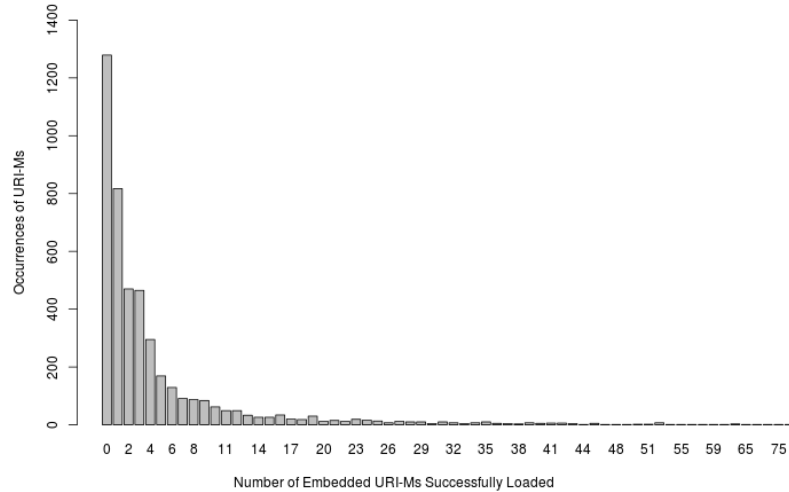


(a) Occurrences of missing embedded resource numbers in the Internet Archive as a histogram.

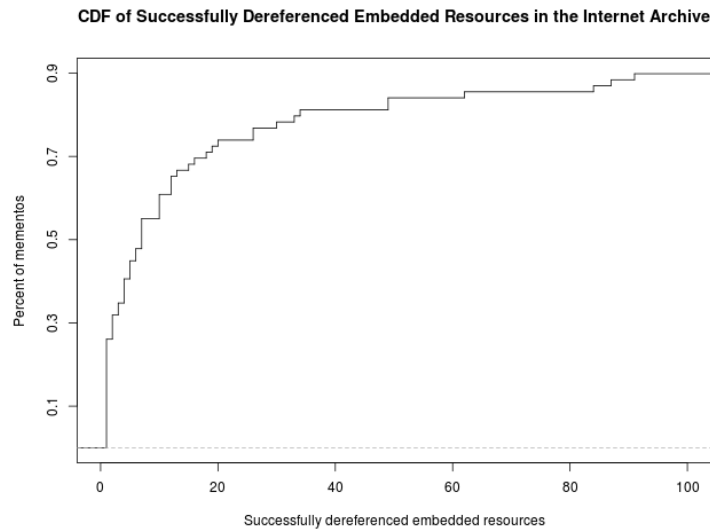


(b) Distribution of missing embedded resources within the collection of Internet Archive mementos as a CDF.

FIG. 78: The distribution of the number of missing embedded resources per URI-M in the Internet Archive. Note that we limited the figures to 100 missing embedded resources.



(a) Occurrences of successfully dereferenced embedded resource numbers in the Internet Archive as a histogram.



(b) Distribution of successfully dereferenced embedded resources within the collection of Internet Archive mementos as a CDF.

FIG. 79: The distribution of the number of successfully dereferenced embedded resources per URI-M in the Internet Archive. Note that we limited the figures to 100 successfully dereferenced embedded resources.

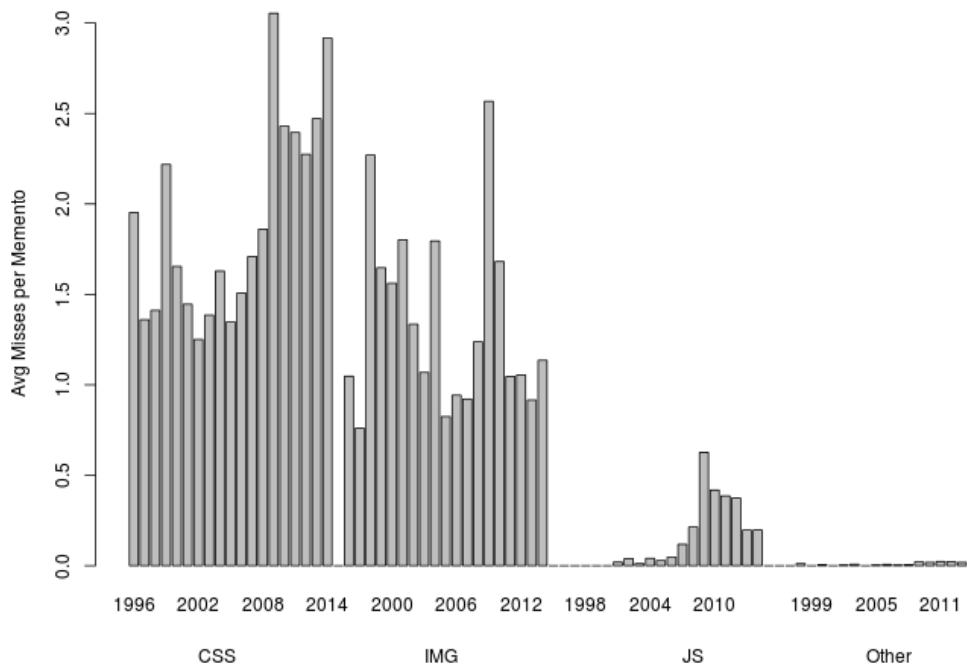


FIG. 80: The number of missed embedded resources per Internet Archive memento per year and MIME type.

6.5.3 MEASURING WEBCITE

In an effort to measure a different type of archive, we used the damage algorithm to determine M_m and D_m of WebCite⁴. WebCite is a page-at-a-time archival service that, as shown in Chapter 5, does not handle JavaScript well and may have more damaged mementos as a result.

WebCite has 992 mementos in the TimeMaps of our collection of 1,861 URI-Rs. The earliest available memento is from 2007, and the latest is from 2014. Only six mementos are available from 2014; therefore, we will focus on 2007-2013 as the target years of investigation due to the limited number of 2014 mementos, as well as to match the period of observation of the Internet Archive. The $\overline{D_m}$ of the WebCite collection over all years is 0.397 ($s = 0.194$), and the $\overline{M_m}$ is 0.176 ($s = 0.0926$). All of the mementos in this collection are missing at least one embedded resource – 100% of the mementos are incomplete.

As shown in Figure 81, the $\overline{D_m}$ in WebCite is increasing over time, going from

⁴<http://www.webcitation.org/>

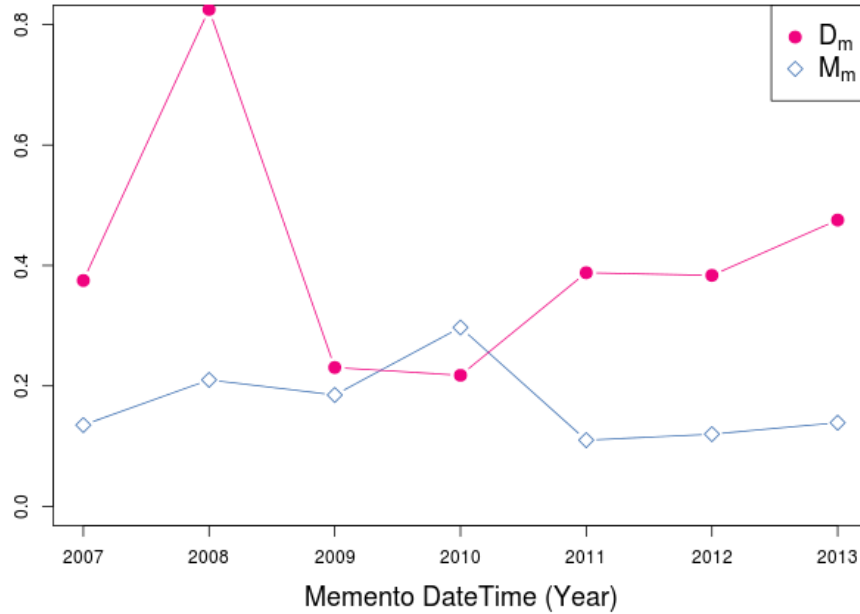


FIG. 81: The average percentage of embedded resources missed per memento per year in WebCite as compared to damage per memento per year ($\overline{D_m}=0.397$, $\overline{M_m}=0.176$).

0.285 in 2007 to 0.442 in 2013. Meanwhile, the average M_m remains steady, going from 0.135 in 2007 to 0.139 in 2013. Only slight variation occurs, peaking at 0.287 in 2010.

Compared to the Internet Archive, WebCite has a higher damage value as well as is missing a larger percentage of embedded resources. Additionally, D_m per memento is higher, indicating that a larger percentage of missing embedded resources are important (3,514 or 41.7%) in WebCite than in the Internet Archive.

WebCite is missing on average 10.1 embedded resources per memento ($s = 8.0$, $Mad = 2.0$). This distribution exhibits a long tail (as did the Internet Archive), with a few mementos missing a large number of embedded resources (maximum is 133). WebCite mementos successfully dereference on average 15.3 embedded resources per memento ($s = 30.7$, $Mad = 4.0$); again, note the long tail (maximum is 154). Across the entire collection, 8,420 of 54,824 (or 15.4%) of the embedded resources were missing in our investigation. We calculate that 56% of mementos are missing 3 or fewer embedded resources, and 74% of mementos are missing 6 or fewer embedded

resources (Figure 82(a)).

The distribution of file types missing per memento (Figure 84) shows that most URI-Ms are missing ≥ 1 embedded image and CSS resources, on average. WebCite has a lower occurrence of missing style sheets, but a higher occurrence of missing images.

As we discussed in Section 5, WebCite has difficulties when encountering JavaScript and embedded iframes. However, its archiving policies provide immediate results as opposed to crawlers that may incur delay between the time a URI-R is added to the frontier and a memento is created. WebCite’s difficulties with JavaScript may contribute to the missing embedded resources in deferred representations.

6.6 IMPACT OF JAVASCRIPT ON DAMAGE

To investigate the impact of JavaScript on archival tools, we set up an experiment to use Heritrix and PhantomJS to crawl the same set of URI-Rs and measure the damage difference between the two sets of mementos. Our goal is to understand how D_m is impacted by JavaScript by comparing mementos archived by a crawler that can execute JavaScript (PhantomJS) and a crawler that does not execute JavaScript (Heritrix).

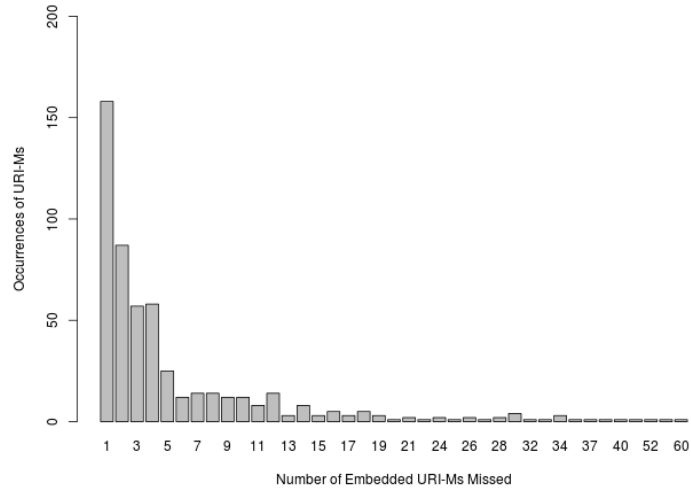
6.6.1 CRAWLING DEFERRED REPRESENTATIONS

Crawlers are unable to discover the resources requested via Ajax (Chapter 4) and are missing embedded resources which ultimately causes the mementos of the crawled resources with deferred representations to be incomplete and have higher D_m .

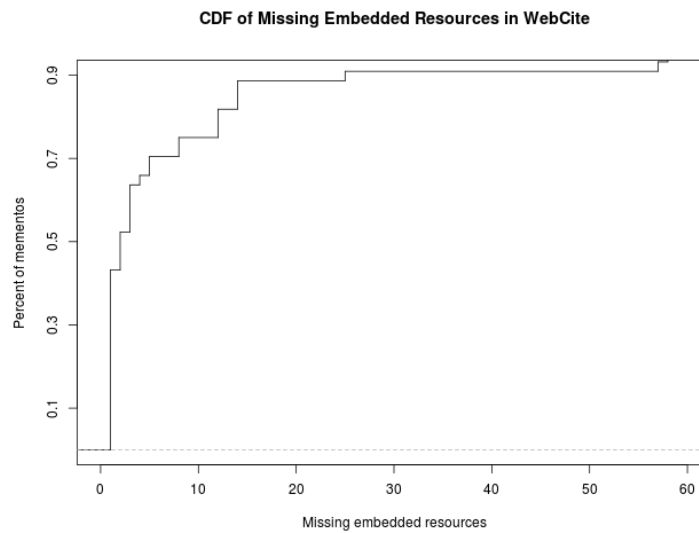
We sampled 50 URI-Rs by randomly generating Bitly URI-Rs and identifying the URI-Rs to which the Bitly URI-Rs redirected. We then manually classified the 50 URI-Rs as having either deferred or nondeferred representations and crawled the set of URIs with Heritrix and PhantomJS.

During the Heritrix crawl, we used the 50 URI-Rs as a set of seed URIs and allowed Heritrix to create their mementos. The final frontier size of this crawl was 1,588 URI-Rs. Using our damage algorithm, we measured the damage of the mementos created by Heritrix and found that $\overline{D_m}=0.148$. Recall that our previous measurement showed the Internet Archive, $\overline{D_m}=0.13$.

To ensure the crawler executes JavaScript and captures JavaScript-dependent resources during the creation of mementos, we then crawled the 50 URI-Rs with

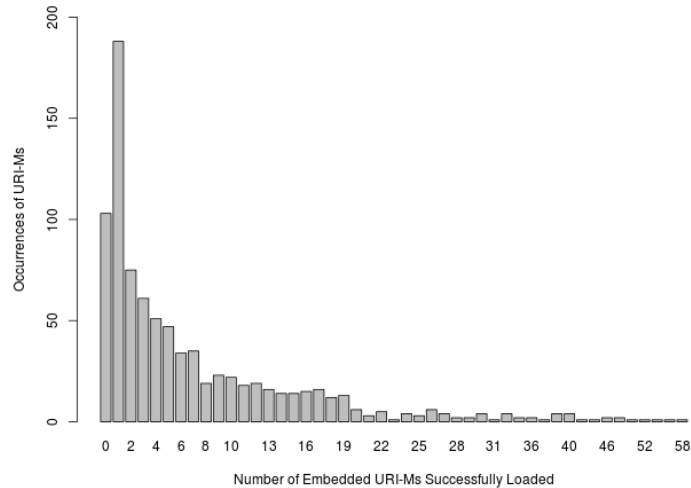


(a) Occurrences of missing embedded resource numbers in WebCite as a histogram.

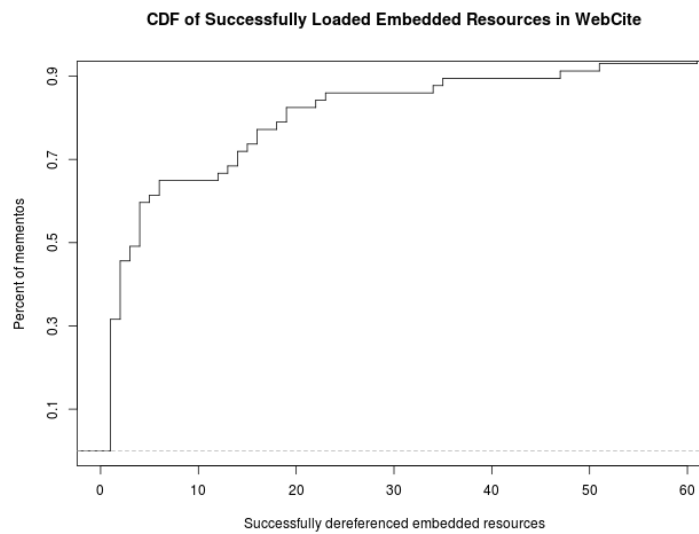


(b) Distribution of missing embedded resources within the collection of WebCite mementos as a CDF.

FIG. 82: The distribution of the number of missing embedded resources per URI-M in WebCite. Note that we limited the figures to 60 missing embedded resources.



(a) Occurrences of successfully dereferenced embedded resource numbers in WebCite as a histogram.



(b) Distribution of successfully dereferenced embedded resources within the collection of WebCite mementos as a CDF.

FIG. 83: The distribution of the number of successfully dereferenced embedded resources per URI-M in WebCite. Note that we limited the figures to 60 successfully dereferenced embedded resources.

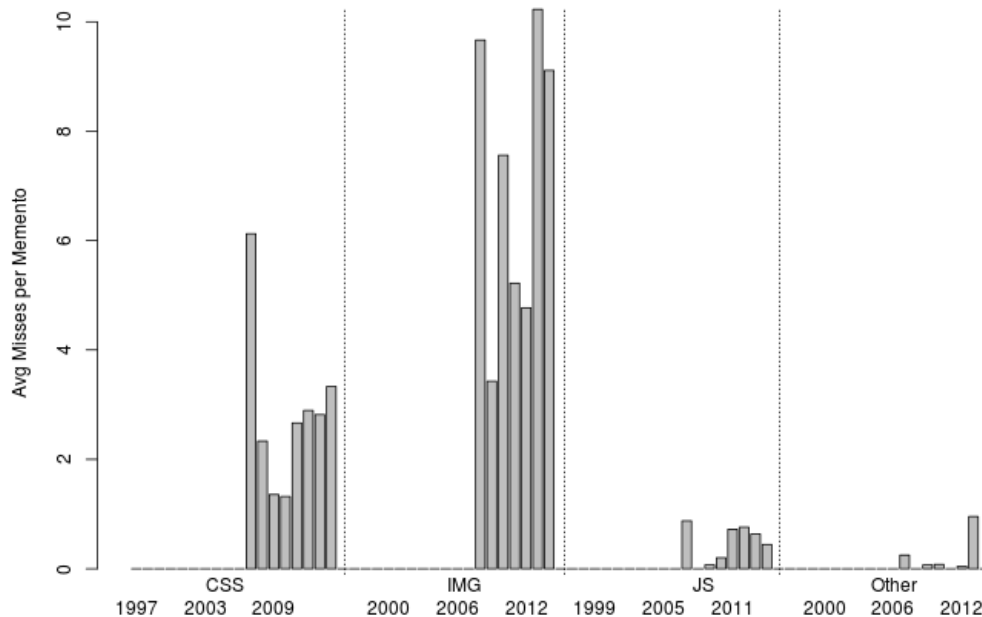


FIG. 84: The number of missed embedded resources per WebCite memento per year and MIME type.

PhantomJS. We recorded the embedded resources needed to create the representation, including those originating from JavaScript. This created a frontier of 3,364 URIs which we used as a seed URI list in Heritrix. We used Heritrix to create the mementos using only the seed URI list, effectively creating mementos using the frontier list of PhantomJS. For this crawl, $\overline{D_m}=0.129$.

PhantomJS provided a 13.5% improvement to the collection damage over Heritrix. This provides further evidence that deferred representations reduce the quality of mementos due to traditional crawlers' inability to execute JavaScript.

Not only does using PhantomJS provide a larger crawl frontier, but the damage rating of the resulting mementos is lower. In short, this initial investigation suggests that using PhantomJS mitigates the impact of JavaScript on resources with deferred representations and results in higher-quality mementos. We discuss this in more detail in Chapter 7.

6.7 MEASURING ARCHIVE.IS

While WebCite does not properly archive deferred representations, Archive.is

creates mementos that limit leakage. To study the impact of Archive.is’s handling of JavaScript on memento quality, we submitted each of our 1,861 URI-Rs to Archive.is for archiving to create mementos of each resource.

When Archive.is creates a memento, it modifies the DOM to remove references to embedded resources that were not available at archive time (i.e., embedded resources that returned a non-200 HTTP response code) [209]. This results in a memento that – if created properly – has no missing embedded resources. Additionally, Archive.is obfuscates the URIs of embedded mementos, preventing a reliable mapping from URI-R to URI-M. For example, the live resource might have an embedded image such as

```

```

and Archive.is will convert the URI-R to the following URI-M:

```

```

Due to these two archival practices, the damage algorithm used in this paper is ineffective for determining memento quality in a directly comparable with to the Internet Archive and WebCite. For this reason, we alter the method of measuring the effectiveness of Archive.is’s archival process.

We initiated the archiving of each URI-R in our collection by Archive.is. We counted the number of embedded resources that were successfully loaded live resources (i.e., returned an HTTP 200 response when their URIs were dereferenced) and compared this number to the number of embedded resources successfully archived by Archive.is, resulting in a delta between live embedded resources and the mementos embedded resources that we will refer to as Δ_m . It is worth noting that the delta between the number of embedded resources in live resources and mementos (Δ_m) is a measure of neither M_m nor D_m , but is instead a mechanism for understanding memento fidelity.

We found that Archive.is has a $\overline{\Delta_m}=19.9$ ($s = 39.2$), meaning that on average, Archive.is did not archive 19.9 embedded resources from the live page due to either its inability to archive the resources, or because Archive.is may have deemed the embedded resources not suitable for archiving⁵. A histogram of all Δ_m measures is provided in Figure 85(a).

We submitted each URI-R in the collection to WebCite and recorded Δ_m for the WebCite mementos in the exact way we measured Δ_m for Archive.is. In this way, we can compare the two page-at-a-time archiver to determine which service creates higher fidelity mementos. WebCite has a $\overline{\Delta_m}=21.6$ ($s = 41.7$), which is higher than the $\overline{\Delta_m}$ of Archive.is. The histogram of the WebCite Δ_m is provided in Figure 85(b). The higher WebCite $\overline{\Delta_m}$ indicates that Archive.is creates higher fidelity mementos than WebCite, likely due to its superior support of JavaScript dependent representations.

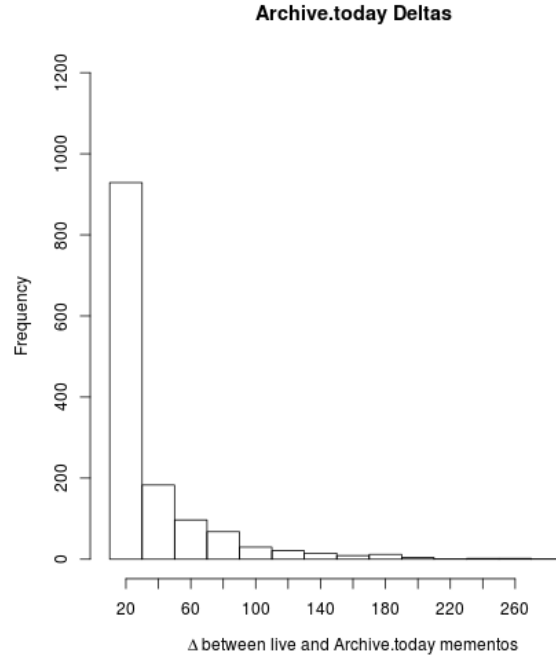
6.8 CONTRIBUTION TO RESEARCH QUESTION 2

To answer Research Question 2: **How do we measure memento quality?**, we constructed an algorithm to automatically assess the damage of mementos based on their missing embedded resources. We designed and executed an experiment that evaluated the algorithm, and showed the historical patterns of D_m over time.

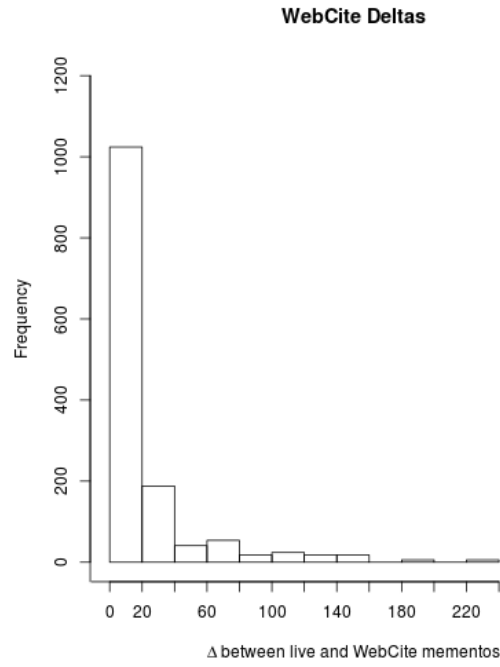
We demonstrated that Web users (as represented by Mechanical Turk Workers) can correctly identify original mementos 81% of the time when presented with an original and a manually damaged pair of mementos (m_0 vs m_1). After randomly selecting 100 URI-Ms from the Internet Archive TimeMaps of 1,861 URI-Rs, we show that turkers' assessment of damage does not match that of M_m ; in fact, their perception of damage more closely aligns with a random selection than with M_m .

To provide a damage metric closer to the perception of Web users, we proposed D_m , a damage calculation algorithm that estimates embedded resource importance to determine the perceived damage of mementos. Using turker evaluations, we showed that D_m aligns with turker perception 32% of the time when considering all ΔD_m values – an improvement of 17% over M_m . If we limit $\Delta D_m > 0.30$, we achieve an agreement of 71%, an improvement of 51% over M_m . We show that the performance of D_m is closer to that of the m_0 vs m_1 test than both M_m and a random selection.

⁵Archive.is lists the resources it saves and does not save in its FAQ page at <http://archive.is/faq.html>.



(a) Histogram of the memento vs live resource Δ_m in Archive.is.



(b) Histogram of the memento vs live resource Δ_m in WebCite.

FIG. 85: The Δ_m measurements of Archive.is and WebCite indicate that Archive.is creates higher fidelity mementos than WebCite.

We used D_m to measure the damage of mementos in the Internet Archive by measuring $\overline{D_m}$ of 1,861 URI-Rs. The average damage of the Internet Archive collection is 0.13 per memento and is missing 15% of its embedded resources. Mementos are missing 2.05 important resources on average. The Internet Archive has gotten better at mitigating damage over time, reducing D_m from 0.16 (1998) to 0.13 (2013).

Page-at-a-time archival services perform differently than the Internet Archive’s Heritrix crawler⁶. We measured mementos of our collection in WebCite, finding that the average damage of the collection is 0.397 per memento and is missing 18% of its embedded resources. Mementos are missing 10.1 resources on average. Even though damage in the Internet Archive is improving, the damage in WebCite is getting worse, increasing $\overline{D_m}$ from 0.375 (2007) to 0.475 (2013).

We also demonstrate that deferred representations have a detrimental impact on D_m and M_m . By using a crawl strategy in which JavaScript is executed during the crawl, damage in the resulting mementos can be improved by 13.5%.

We also measured the damage of mementos in WebCite, and demonstrated that the damage in the Internet Archive ($\overline{D_m}=0.128$) is less than that in WebCite ($\overline{D_m}=0.397$). We know from Chapter 5 that WebCite does not archive JavaScript-dependent representations effectively. We measured Archive.is to determine the fidelity of an archival service that makes an effort to use headless browsing to capture JavaScript dependent representations. We found that Archive.is had a Δ_m of 19.9 embedded resources between mementos and live resources, while WebCite had a Δ_m of 21.6. This shows that Archive.is provides a higher fidelity memento than WebCite. We also show that deferred representations are of lower quality than their counterparts that avoid JavaScript.

With D_m , archival services can automatically evaluate their performance and the quality of their mementos. The archives could measure a selection of mementos (either randomly sampled or by identifying those missing a proportion of embedded resources, such as $\Delta M_m > 0.30$) for damage to determine whether or not they have been satisfactorily archived. That is, with this algorithm, the archives can provide the greatest damage improvement through targeted repair efforts (e.g., identify mementos that require additional attention to ensure proper archiving). Archives can also use historical damage ratings of a URI-R to identify memento improvements or changes.

⁶The Internet Archive’s Save Page Now service operates as a page-at-a-time archival service [250], but is not considered for the purposes of this study.

Even though we have answered Research Question 2, this is a preliminary investigation of memento damage and archival quality. We have shown that percentage of embedded resources missing is not an accurate representation of damage and have proposed a more accurate metric. We will investigate the cumulative damage rating over time. For example, a logo that never changes over a five year period could have increased importance due to its use over multiple mementos. We plan to also measure the damage improvement of mementos if embedded resources are retroactively captured and included in past mementos. This cumulative damage improvement can help identify embedded resources that should be targeted by archives.

CHAPTER 7

A TWO-TIERED APPROACH FOR CRAWLING DEFERRED REPRESENTATIONS

Research Question 3: **How can we crawl, archive, and play back deferred representations?** is a vast area of investigation in which we present a framework for crawling deferred representations that incorporates PhantomJS into the legacy archival workflow with Heritrix. We showed in Chapter 6 that PhantomJS helps create higher quality mementos, and in this chapter, we measure the impact of using PhantomJS in an archival workflow. We measure the performance (i.e., crawl time and frontier size) of the crawler components, expected performance of our two-tiered crawling approach, present a classifier to recognize deferred representations, and describe how the framework can discover and archive descendants of deferred representations.

We investigate the performance impact of crawling deferred representations as part of our improved framework for archiving deferred representations. We measure the expected increase in frontier size and wall-clock time required to crawl resources, and we investigate a way to recognize deferred representations to optimize crawler performance (by crawl time as well as discovered frontier) using a two-tiered crawling approach that combines PhantomJS and Heritrix [57]. Our efforts measure the crawling tradeoff between traditional archival tools and tools that can better archive JavaScript with headless browsing – a tradeoff that was anecdotally understood but not yet been measured.

We use 10,000 seed URI-Rs¹ to explore the impact of including PhantomJS in the crawling process by comparing the performance of wget (the baseline), PhantomJS, and Heritrix. Heritrix crawled 2.07 URIs per second, 12.15 times faster than PhantomJS and 2.4 times faster than wget. However, PhantomJS discovered 531,484 URIs, 1.75 times more than Heritrix and 4.11 times more than wget. To take advantage of the performance benefits of Heritrix and the URI discovery of PhantomJS, we recommend a tiered crawling strategy in which a classifier predicts whether or not

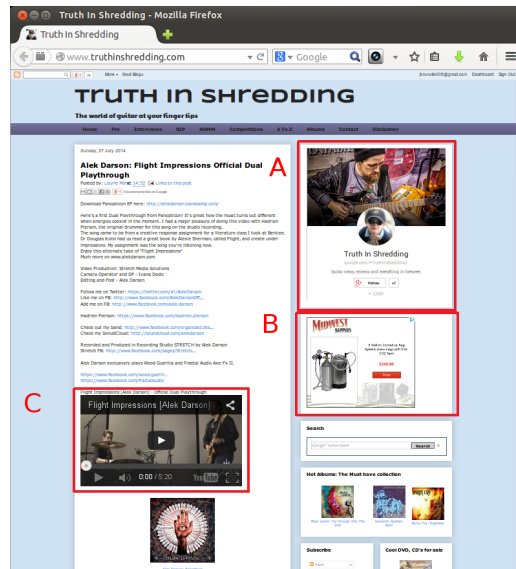
¹Dataset available at <https://github.com/jbrunelle/DataSets/blob/master/10kuris.txt>.

a representation will be deferred, and only resources with deferred representations are crawled with PhantomJS whereas resources without deferred representations are crawled with Heritrix. We show that this approach is 5.2 times faster than using only PhantomJS and creates a frontier 1.8 times larger than using only Heritrix.

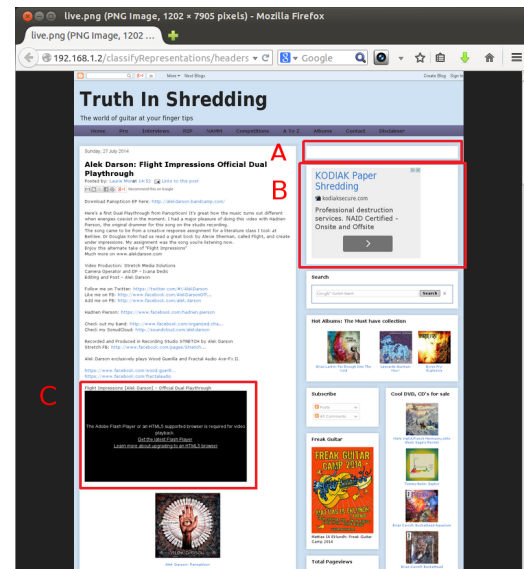
Following the investigation of performance tradeoffs, we study the extent of interactivity in deferred representations [58], construct interaction trees of representations, and propose a model for representing interactions. We study the amount of new and unarchived embedded resources that can be uncovered through automated interaction with a representation. From these investigations, we propose a method of discovering and crawling deferred representations and their descendants that are only reachable through client-side events. We adapt Dincturk et al.’s Hypercube model [79, 30, 80] to construct a model for archiving descendants, and we measure the number of descendants and requisite embedded resources discovered in a proof-of-concept crawl. Our two-tiered crawling approach identified 38.5 descendants per seed URI crawled, 70.9% of which are reached through an onclick event, and adds 15.6 times more embedded resources than Heritrix to the crawl frontier. However, crawling descendants is 38.9 times slower than simply using Heritrix. We demonstrate that there are only two levels of descendants in our dataset, indicating that the interaction trees of deferred representations are much shallower than we anticipated. We conclude with proposed crawl policies, and an analysis of the expected storage requirements for archiving descendants.

7.1 MOTIVATING EXAMPLES

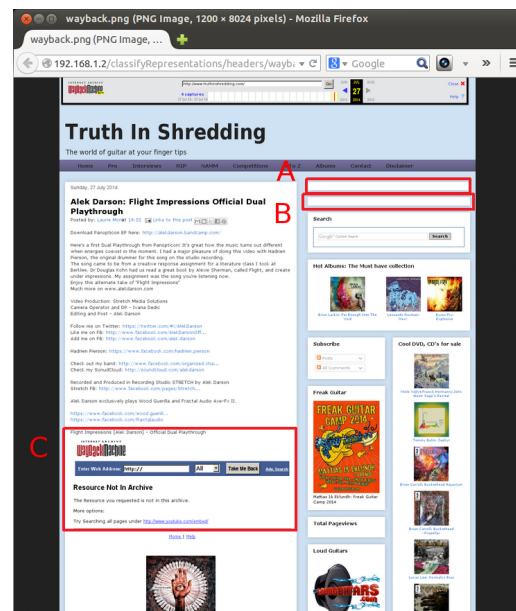
We present an example of the challenge of archiving deferred representations and the difference between mementos of URI-R <http://www.truthinshredding.com/> captured by traditional tools versus PhantomJS in Figure 86. We took a PNG snapshot of the live Web resource (Figure 86(a)), the resource as loaded by PhantomJS (Figure 86(b)), and the memento that is the result of crawling the resource with Heritrix and viewing the memento in a local instantiation of the Wayback Machine (Figure 86(c)). The title of the page “Truth in Shredding” appears in a different font in Figure 86(a) than in Figures 86(b) and 86(c). This is not due to a missing style sheet, but rather an incompatibility of the font for the headless browser. Figure 86(a) was rendered in the Mozilla Firefox browser, whereas Figures 86(b) and 86(c) were rendered using the PhantomJS headless browser.



(a) The live-Web resource at URI-R <http://www.truthinshredding.com/> loads resources A, B, and C via JavaScript.



(b) Using PhantomJS, the advertisement (B) and video (C) are found but the account frame (A) is missed.



(c) Using Heritrix, the embedded resources A, B, and C are missed.

FIG. 86: Neither archival tool captures all embedded resources, but PhantomJS discovers the URI-Rs of two out of three embedded resources dependent upon JavaScript (B, C) while Heritrix misses them.

The live Web resource loads embedded resources (annotated as A, B, and C) via JavaScript. Embedded Resource A is an HTML page loaded into an iframe. $URI-R_A$ is

```
https://apis.google.com/u/0/_/widget/render/page?useapi=1
&rel=publisher&href=%2F%2Fplus.google.com%2F11074366589
0542265089&width=430&hl=en-GB&origin=http%3A%2F%2Fwww.t
ruthinshredding.com&gsrc=3p&ic=1&jsh=m%3B%2F_%2Fscs
%2Fapps-static...
```

The page loaded into the iframe uses JavaScript to pull the profile image into the page from $URI-R_{A_1}$

```
https://apis.google.com/_/scs/apps-static/_/ss/k=oz.widget.
-ynlzpp4csh.L.W.0/m=bdg/am=AAAAAJAwAA4/d=1/rs=AItrSTNrapsz0
r4y_tKMA1hZh6JM-g1haQ
```

Embedded Resource B is an advertisement that uses the JavaScript at $URI-R_{B_1}$

```
http://pagead2.googlesyndication.com/pagead/show_ads.js
```

to pull in ads to the page. Embedded Resource C is a YouTube video that is embedded in the page using the following HTML for an iframe:

```
<iframe allowfullscreen="" frameborder="0" height="281"
src="//www.youtube.com/embed/QyLl4Fd4cGA?rel=0" width="500">
</iframe>.
```

PhantomJS does not load Embedded Resource A because the host resource completes loading before the page embedded in the iframe can finish loading. PhantomJS stops recording embedded URIs and monitoring the representation after a page has completed loading (i.e., on the `onload` event), and Embedded Resource A executes its JavaScript to load the profile picture after the main representation has completed the page load². PhantomJS does discover the advertisement (Embedded Resource B) and the YouTube video (Embedded Resource C). Even though the headless browser

²A PhantomJS script can be written in such a way that this race-condition can be avoided using longer timeouts or client-side event detection, but this functionality is outside the scope of this discussion.

used by PhantomJS does not have the plugin necessary to display the video, and therefore it is missing from the PNG, the URI-R is still discovered by PhantomJS.

Heritrix fails to identify the URI-Rs for the Embedded Resources A, B, and C. When the memento created by Heritrix is loaded by the Wayback Machine, Embedded Resources A, B, and C are missing. This behavior is due to Heritrix's inability to discover the URI-Rs for these resources during the crawl. However, when viewing the memento through the Wayback Machine, the JavaScript responsible for loading the embedded resources is executed resulting in either a zombie resource or an HTTP 404 response for the embedded resources.

Heritrix's inability to discover the URI-Rs of the embedded resources could be mitigated by utilizing PhantomJS during the crawl. However, this raises many questions, most notably: How much slower will the crawl time be? How many additional embedded resources could a two-tiered crawling approach uncover and potentially need to store? Can we optimize the two-tiered crawling approach based on the detection of deferred representations? In short, our investigation into these questions will assess the feasibility of equipping Heritrix with PhantomJS to create a two-tiered crawling approach to mitigate the impact of JavaScript on missed embedded resources in the archives.

7.2 A FRAMEWORK FOR TWO-TIERED CRAWLING

To this point, we have presented our body of work regarding the impact of JavaScript on the archives. We propose a *two-tiered crawling approach* designed to mitigate these challenges, and measure the impact, performance trade-offs of crawling approaches, and the benefits of a two-tiered crawling approach in the rest of this chapter. In this chapter, we summarize and measure the framework we have constructed as a result of our body of work.

7.2.1 SINGLE-TIER CRAWLING FRAMEWORK

In the legacy model of Web archiving (that in this section we refer to as the *single-tier framework*), the archival workflow matched the way users and browsers interacted with the Web before JavaScript and client-side technologies rose to prominence. The crawler begins with a seed list as the initial frontier – URIs U_1 , U_2 , U_3 in Figure 87. Beginning with U_1 (the first URI in the frontier list), the crawler will simply dereference the URI (Step 1), receive the representation (at this point, we do not know

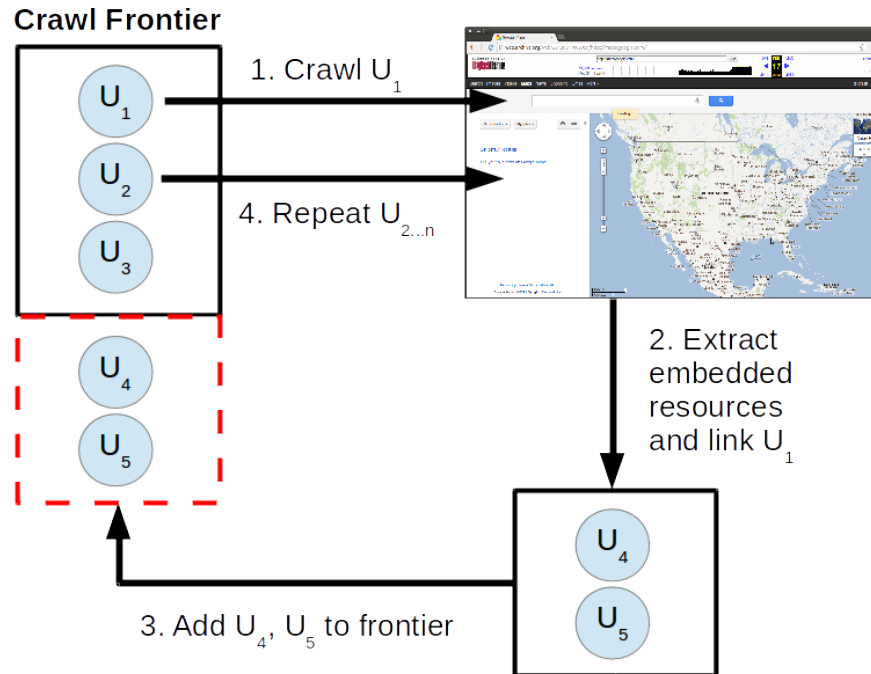


FIG. 87: The current archival work flow was designed to archive pre-JavaScript nondeferred representations.

if the representation is deferred or if it has descendants), archive the representation, extract the URI-Rs of embedded resources and links (U_4 , U_5) (Step 2), and add the extracted URI-Rs to the frontier (Step 3). The crawler repeats the process (e.g., the next URI, U_2 , will be popped off the top of the frontier) until the frontier is exhausted.

While this method was effective when representations did not change on the client, and URIs for embedded resources were delivered from a server embedded in the representation at the time of dereferencing, JavaScript’s ability to construct URIs of embedded resources and make additional requests for embedded resources as a result of the page rendering or user interaction has made the archival workflow in Figure 87 ineffective for the current Web. Our two-tiered crawling approach is a proposed modification to the *single-tier framework* that incorporates a classifier and headless browser to better archive deferred representations and their descendants.

7.2.2 TWO-TIERED CRAWLING APPROACH

We propose a two-tiered approach for crawling deferred representations. The

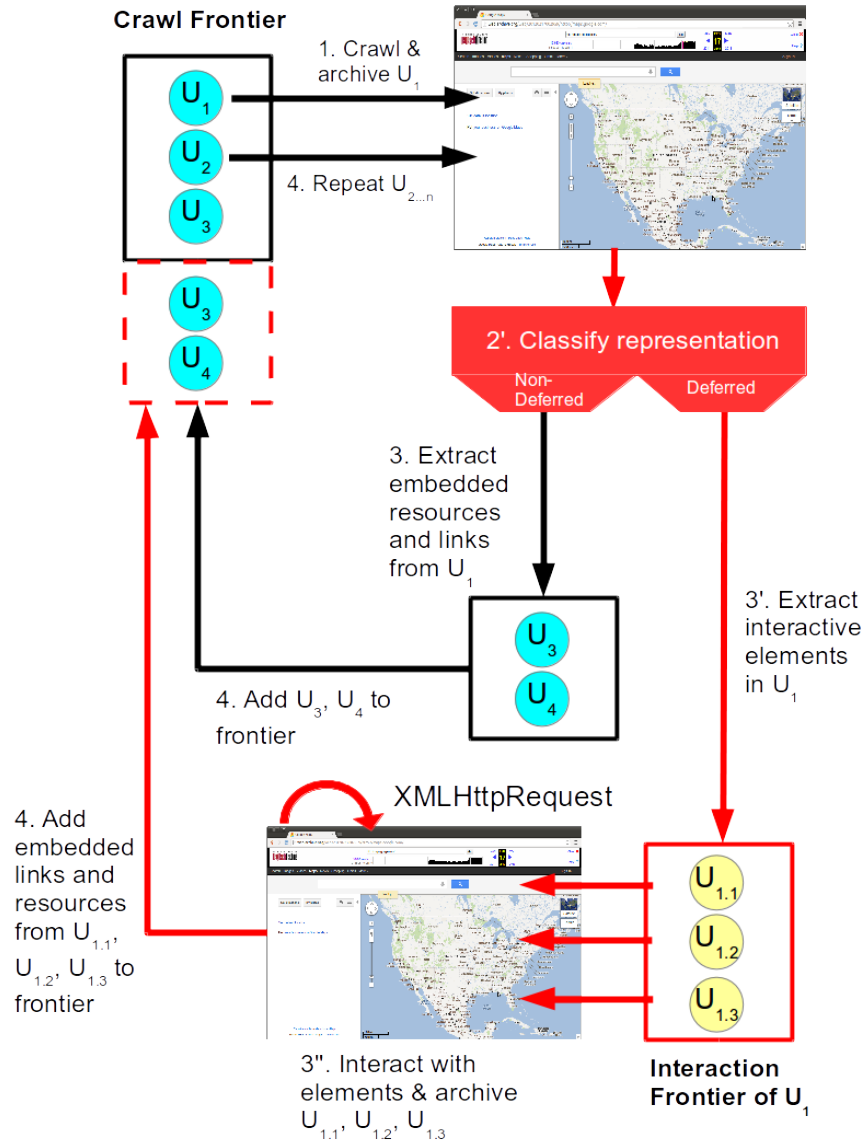


FIG. 88: The proposed adaptation to the archival work flow handles deferred representations and descendants.

processes added to the original workflow are in red in Figure 88. In our two-tiered crawling approach, we begin similar to the *single-tier framework* with a seed list of URIs (U_1, U_2, U_3), and the first URI on the frontier is selected and crawled (Step 1). Once the representation is archived, we reach the first change to the archival workflow from the *single-tier framework*. We classify the representation as either deferred or nondeferred (Step 2'), the process for which is discussed in more detail in Section 7.4.

Nondeferred representations fit the *single-tier framework* of archiving and can be properly archived using the single-tier workflow and proceeds as normal through Steps 3 and 4. If the representation is classified as deferred, it will not be properly archived using the *single-tier framework* workflow.

Deferred representations use a specialized sub-workflow beginning with Step 3', in which PhantomJS or another headless browsing utility dereferences the URI-R, identifies the interactive DOM elements, and an interaction frontier ($U_{1.1}, U_{1.2}, U_{1.3}$) is built using the *Max Coverage* policy (Section 7.10.3). Each interaction associated with the interactive elements on the interaction frontier is executed on the client to discover the embedded resources and descendants on the client (Step 3''). From these interactions and descendants, the URIs of any embedded resources are added to the crawl frontier.

The two-tiered crawling approach is slower – in wall-clock time – per URI-R with a deferred representation than the *single-tier framework* (Section 7.3.1), and requires additional storage to handle the increased coverage of the Web (Section 7.12). However, using the two-tiered crawling approach, we can uncover embedded resources from deferred representations that the single-tier framework cannot (Section 7.3.2), increasing the quality and completeness of the archive.

7.3 COMPARING CRAWLS

Before discussing the performance of our two-tiered crawling approach, we designed an experiment to measure the performance differences between a command-line archival tool (wget), a traditional crawler (Heritrix), and a headless browser client (PhantomJS). We use wget as a baseline to which we compare the performance of PhantomJS and Heritrix, and compare the performance of each tool to understand their operation individually. We use their individual performances to

recommend crawl policies and maximizing the performance and coverage of the two-tiered crawling approach.

We constructed a 10,000 URI-R dataset by randomly generating a Bitly URI and extracting its redirection target. We split the 10,000 URI dataset into 20 sets of 500 seed URI-Rs and used `wget`, `Heritrix`, and `PhantomJS` to crawl in parallel each set of seed URI-Rs and their embedded resources. We repeated each crawl ten times to establish an average performance, resulting in ten different crawls of the 10,000 URI dataset (executing the crawl one of the 500-URI sets at a time³) with each of `wget`, `Heritrix`, and `PhantomJS`. We measured the increase in frontier size and the time taken per URI to crawl the resource.

Although `Heritrix` provides a user interface that identifies the crawl frontier size, `PhantomJS` and `wget` do not. We calculate the frontier size of `PhantomJS` by counting the number of embedded resources that `PhantomJS` requests when rendering the representation. We calculate the frontier size of `wget` by executing a command⁴ that records the HTTP GET requests issued by `wget` during the process of mirroring a Web resource and its embedded resources. We consider the frontier size to be the total number of resources and embedded resources that `wget` attempts to download.

We began a crawl of the same 500 URI-Rs using `wget`, `Heritrix`, and `PhantomJS` simultaneously to mitigate the impact of changing resources. For example, if the representation changes (such as includes new embedded resources) in between the times `wget`, `PhantomJS`, and `Heritrix` perform their crawls, the number or representations of embedded resources may change and therefore the representation influenced the crawl time, not the crawler itself.

We crawled live Web resources because mementos inherit the limitations of the crawler used to create them. Depending on crawl policies, a memento may be incomplete and different than the live resource. The `robots.txt` protocol, breadth- versus depth-first crawling, or the inability to crawl certain representations (like deferred representations) can all influence the mementos created during a crawl.

³We execute in 500 URI-R chunks to limit the chance that the resources change in between the tools' crawls of the same URI-R.

⁴We executed the command `wget -T 40 -o outfile -p -O headerFile [URI-R]` which downloads the target URI-R and all embedded resources and dumps the HTTP traffic to a `headerFile`.

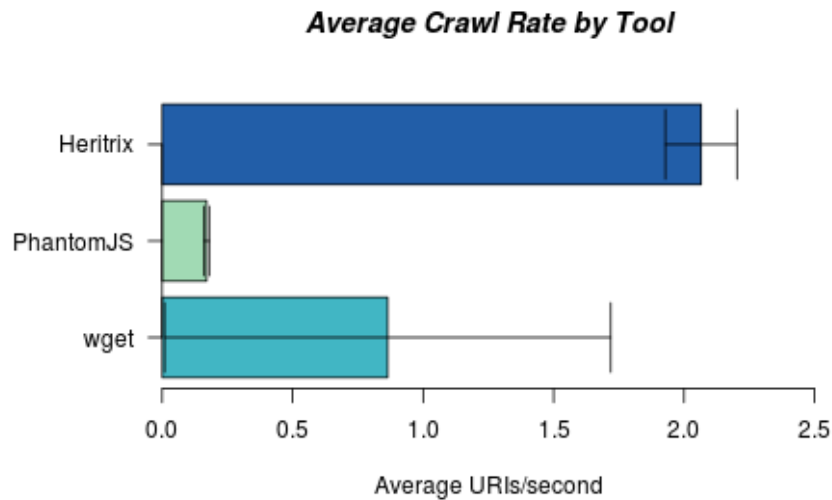


FIG. 89: Heritrix crawls 12.13 times faster than PhantomJS. The error lines indicate the standard deviation across all ten runs.

7.3.1 CRAWL TIME BY URI

To better understand how the crawl times of wget, PhantomJS, and Heritrix differ, we determined the time needed to crawl a URI-R. Heritrix has a browser-based user interface that provides the URIs/second metric. We collected this metric from the Web interface for each crawl. We used Unix system times to calculate the crawl time for each PhantomJS and wget crawl by determining the start and stop times for dereferencing each resource and its embedded resources. We compare the wget, PhantomJS, and Heritrix crawl times per URI in Figure 89. Heritrix outperforms PhantomJS, crawling $\overline{URIs/sec} = 2.065$ ($s = 0.137$) while PhantomJS crawls $\overline{URIs/sec} = 0.170$ ($s = 0.001$) and wget crawls $\overline{URIs/sec} = 0.864$ ($s = 0.855$). Heritrix crawls, on average, 12.13 times faster than PhantomJS and 2.39 times faster than wget.

The performance difference comes from two aspects of the crawl. First, Heritrix executes crawls in parallel with multiple threads being managed by the Heritrix software; this is not possible with PhantomJS on a single core machine since PhantomJS requires access to a headless browser and its associated JavaScript engine, and parallelization will result in process and threading conflicts. Second, Heritrix does not execute the client-side JavaScript and only adds URIs that are extracted from the

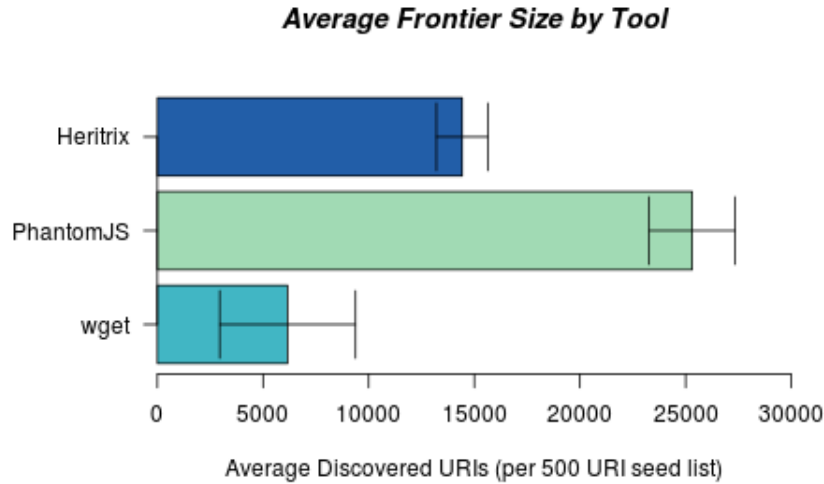


FIG. 90: PhantomJS discovers 1.75 times more embedded resources than Heritrix and 4.11 times more resources than wget. The averages and error lines indicate the standard deviation across all ten runs.

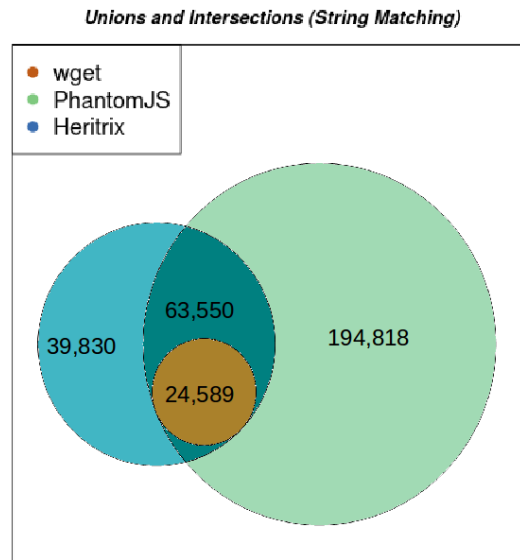
DOM, embedded CSS, and other resources to its frontier; this results in a smaller frontier.

7.3.2 URI DISCOVERY AND FRONTIER SIZE

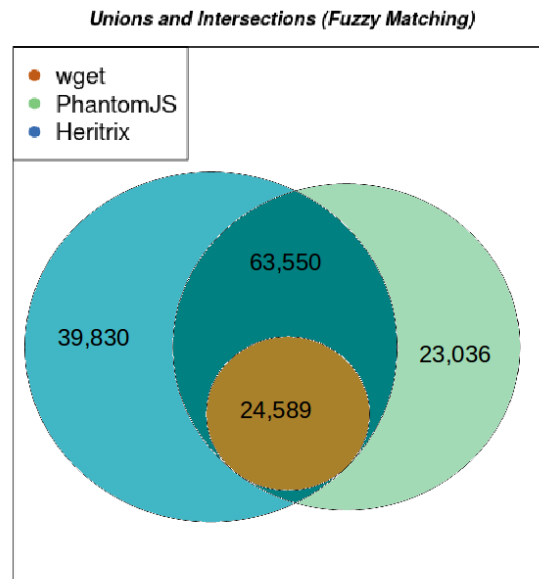
Crawler	Crawl time		Frontier Size	
	$\overline{URIs/sec}$	s	\overline{URIs}	s
wget	0.864	0.855	129,443	3,213.65
Heritrix	2.065	0.137	302,961	1,219.82
PhantomJS	0.170	0.001	531,484	2,036.92

TABLE 18: Performance of wget, Heritrix, and PhantomJS for crawls of 10,000 seed URIs.

As shown in Figures 89 and 90 and summarized in Table 18, we found that PhantomJS discovered and added 1.75 times more URI-Rs to its frontier than Heritrix, and 4.11 times more URI-Rs than wget. Per URI-R, PhantomJS loads 19.7 more embedded resources than Heritrix and 32.4 more embedded resources than wget. The superior frontier size for PhantomJS is attributed to its ability to execute JavaScript and discover URIs constructed and requested through client-side scripts.

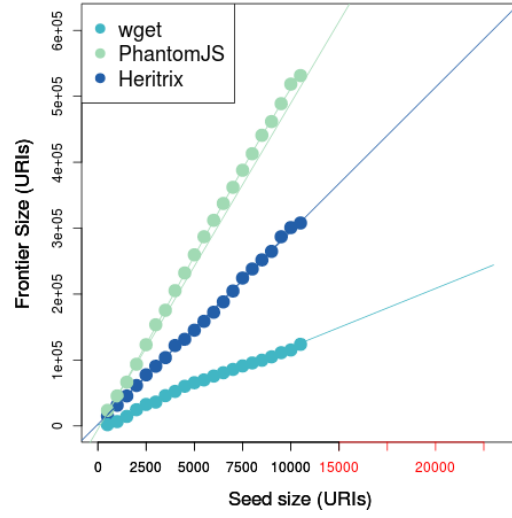


(a) A portion of the Heritrix, PhantomJS, and wget frontiers overlap, and PhantomJS and Heritrix identify URIs that the others do not.

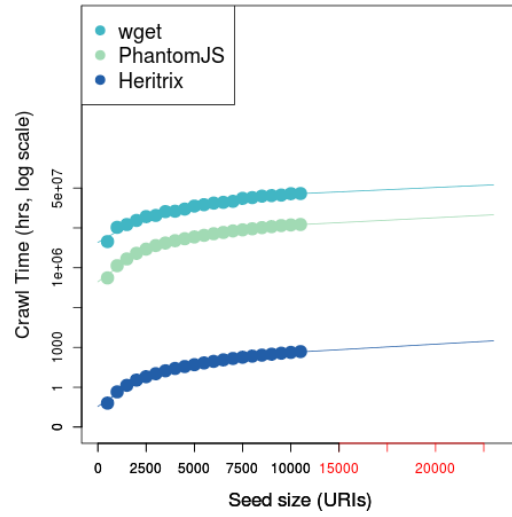


(b) The frontier of URI-Rs unique to PhantomJS shrinks when only considering the host and path aspects (Base Policy for matching) of the URI-R.

FIG. 91: Heritrix, PhantomJS, and wget frontiers as an Euler Diagram. The overlap changes depending on how duplicate URIs are identified.



(a) Frontier size grows linearly with seed size.



(b) Crawl speed is dependent upon frontier size.

FIG. 92: Frontier size and crawl speed are dependent upon seed size, with PhantomJS creating a larger crawl frontier but running more slowly than wget and Heritrix. Note that the plotted dots are measured performance, while the lines are predictions.

However, raw frontier size is not the only performance metric for assessing the quality of the frontier. Rather than rely on naïve cardinality of the frontier, we performed a string-matching de-duplication (that is, removing duplicate URIs) to determine the true frontier size⁵. PhantomJS and Heritrix discover some of the same URIs, whereas PhantomJS discovers URIs that Heritrix does not and Heritrix discovers URIs that PhantomJS does not. We measured the union and intersection of the Heritrix and PhantomJS frontiers.

As shown in Figure 91(a), per 10,000 URI-R crawl Heritrix finds 39,830 URI-Rs missed by PhantomJS on average, while PhantomJS finds 194,818 URI-Rs missed by Heritrix per crawl on average. PhantomJS and Heritrix find 63,550 URI-Rs common between the two crawlers. The wget crawl resulted in a frontier of 24,589 URI-Rs that was a proper subset of both the Heritrix and PhantomJS frontiers.

This analysis shows that PhantomJS finds 19.70 more embedded resources per URI than Heritrix (Figure 92(a)). Heritrix runs 12.13 times faster than PhantomJS (Figure 92(b)).

7.3.3 FRONTIER PROPERTIES AND DEDUPLICATION

During the PhantomJS crawls, we observed that PhantomJS discovers session-specific URI-Rs that Heritrix misses and Heritrix discovers higher level domains that PhantomJS misses, presumably from Heritrix’s inspection of embedded JavaScript. For example:

```
http://dg.specificclick.net/?y=3&t=h&u=http%3A%2F%2F
Fmisscellania.blogspot.com%2Fstorage%2F
Twitter-2.png%3F__SQUARESPACE_CACHEVERSION
%3D1230999100588&r=
```

from PhantomJS versus

```
http://dg.specificclick.net/
```

from Heritrix. The uniquely Heritrix URI-Rs are potentially the base of a URI (or TLD) to be further built by JavaScript. This URI-R is neither used by the representation nor the JavaScript in which the URI-R is found and in this case, the

⁵Heritrix does frontier deduplication automatically, so we had to implement this feature in PhantomJS and wget.

URI does not return a representation. Because PhantomJS only discovers URIs for which the client issues HTTP requests and `http://dg.specificclick.net/` is never requested, this URI-R is not discovered by PhantomJS.

To determine the nature of the differences between the Heritrix and PhantomJS frontiers, we analyzed the union and intersection between the URI-Rs in the frontiers using different matching policies (Figure 91(b)).

During a crawl of 500 URI-Rs by PhantomJS, 19,022 URI-Rs were added to the frontier for a total of 19,522 URI-Rs in the frontier. We also captured the content body (the returned representation received when dereferencing a URI-R from the frontier) and recorded its MD5 hash value. We used the hash value to identify duplicate returned representations during the crawl (i.e., we used the signature of the representation rather than rely on string matching the URI-R).

To determine duplication between URIs, we used five matching policies to deduplicate the URI-Rs within the frontier (Table 19). In other words, we identify cases in which the URIs are different but the content is the same, similar to the methods used by Sigurðsson [265, 266].

The *No Trim* policy uses strict string matching of the URI-Rs to detect duplicates. The *Base Trim* policy trims all parameters from the URI. For example, the URI

```
http://example.com/folder/index.html?param=value
```

would be trimmed to

```
http://example.com/folder/index.html
```

The *Origin Trim* policy eliminates all parameters and associated values that reference a referring source, such as `origin`, `callback`, `domain`, or `referrer`. These parameters are often associated with a value including the TLD of the referring page. Frequent implementers include Google Analytics or ad services.

The *Session Trim* policy eliminates all parameters and their associated values that reference a session. For example, the parameters such as `session`, `sessionid`, `token_id`, etc. are all removed from the URI-R before matching. These parameters are often used by ad services or streaming media services to identify browsing sessions for tracking and revenue generation purposes.

The *HTTP Trim* policy removes all parameters with values that mention a URI. Ad services, JavaScript files, and other statistics tracking services frequently utilize these parameters. For example, the URI

Trim Type	URI Duplicates	URI and Entity Duplicates	Accuracy
No Trim	6,469	4,684	0.68
Other Trim	6,933	2,810	0.62
Base Trim	7,078	4,749	0.68
Origin Trim	10,359	5,191	0.56
Session Trim	8,159	4,921	0.64
HTTP Trim	7,315	4,868	0.67

TABLE 19: Detected duplicate URIs, entity bodies, and the overlap between the two using the five URI string trimming policies.

`http://example.com/folder/index.html?param=value&httpParam=http://
www.test.com/`

is trimmed to

`http://example.com/folder/index.html?param=value`

We used the five trimming policies to detect duplicates in the frontiers constructed by PhantomJS in one of the crawls of 500 URI-Rs. At the end of the crawl, PhantomJS had a frontier of 19,522 URI-Rs. We determined that this set of entity bodies had 8,859 duplicate entity bodies. With the trimmed URI and the MD5 hash of the entity, we can compare the identifiers and the returned entities for duplication.

For each of the 19,522 URIs in the frontier and their associated entity hash values, we determined the trimmed URI string, the duplications of URIs in the frontier, and the number of duplicate URIs that also had a duplicate entity body (Table 19). We calculated the accuracy of each trim policy using the number of URIs with the same entity hash and URI as a true positive (TP), the number of URIs that had neither a duplicate URI nor a duplicate entity body as a true negative (TN), and the set of all positives and negatives ($P + N$) as the total number of URIs (19,522).

The *Base Trim* and *No Trim* policies had identical accuracy ratings (0.68). The *Base Trim* policy identified the most URI duplicates, and is used to determine the overlap between the Heritrix and PhantomJS frontiers.

Using the *Base Trim* policy to only consider the host and path (e.g., `http://pubads.g.doubleclick.net/gampad/ads`) of the PhantomJS and Heritrix frontiers,

PhantomJS identifies 376,578 URI-Rs added to the frontier, 199,761 (55%) of which are duplicates of the discovered URIs. If we consider only the host and path of the PhantomJS URIs, the Euler Diagram of PhantomJS and Heritrix frontiers is more evenly matched (Figure 91(b)).

7.3.4 DEFERRED VS. NONDEFERRED CRAWLS

To isolate the impact of resources with deferred representations on crawl performance, we manually classified 200 URI-Rs from our set of 10,000 URI-Rs as having deferred representations⁶ and another 200 as having nondeferred representations. We crawled each of the deferred and nondeferred sets of URI-Rs with PhantomJS and Heritrix.

During the crawl of the nondeferred set, PhantomJS crawled 0.255 URIs/sec while Heritrix crawled 1.34 URIs/sec, 5.25 times faster than PhantomJS. Heritrix uncovered 1,044 URI-Rs to add to the frontier, while PhantomJS discovered 403 URI-Rs to add to the frontier. This phenomenon of Heritrix having a larger frontier than PhantomJS is due to Heritrix’s policy of looking into the JavaScript files to extract URIs found in the code – the URI-Rs discovered by Heritrix are TLDs listed in the JavaScript that may be used to construct URIs at run time (e.g., appending a username or timestamp to the URI) or not used by JavaScript at all (e.g., a URI that exists in an un-executed code block). We expect this phenomenon to occur with all small seed lists because the Heritrix URIs will not overlap and therefore will not be de-duplicated at small scales.

During the deferred crawl, PhantomJS crawled 0.5 URIs/sec. Heritrix ran 12.56 URIs/sec, 25.12 times faster than PhantomJS. Heritrix added 3,206 URIs to the frontier, while PhantomJS added 3,436 URIs to the frontier. PhantomJS adds more URIs to the frontier despite Heritrix’s introspection on the JavaScript of each crawl target. This result is due to PhantomJS’s execution of JavaScript on the client.

We observe that the PhantomJS frontier outperforms the Heritrix frontier during the deferred crawl. Heritrix crawls URIs faster than PhantomJS on each of the deferred and nondeferred crawls, but far exceeds the speed of PhantomJS during the deferred crawl.

⁶For the purposes of this part of our investigation, we do not consider descendants. We exclude checking for descendants since we are only measuring the performance of PhantomJS versus Heritrix and discuss the additional impact of descendants in Section 7.6.

7.4 CLASSIFYING REPRESENTATIONS

In practice, archival crawlers such as Heritrix would ideally be able to identify URI-Rs that will be difficult to archive in real-time. Heritrix currently does not have such an automatic classification capability. Archive-It uses a hard-coded set of URI-Rs (identified by human administrators) to identify URI-Rs with deferred representations.

The ability to determine whether a resource will be difficult to archive or not will allow Heritrix to assign the URI-R to either the faster, traditional Heritrix crawler or the slower, PhantomJS (or other JavaScript-enabled) crawler. By enabling this two-tiered crawling approach, the archival crawlers can achieve an optimal crawl speed by utilizing the heavy-duty JavaScript-capable crawlers (e.g., PhantomJS) for only those URI-Rs that need it. However, this approach requires the ability to, in real-time, recognize or predict a deferred representation.

In an effort to predict whether or not a representation would be deferred, we constructed a feature vector of DOM attributes and features of the embedded resources. We used Weka [120] to classify the resources on subsets of the feature vectors to gauge their performance. We extracted the following features and use them to create a feature vector representing the resource:

1. **Ads:** Using a list of known advertisement servers, we determined whether or not a representation would load an ad based on DOM and JavaScript analysis.
2. **JavaScript Tags:** We counted the number of script tags with JavaScript, both in files and embedded code.
3. **Interactive Elements:** We counted the number of DOM elements that have JavaScript events attached to them (e.g., `onclick`, `onload`).
4. **Ajax (in JavaScript):** To estimate the number of Ajax calls (e.g., `$.get()`, `XmlHttpRequest`) we counted the number of occurrences of Ajax requests in the embedded external and independent JavaScript files.
5. **Ajax (in HTML):** To estimate the number of Ajax calls (e.g., `$.get()`, `XmlHttpRequest`), we counted the number of occurrences of Ajax requests in JavaScript tags embedded in the DOM.

6. **DOM Modifications:** We counted the number of times JavaScript made a modification of the DOM (e.g., via the `appendChild()` function) to account for DOM modifications after the initial page load.
7. **JavaScript Navigation:** We counted the occurrences of JavaScript redirection and other navigation functions (e.g., `window.location` calls).
8. **JavaScript Storage:** We count the number of JavaScript references to storage elements on the client (e.g., cookies) as an indication of client-controlled information.
9. **Found, Same Domain:** Using PhantomJS, we counted the number of embedded resources originating from the URI-R's TLD that were successfully dereferenced (i.e., returned an HTTP 200).
10. **Missed, Same Domain:** Using PhantomJS, we counted the number of embedded resources originating from the URI-R's TLD that were not successfully dereferenced (i.e., returned a class HTTP 400 or 500).
11. **Found, Different Domain:** Using PhantomJS, we counted the number of embedded resources originating outside of the URI-R's TLD that were successfully dereferenced (i.e., returned an HTTP 200).
12. **Missed, Different Domain:** Using PhantomJS, we counted the number of embedded resources originating outside of the URI-R's TLD that were not successfully dereferenced (i.e., returned a class 400 or 500 HTTP response code).

We manually sampled 440 URI-Rs (from our collection of 10,000) and classified the representations as deferred or nondeferred, with 200 training and 20 test URI-Rs for each based on whether or not their representations were deferred.

Even though our goal is to detect whether or not representations are dependent on JavaScript, the simple presence of JavaScript is not a suitable indicator of a deferred representation; JavaScript can alter the appearance of the page or DOM without issuing a new HTTP request for a new resource. In our set of URI-Rs, the resources with deferred representations had, on average, 21.98 embedded script tags or files, while the resources with nondeferred representations had 5.3 script tags or files. Of those resources with deferred representations, 84.1% had at least one script tag, while 49.5% of the nondeferred representations had at least one JavaScript tag. Because of

Actual Classification	Predicted Classification	
	Deferred	Nondeferred
Deferred	182	38
Nondeferred	58	166

TABLE 20: Confusion matrix for the entire feature vector (F-Measure = 0.791).

Actual Classification	Predicted Classification	
	Deferred	Nondeferred
Deferred	179	41
Nondeferred	47	173

TABLE 21: Confusion matrix for the resource features (features 9-12 of the vector; F-Measure = 0.844).

the ubiquity of JavaScript in both deferred and nondeferred representations, we opted for a more complex feature vector to represent the features of the representations.

Using PhantomJS, we collected the 12 features required for a feature vector for each of our 440 URI-Rs. Using Weka, we ran each classifier on the feature vectors. Rotation Forests [242] performed the best of any of the standard Weka classifiers for any of our datasets.

We used three subsets of the feature vector to investigate the best method of predicting deferred representations. We selected attributes 1-8 to represent DOM-derived features (those that can be collected without monitoring HTTP traffic). We selected attributes 9-12 as embedded resource attributes, as in the attributes we can extract if we load and monitor the embedded resources. Together, attributes 1-12 make up the entire dataset. We use these three feature sets (DOM-derived, monitor resources, and full) to train and test our classifier.

We use a 10-fold cross validation of our training set data. We use the same three data subsets and provide a confusion matrix of each set to include the entire feature vector (Table 20), the resource feature vector (Table 21), and the DOM feature vector (Table 22).

The accompanying statistics for the classifications are shown in Table 23. As a reminder, accuracy is defined in Equation 5, and F-Measure is defined in Equation 6. With only the DOM features, the test set is accurately classified representations as

Actual Classification	Predicted Classification	
	Deferred	Nondeferred
Deferred	168	52
Nondeferred	41	179

TABLE 22: Confusion matrix for the DOM features (features 1-8 of the vector; F-Measure = 0.806).

Features	Classification	Accuracy	F-measure	Precision	Recall
DOM Features Only	Deferred	79%	79%	78%	81%
	Nondeferred			76%	80%
DOM & Resource Features	Deferred	81%	82%	79%	81%
	Nondeferred			90%	80%

TABLE 23: Classification success statistics for DOM-only and DOM and Resource feature sets.

deferred or nondeferred 79% of the time. With only the embedded resource features, the test set is accurately classified 81% of the time. If we combine both sets to create the full feature set, we can correctly classify representations 81% of the time.

After a URI is dereferenced and a representation is returned, we can determine whether or not the representation is deferred with 79% accuracy. If we also dereference the URIs for the embedded resources and monitor the HTTP status codes, we can increase, albeit minimally, the accuracy of the prediction to 81% of the time. However, crawling with PhantomJS is much more expensive. Due to this minimal improvement and much higher cost to measure, we limit the feature extraction to the DOM classification. With a negligible impact on performance (Heritrix already dereferences the URI-R during the crawl), our classifier is able to identify deferred representations using the DOM crawled by Heritrix with 79% accuracy.

7.5 PERFORMANCE OF TWO-TIERED CRAWLING

Due to Heritrix’s limitations when discovering JavaScript-dependent embedded resources and the slow crawl speed of PhantomJS, a two-tiered crawling approach to crawling would allow an archive to simultaneously benefit from the frontier size

Crawl Strategy	Crawl Time (hrs)	Crawl Rate (URIs/sec)	Frontier Size (URIs)
wget	416.16	0.864	129,443
Heritrix	407.53	2.065	302,961
PhantomJS	8,684.38	0.170	531,484
Heritrix + PhantomJS	9,100.54	0.152	537,609
Heritrix + PhantomJS with Classifier	6,495.23	0.196	458,815

TABLE 24: A summary of *extrapolated* performance (based on our calculations) of single- and two-tiered crawling approaches.

of PhantomJS and the speed of Heritrix. Table 24 provides a summary of the extrapolated crawl speed and discovered frontier size of each crawler. While the test environment used a single system, a production environment should expect to see performance improvements with additional resources. PhantomJS crawls are not run in parallel, and additional nodes for PhantomJS threads will further improve performance.

We have described the operation of crawls with wget, Heritrix, and PhantomJS in Sections 7.3.1 and 7.3.4. To reiterate, Heritrix crawls much more quickly than PhantomJS, whereas PhantomJS discovers many more embedded resources required to properly construct a representation. Optimally during a crawl, Heritrix would dereference a URI-R and run the resulting DOM through the classifier to determine whether or not the representation will be deferred (with 79% accuracy, as discussed in Section 7.4). If the representation is predicted to be deferred, PhantomJS should also be used to crawl the URI-R and add the newly discovered URI-Rs to the Heritrix frontier.

Heritrix should be used to crawl all URI-Rs in the frontier because the DOM is required to classify a representation as deferred. Since Heritrix is the fastest crawler, it should be used to dereference the URI-Rs in the frontier and retrieve the DOM of the resource for classification. Subsequently, only if the representation is classified as deferred will PhantomJS be used to crawl the resource to ensure the maximum amount of embedded resources are retrieved.

Crawler	URI-R Set	Seed Size	Frontier Size	Crawl Time
PhantomJS	Deferred	5,187	311,903	84.9 hrs
Heritrix	Nondeferred	4,813	124,728	23.6 hrs
Heritrix	Deferred	5,187	171,499	26.7 hrs
PhantomJS	All URI-Rs	10,000	438,388	686 hrs
Heritrix	All URI-Rs	10,000	275,234	48.3 hrs
Two-tier	All URI-Rs	10,000	399,202	133 hrs

TABLE 25: A simulated two-tiered crawl showing that the frontier sizes can be optimized while mitigating the performance impact of PhantomJS’s crawl speed vs Heritrix’s.

For a naïve two-tiered crawl strategy that will discover the most embedded URI-Rs and create the largest frontier, Heritrix and PhantomJS should both crawl each URI-R regardless of whether the representation can be classified as deferred or non-deferred. This strategy creates a crawl that is expected to be 13.5 times slower than simply using Heritrix, but is expected to discover 1.77 times more URI-Rs than using only Heritrix. This would ensure that 100% of all resources with deferred representations would be crawled with both Heritrix and PhantomJS. However, our classifier has an accuracy of 79%, and we want to limit the use of PhantomJS to minimize the performance impacts it has on the crawl speed.

If we include the classifier to predict when PhantomJS should be used or when Heritrix will be a suitable tool, the two-tiered crawling approach is expected to run 10.5 times slower than simply using Heritrix. However, this policy is expected to discover 1.5 times more URI-Rs than Heritrix. This crawl policy balances the trade-offs between speed and larger frontier size by using the classifier to indicate when to use PhantomJS to crawl resources with deferred representations.

To validate this expected calculation, we classified our 10,000 URI-R dataset, which produced 5,187 URI-Rs classified as having deferred representations, and 4,813 as having nondeferred representations. We used PhantomJS to crawl the URI-Rs classified as deferred, and only Heritrix to crawl the URI-Rs classified as nondeferred. The results of the crawls are detailed in Table 25.

In this table, we show that PhantomJS creates a frontier of 438,388, 1.6 times larger than that of Heritrix. However, PhantomJS crawls 14 times slower than Heritrix. If we perform a tiered crawl in which PhantomJS is responsible for crawling

only deferred representations, we can crawl 5.2 times faster than using PhantomJS for all URI-Rs (but 2.7 times slower than the Heritrix-only approach, or *the old way*) while creating a frontier 1.8 times larger than using only Heritrix. As a result, we can maximize the frontier size, mitigate the impacts of JavaScript on crawling, and mitigate the impact of the reduced crawl speeds when using a two-tiered crawling approach.

7.6 MAPPING AND IDENTIFYING DESCENDANTS

In the remainder of this chapter, we adapt the Dincturk et al. Hypercube model [80] to construct a model for archiving descendants, and we measure the number of descendants and requisite embedded resources discovered in a proof-of-concept crawl. We define a representation constructed as a result of user interaction or other client-side event without a subsequent request for the resource’s URI as a descendant (i.e., a member of the client-side event tree below the root).

We explore the number and characteristics of descendants as they pertain to Web archiving and explore the cost-benefit trade-off of actively crawling and archiving descendants en route to a higher quality, more complete archive. Dincturk et al. constructed a model for crawling RIAs by discovering all possible descendants and identifying the simplest possible state machine to represent the states. We explore the archival implementations of their Hypercube model by discovering client-side states and their embedded resources to understand the impact that deferred representations and descendants have on the archives.

We evaluate the performance impacts of exploring descendants (i.e., crawl time, depth, and breadth) against the improved coverage of the crawler (i.e., frontier size) along with the presence of embedded resources unique to descendants in the Internet Archive, using Heritrix as our case study of a Web-scale crawling tool. We show that the vast majority (92% in s_1 and 96% in s_2) of embedded resources loaded as a result of user interactions are not archived, and that there are only two levels in the interaction tree.

Dincturk et al. present a model for crawling RIAs by constructing a graph of descendants⁷. Their work focuses on Ajax requests for additional resources initiated by client-side events which leads to deferred representations with descendants.

⁷Dincturk et al. refer to these as “AJAX states” within the Hypercube model; we use a tree structure and therefore refer to these as descendants.

Their work, which serves as the mathematical foundation for our work, identifies the challenges with crawling Ajax-based representations and uses a hypercube strategy to efficiently identify and navigate all client-side states of a deferred representation. The Hypercube model defines a client-side state as a state reachable from a URL through client-side events and is uniquely identified by the state's DOM. That is, two states are identified as equivalent if their DOM (e.g., HTML) is directly equivalent.

The hypercube model is defined by the finite state machine (FSM)

$$M = (S, s_0, \Sigma, \delta)$$

and defined further in Equation 13, where

- S is the finite set of client states
- $s_0 \in S$ is the initial state reached by dereferencing the URI-R and executing the initial on-load events
- $e \in \Sigma$ defines the client-side event e as a member of the set of all events Σ
- $\delta : S \times \Sigma \rightarrow S$ is the transition function in which a client-side event is executed and leads to a new state

$$\begin{aligned}
 s_i, s_j &\in S \\
 \delta(s_i, e) &= s_j \\
 e &= \text{client-side event} \\
 j &= i + 1
 \end{aligned} \tag{13}$$

Dincturk et al. define a graph $G = (V, E)$ in which V is the set of vertices $v_i \in V$ where v_i represents an “AJAX State” s_i . Edges represent the transitions, or events, e such that $(v_i, v_j; e) \in E$ IFF $\delta(s_i, e) = s_j$. A path P is a series of adjacent edges that constitute a series of transitions from s_0 to s_i via $e_{i...j}$. In effect, P is a series of descendants derived from s_0 with one descendant at each level of the tree.

We adopt the FSM presented by Dincturk et al. nearly in its entirety. Because our application of this FSM is web archiving, our goal is to identify all of the embedded resources required by the representation to build any descendant as a result of user

interactions or client-side events, archive them, and be able to replay them when a user interacts with the memento.

The representation returned by simply dereferencing a URI-R is defined as $URI-R_{s_0}$. Subsequent descendants $URI-R_{s_i}$ and $URI-R_{s_j}$ are derived from $URI-R_{s_0}$ through a series of events $e_{i...j} \in \Sigma$. We define a descendant $URI-R_{s_i}$ as a client-side state originating at $URI-R_{s_0}$ as transitioning via events e such that $\delta(s_0, e) = s_1$. Additionally, we define our paths through G as the set of embedded resources required to move from s_0 to s_i .

We present a generic interaction tree of descendants in Figure 93. When we dereference a URI-R, we get a representation from the server; this is s_0 . If there are two interactions available from s_0 , we can execute the interactions to get to V_a or V_b from our root s_0 (note that the onclick event required an external image to be retrieved). In this example, V_a and V_b are descendants of s_0 and are both s_1 in P from s_0 . If new interactions are available from V_a , we can reach V_c and V_d , which are both s_2 in P from s_0 (similarly, we can reach V_e and V_f from V_b , peers of V_c and V_d).

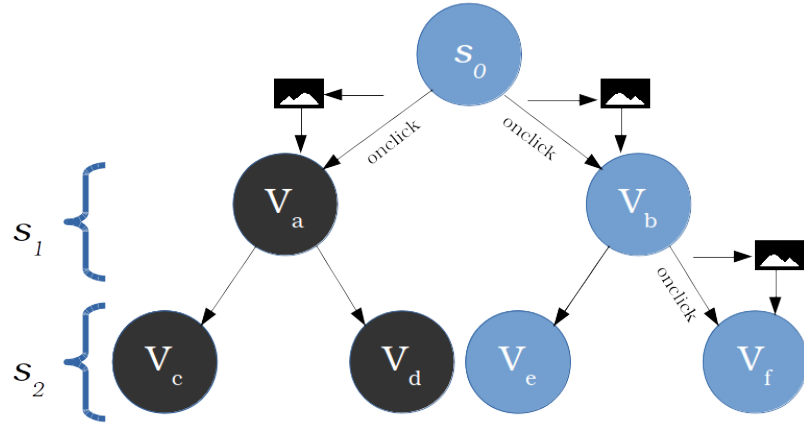


FIG. 93: A generic, three-level client-side state tree with interactions as state transitions.

Because of the differences between our model and the hypercube model (Section 7.7), our focus on web archiving, and to ensure we have omniscient knowledge of all interactions, state dependencies, equivalences, and available interactions, we organize the states as a tree rather than a hypercube. Because new interactions lead to states s_n deeper in the tree, we generalize the levels of the trees as s_n and refer to new states by their vertices V_n .

7.7 DESCENDANT EQUIVALENCY

Due to the archival focus of this study, we have a different concept of state equivalence than the Hypercube model. While Dincturk establishes state equivalence based on the DOM (using strict equivalence based on string comparison), we consider the embedded resources required to construct a descendant. We consider descendants to be equivalent if they rely on the same embedded resources. As such, we define the set of embedded resources for a descendant s_n as R_n .

Any two descendants with identical unordered sets of embedded resources are defined as equivalent, effectively a bijection between the two descendants. P between s_0 to s_i is isomorphic if over the course of P , the embedded resources required are identical, and each descendant within P are bijections. Note that in Figure 93, V_a , V_c , and V_d are equivalent because they require the same set of embedded resources, even if V_c and V_d are reached through an additional e_i and e_j from V_a .

Two paths are identical if, over the course of each $s_n \in P$, the cumulative set of embedded resources required to render each descendant is identical. We define the set of embedded resources over the entire path as RP in Equation 14.

$$RP = \sum_{i=0}^{n \in P} R_i \quad (14)$$

We present our process for traversing paths in Algorithm 1. We traverse all states within the interaction tree to understand what embedded resources are required by each state. If a state s requires a new embedded resource that has not yet been added to the crawl frontier, it is added as part of path P . From RP , we identify the archival coverage (using Memento – line 10). We also identify the duplicate URI-Rs by canonicalizing using the *Base trim* policy to determine equality.

As an example, we present the state tree of a Bloomberg.com page in Figure 94. At s_0 , the page has a menu at the top of the page with a mouseover event listener. Mousing over the labels initiates Ajax requests for JSON data, and the data is used to populate a sub-menu (s_1). The sub-menu has another mouse-over menu that requests images and other JSON data to display new information, such as stock market data and movie reviews (s_2). Note that s_1 and s_2 are very broad given the number of menu items. This is an example of P through two levels of mouseover interactions

```

1 forall the URI-R do
2   find  $s_0...s_n$ ;
3   construct tree G of all progressions;
4   traverse G; identify  $R_n$  of  $s_n$ ;
5   if  $s_n$  has a new resource then
6     treat  $s_n$  as part of P;
7   end
8 end
9 identify RP;
10 find URI-Ms of RP to determine coverage;
11 de-duplicate RP to determine overlap;

```

Algorithm 1: Algorithm for traversing *P*.

that leads to new JSON and image embedded resources (Figure 95).

The page also has click events such as the “x” mark on the top right corner of the advertisements. If the user clicks these, a request is made to the ad server to issue new ads (s_1). However, the ads may be new but the call to the server is not, resulting in duplicate descendants in s_1 . Similarly, the comments section at the bottom of the page is loaded into s_0 via Ajax, and has a “Sort” feature. Selecting a sort feature issues an Ajax request for the data in the comments section; note that the same URI is used regardless of the sort option selected (Figure 96). This results in duplicate descendants in s_1 .

Finally, note that the comments section has a form that users can fill out to enter a comment. This is beyond the scope of this work; this type of action changes the representation itself and can lead to infinite states, changing the nature of our investigation from identifying archival targets to altering the live Web record.

7.8 DESCENDANT CRAWLING APPROACH

To measure descendants, we needed to construct a tool that can crawl, uncover, and understand descendants and deferred representations. We have previously shown that PhantomJS is an effective utility for crawling deferred representations (Section 7.3.4). We constructed a PhantomJS-based utility that dereferences a URI-R, identifies the interactive portions of the DOM (i.e., the DOM elements with event listeners), and constructs a tree of descendants, reached via initiating interactions and client-side events (just as in the Hypercube model). PhantomJS records the set of

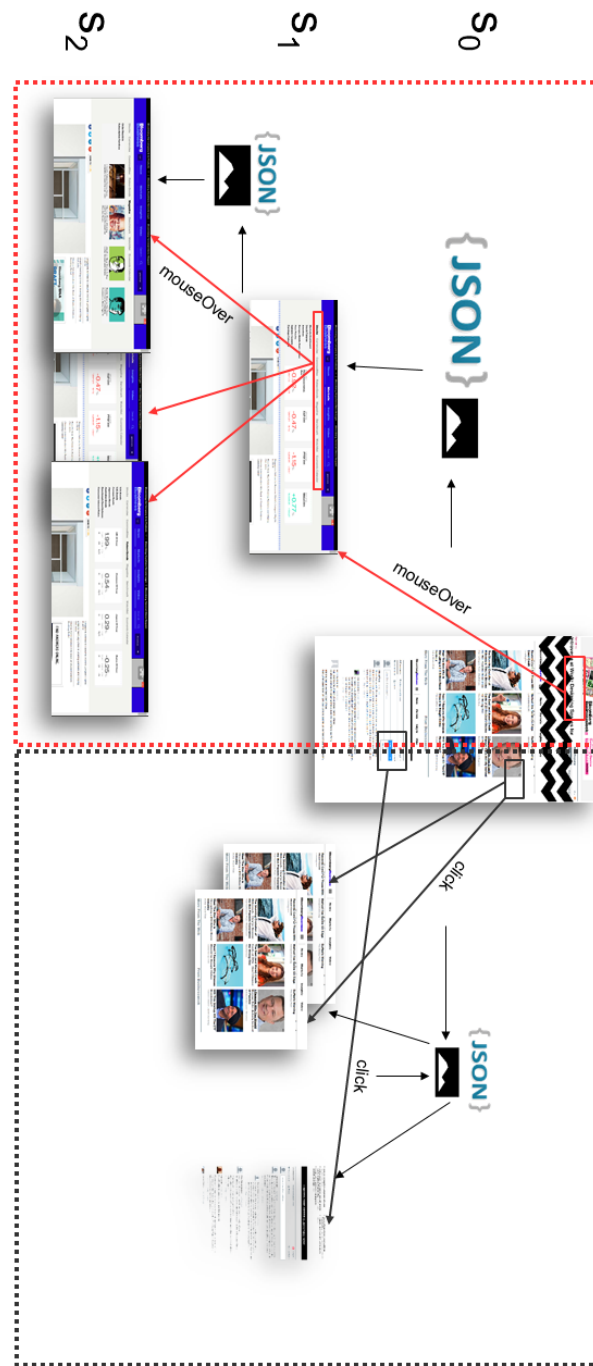


FIG. 94: Example state tree of <http://www.bloomberg.com/bw/articles/2014-06-16/open-plan-offices-for-people-who-hate-open-plan-offices>. Mouseover events lead to multiple descendants at s_1 and further mouseover events lead to descendants at s_2 , each requiring Ajax requests for JSON and image resources. Please refer to Figures 95 and 96 for larger representations of this Figure.

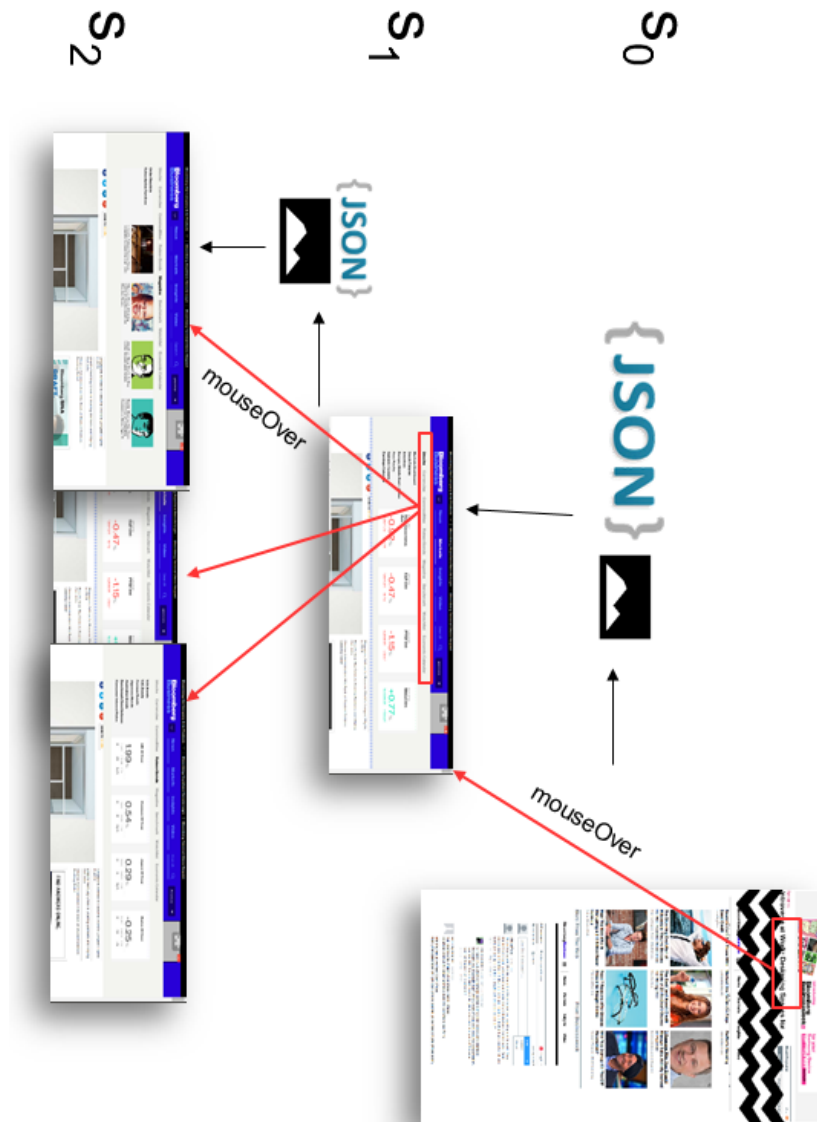


FIG. 95: Mouseover events attached to the menu bar lead to Ajax requests for JSON and images to build submenus. Note that this is an up-close view of the left side of Figure 94.

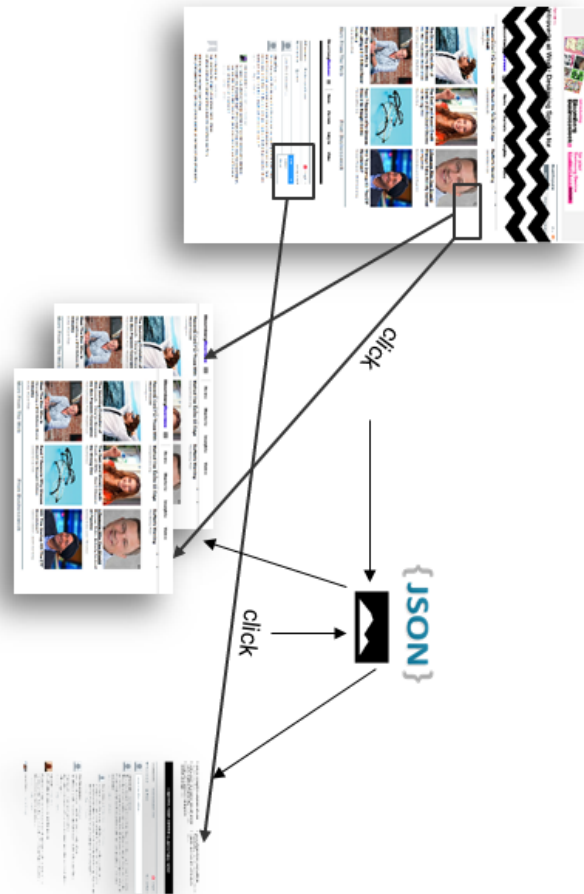
S_2 S_1 S_0 

FIG. 96: Click events attached to embedded advertisements and the comment section of the page lead to Ajax requests for JSON to serve new advertisements and to sort the comments. Note that this is an up-close view of the right side of Figure 94.

embedded resources requested by the client; in a production system, this would be the set of resources added to the Heritrix crawl frontier.

Because PhantomJS is closely tied to the DOM and client’s JavaScript engine, race conditions and other event listener assignments prevent PhantomJS from understanding the entirety of events available on a representation. As such, we leveraged VisualEvent⁸, a bookmarklet that is designed to visually display the DOM elements that have event listeners and JavaScript functions attached, to understand which events and interactions can be executed on the client. Our PhantomJS tool uses the list of events identified by VisualEvent to construct a set of interactions E that may lead to descendants. PhantomJS performs an exhaustive depth-first traversal of all possible combinations of interactions. Post-mortem, we perform state equivalence and identify the number of unique paths P , states s_n , and embedded resources RP that a crawler would have to visit in order to comprehensively archive the resources needed to provide full functionality in a memento.

Note that our use of PhantomJS and VisualEvent is an alternate approach from that used by Browsertrix and WebRecorder.io (Section 3.1). We performed an evaluation of Selenium and PhantomJS for their suitability to simulated and initiate client-side events (a form of simulated user interaction with a page) [48]. We concluded that PhantomJS – with additional tools like VisualEvent – is better for automatically detecting available user interactions and triggering them, but Selenium is the superior tool for monitoring streaming content and playing back a canned script of user interactions.

Note that a V_n is reached by a series of interactions $e_{i...j}$. We consider a descendant that is a candidate to add to the tree identical to another descendant within the tree if the set of interactions to reach the descendant are identical. If we encounter a potential descendant that is reachable by the same interactions as another descendant within the tree, we do not add the descendant to the tree because the descendant already exists within the tree⁹.

We present our algorithm for crawling the descendants in Algorithm 2; this describes Steps 3’ and 3’’ in the two-tiered crawling approach (Section 7.2). We begin

⁸VisualEvent is an open source project (<https://github.com/DataTables/VisualEvent>) that overlays a DIV on a page and visually highlights the interactive DOM elements along with their event handlers.

⁹Note that this refers to equivalency of interaction scripts, meaning the crawler should not visit this state, rather than two states that are reached with different interactions but have the same sets of embedded resources (Section 7.7).

by using PhantomJS to dereference a URI-R (line 4) at s_0 , and use VisualEvent to extract the interactive elements (line 5). We identify all possible combinations of interactions and use them as an interaction frontier (line 7), and iterate through the interaction frontier to crawl s_1 . From s_1 , we extract all possible interactions available, and add them to the interaction frontier. We iterate through the interaction frontier until we have exhausted all possible combinations of interactions at each s_n . At the end of each s_n construction, we run state deduplication (line 14). We deem two interaction scripts as equivalent if they perform identical actions in identical order:

$$\{e_i, e_{i+1}, \dots, e_i + n\} = \{e_j, e_{j+1}, \dots, e_j + n\}$$

```

1 forall the URI-R do
2   run state  $s_0$ ;
3   begin
4     run PhantomJS; monitor and log  $R_n$ ;
5     call VisualEvent functions to retrieve interactive DOM elements and
       $e_{i\dots j}$ ;
6   end
7   construct interaction scripts from all possible combinations of available
      interactions to read  $s_n$ ;
8   forall the interaction scripts do
9     run PhantomJS, executing interactions in script;
10    monitor and log  $R_n$ ;
11    call VisualEvent for events to reach  $s_{n+1}$ ;
12    construct new  $s_{n+1}$ ;
13    push new interaction scripts to list of all interaction scripts;
14    run interaction script de-duplication begin
15      if  $s_{n+1}$  has identical set of interactions in  $G$  then
16        remove  $s_{n+1}$ 
17      end
18    end
19  end
20 end

```

Algorithm 2: Algorithm for constructing G .

7.9 EDGE CASES

The approach that we identify in Section 7.8 is suitable for most of the deferred

Event Type	Deferred		Nondeferred	
	Average	s	Average	s
Depth	0.47	0.5	0	0
Breadth	36.16	97.15	0.53	2.62
Descendants	38.5	780.72	0.62	2.81

TABLE 26: The average distribution of descendants within the deferred representation URI-R set.

representations that a Web user may encounter while browsing. However, deferred representations with certain conditions are not handled by our approach. Some representations use a DIV overlayed on the entire window area and identify interactions and events according to the pixel the user clicks. This creates an interaction frontier of $(\text{Width} \times \text{Height})!$ or $2,073,600!$ for a screen size of 1920×1080 pixels. Due to this massive frontier size, we omit such interactions. Mapping (e.g., Google Maps) and similar applications that might have a near-infinite descendants are outside the scope of this work.

For these style of deferred representations, a *canned* set of interactions (e.g., pan once, zoom twice, right click, pan again) would be more useful [48]. With enough of these canned interactions, a sizable portion of the descendants can be identified by a crawler over time, with coverage scaling with the number of executions performed. This is the archival equivalent of the Halting Problem – it is difficult to recognize when the crawler has captured *enough* of the embedded resources, when it should stop, or when it has captured everything.

7.10 DESCENDANT STATES

During our crawl of the 440 URI-Rs, we re-classify each as having a deferred or nondeferred representation. Our initial classification only considered whether Ajax was used at s_0 since s_0 was the only level being crawled. We dereferenced each of our 440 URI-Rs and identified 137 URI-Rs with nondeferred representations and 303 URI-Rs with deferred representations.

7.10.1 DATASET DIFFERENCES

The nondeferred URI-Rs have a much smaller graph of descendants, and therefore

# States	Deferred	Nondeferred
Min	0	0
Max	7,308	13
Median	1 (occurrences 17)	0 (occurrences 119)

TABLE 27: The range of descendants varies greatly among the deferred representations.

we expect them to be easier to crawl. The nondeferred representation set of URI-Rs had $\overline{|S_{descendants}|} = 0.62$ per URI-R ($s = 2.81$, $M = 101$) as shown in Table 26. Nondeferred representations have a depth (i.e., max length of the P) of 0 (after state deduplication) because there are no new states reached as a result of the first round of interactions (that is, the set of interactions available in the initial representation does not grow as a result of subsequent interactions). Nondeferred representations have descendants at s_1 but the descendants do not result in additional embedded resources. However, there are 0.53 interactions or events in s_0 that, without our *a priori* knowledge of the dataset, may lead to new states or event triggered requests for new embedded resources.

Deferred representations are much more complex, with $\overline{|S_{descendants}|} = 38.5$ per URI-R ($s = 780.72$). The standard deviation of the sample is quite large, with the number of states varying greatly from resource to resource. For example, the maximum number of descendants for a URI-R is 7,308 (Table 27). Further, there are many interactions available in deferred representations (36.16 per URI-R). Surprisingly, deferred representations are relatively *shallow*, with an average depth of 0.47 levels beyond the first set of interactions per URI-R ($s = 0.5$) and a maximum path depth of 2. This is counter to our intuition that deferred representations would have large, deep trees of interactions to traverse to retrieve all of the possible embedded resources for all descendants¹⁰.

The types of events on the client also vary depending on the event that is executed to create the new s_n . For example, onclick events are prevalent in URI-Rs with deferred representations, with 62.11% of all URI-Rs containing an onclick event (Table 28). Even in the nondeferred set of URI-Rs, 4.29% of the URI-Rs have an onclick event attached to their DOM. While other events occur with relative frequency, clicks dominate the initiated requests for additional embedded resources in

¹⁰Our dataset and toolset omits edge cases as described in Section 7.9.

Event Type	Percent of URI-Rs		Contribution to RP_{new}
	Deferred	Nondeferred	
click	62.11%	4.29%	63.2%
mouseover	25.26%	3.00%	4.7%
mousedown	16.84%	1.72%	2.8%
blur	14.74%	0.86%	9.8%
change	11.58%	2.14%	0.0%
mouseout	8.42%	0.00%	0.8%
submit	6.32%	0.00%	0.0%
unload	5.26%	0.00%	1.2%
keydown	4.21%	0.00%	0.2%
focus	4.21%	0.00%	0.0%
keypress	2.11%	0.00%	5.5%
focusout	1.05%	0.00%	0.0%
dblclick	1.053%	0.00%	0.0%
submit	0.10%	0.43%	0.9%
mouseup	0.00%	0.86%	0.0%
focus	0.00%	0.43%	0.0%
other	29.47%	0.86%	11.0%

TABLE 28: Breakdown of the URI-Rs with various events attached to their DOMs and the percent of all new embedded resources contributed by the events.

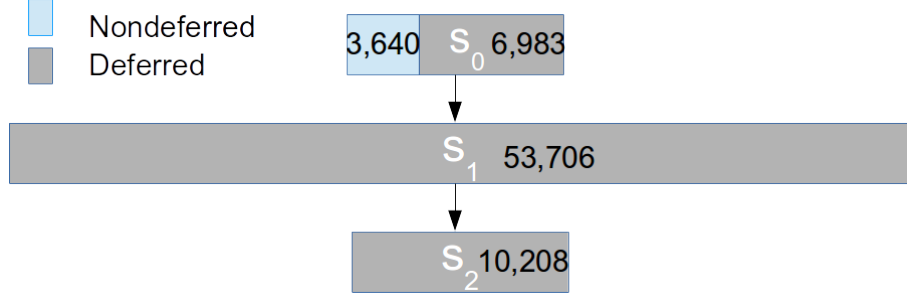


FIG. 97: Crawling s_1 provides the greatest contribution to RP ; the additions to the crawl frontier by s_0 (10,623) and s_2 (10,208) only differ by 415 URIs.

deferred representations (Table 28), with onclick events being responsible for initiating the requests for 70.9% of new embedded resources (and, by definition, 0% in the nondeferred representation set). Recall that Rosenthal et al. are using only click interactions to interact with pages in their LOCKSS Extension project [248]. Table 28 suggests that their approach is effective considering most events are onclick events and the highest value target event (i.e., the most embedded resources are discovered through click events).

7.10.2 TRAVERSING PATHS

As we discussed in Section 7.7, P identifies a unique navigation through descendants to uncover the URI-Rs of new embedded resources. In our dataset, we uncovered 8,691 descendants (8,519 for the deferred set, 172 for the nondeferred set) as a result of client-side events, that is 19.7 descendants per URI-R. However, we only identified 2,080 paths through these descendants to uncover all of the new embedded resources, which is 4.7 paths per URI-R.

Nondeferred representations have more embedded resources ($R_0 = 31.02$ per URI-R) than their deferred counterparts ($R_0 = 25.39$ per URI-R) at their initial s_0 (Table 29). The paths P through the descendants are responsible for uncovering 54,378 new embedded resources (out of 66,320 total). That is, $|R_0| = 11,942$ (7,692 from the deferred representations and 4,350 from the nondeferred representations), $|R_0 + R_1| = 56,957$, and $|R_0 + R_1 + R_2| = |RP| = 66,320$. This shows that traversing P_n to reach s_1 and s_2 will significantly increase the crawl frontier beyond the base case of s_0 , but crawling s_1 provides larger contributions to the frontier than both s_0 and

s_2 . As we mentioned in Section 7.10.1, the depth of the deferred representations was surprisingly shallow ($\max(|P|)=2$). However, the overwhelming majority of the URI-Rs added to the crawl frontier were identified by exploring the paths P of the descendants (Figure 97).

Note that, out of the total 8,691 total descendants, the nondeferred representations have only 138 occurrences of s_0 and 34 occurrences of s_1 . The deferred representations have 6,051 occurrences of s_1 and 2,468 occurrences of s_2 . Following P through each s_1 adds $R_1=53,706$ URI-Rs to the crawl frontier, or 8.88 URI-Rs per descendant. This shows that deferred representations have many more descendants than nondeferred representations. Since $R_2=10,208$, we add, on average, 4.14 new URI-Rs to the frontier per s_2 followed in P . According to these averages, crawling s_1 provides the largest benefit to the crawl frontier. If we consider only the 2,080 paths that lead to new embedded resources, we would add 30.73 URI-Rs to the crawl frontier per P .

7.10.3 IMPACT ON CRAWL TIME

Using our measured crawl times and the set of states S that PhantomJS can uncover and visit, we calculate the expected frontier size and crawl time that we can expect during a crawl of our 440 URI-Rs. We calculated the s_0 crawl time for Heritrix-only crawls using the *Max Coverage* policy, PhantomJS crawls of only the URI-Rs with deferred representations, and s_1 and s_2 uncovered by our PhantomJS utility.

As we note in Table 29, s_1 has the greatest addition to the crawl frontier. If we omit s_2 from our crawl, we still discover 82% of the embedded resources RP required by our deferred representations and reduce crawl time by 30%. Depending on the goal of the crawl, different crawl policies would be optimal when crawling deferred representations. A policy to optimize archival coverage and memento quality should traverse each $s \in S$ and maximize RP ; we will refer to this policy as the *Max Coverage* crawl policy. Alternatively, a crawl policy with the goal of optimizing return-on-investment (ROI) should optimize the crawl time versus frontier size; we refer to this policy as the *Max ROI* crawl policy. For maximizing ROI, we recommend a crawl policy that omits s_2 and instead uses Heritrix and PhantomJS to crawl s_0 and s_1 since s_1 provides the greatest contribution to RP per crawl time (Table 29).

As shown in Table 29, using the Max Coverage policy will lead to a crawl time

	Heritrix only	s_0	s_1	s_2
Time (s)	1,035	8,452	27,990	40,258
Size (URI-Rs)	4,250	11,942	56,957	66,320
Time Increase	-	8.12x	27.04x	38.90x
Size Increase	-	2.81x	13.40x	15.60x
New URIs per added second of crawl time	-	1.04	2.30	0.76

TABLE 29: The increases in run time and frontier size relative to the Heritrix only run.

38.9 times longer than using only Heritrix to perform the crawl, but will also discover and add to the crawl frontier 15.60 times more URI-Rs. Alternatively, the Max ROI policy will have a crawl time 27.04 times longer than Heritrix-only crawls, but will add 13.40 times more URI-Rs to the frontier.

7.11 ARCHIVAL COVERAGE

While the increases in frontier size as presented in Section 7.10 appear impressive, we can only identify the impact on the archives’ holdings by identifying which embedded resources have mementos in today’s archives. We used Memento to retrieve the TimeMap of each embedded resource’s URI-R to determine whether the embedded resource had any mementos (i.e., has been archived before) or if the resource identified by the URI-R has not been previously archived.

The embedded resources from our entire set of 440 URI-Rs are very well archived – only 12% of the set of embedded resources in s_0 do not have a memento. This is consistent with the archival rates of resources from our prior studies [5, 52]. We only consider the *new* embedded resources in the descendants in s_1 and s_2 . That is, we only consider the embedded resources added to the descendant that were not present in the previous state, or the set of resources R_{n+1} not a subset of the previous state’s R_n . More formally, we define new embedded resources R_{new} in Equation 15.

$$R_{\text{new}} = \forall r \in (R_{n+1} - R_n), n \geq 0 \quad (15)$$

We provide the percent of new embedded resources that are unarchived in each state in Figure 98. The unarchived embedded resources are most frequently images

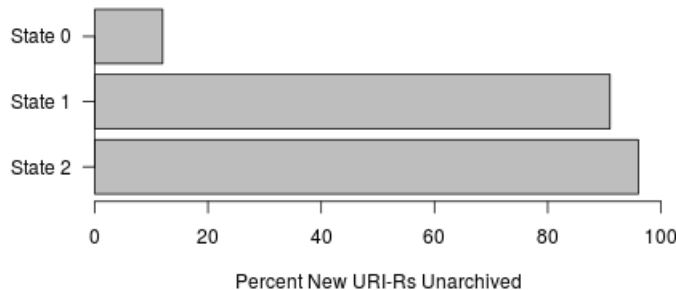


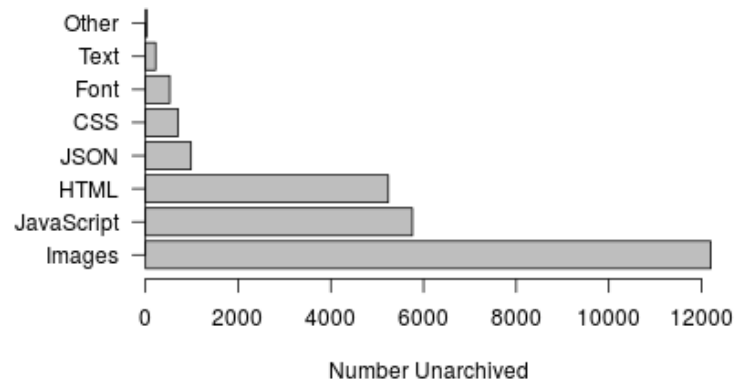
FIG. 98: Embedded resources discovered in s_1 and s_2 are much more frequently unarchived (92% and 96%, respectively) than s_0 (12% unarchived).

(Figure 99(a)), with additional JavaScript files edging out HTML as the second most frequently unarchived MIME-type. The unarchived images specific to deferred representations (shown in the CDF in Figure 99(b)) vary in size between near 0B to 4.6MB, and average 3.5KB. The median image size is 43B, indicating that the majority of images are small in size but several are quite large and presumably important (according to D_m).

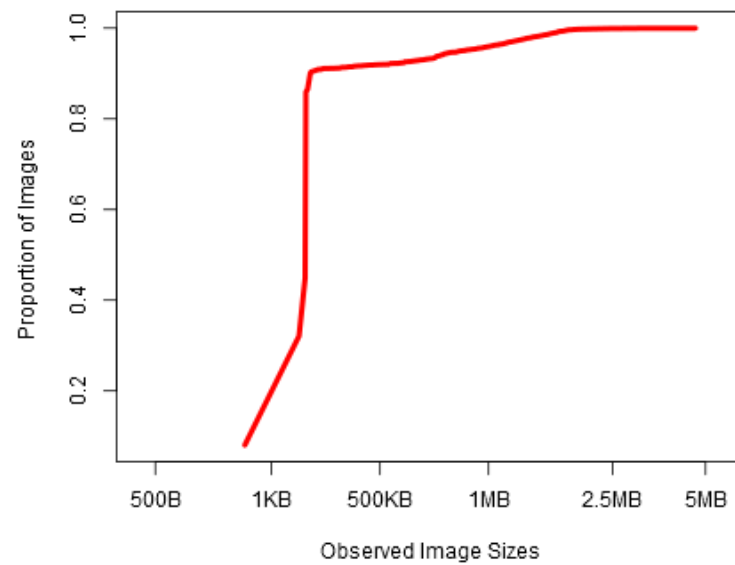
However, the archival coverage of s_1 and s_2 is much lower, with 92% of R_{new} in s_1 missing from the archives (i.e., the URI-R of the embedded resource does not have a memento), and 96% of R_{new} in s_2 missing from the archives. This demonstrates that the embedded resources required to construct descendants are not well archived. Because s_0 is highly visible to crawlers such as Heritrix and archival services like Archive.is, it is archived at a much higher rate than the descendants (Figure 98).

We also observe a large amount of overlap between the embedded resources among descendants. For example, the top 10 embedded resources and their occurrence counts are provided in Table 30 (we trim the session-specific portions of the URI-Rs for comparison purposes). In all, just the top 10 occurring embedded resources account for 22.4% of R_{new} discovered by traversing through the paths. In theory, if we can archive these embedded resources once, they should be available for their peer mementos while in the archives.

The resources in Table 30 are mostly ad servers and data services such as Google Analytics. The top 300 occurring embedded resources in our entire crawl frontier are graphed – in order of most frequent to least frequently occurring – in Figure 100. Figure 101 is a CDF of R_{new} by URI-Rs with deferred representations and measures



(a) Images are most frequently unarchived in deferred representations.



(b) Unarchived image vary in size, with most being small and few being very large.

FIG. 99: Various mime-types of embedded resources are specific to deferred representations. Data, text, and new JavaScript is loaded by JavaScript into the deferred representations.

URI-R	Occurrences
ads.pubmatic.com/AdServer/js/showad.js#PIX&kdntuid=1&p=52041&s=undefined&a=undefined&it=0	1782
edge.quantserve.com/quant.js	1656
www.benzinga.com/ajax-cache/market-overview/index-update	1629
ads.pubmatic.com/AdServer/js/showad.js	1503
www.google-analytics.com/analytics.js	1330
b.scorecardresearch.com/beacon.js	1291
www.google-analytics.com/ga.js	1208
www.google.com/pagead/drt/ui	1151
js.moatads.com/advancedigital402839074273/moatad.js	1112
a.postrelease.com/serve/load.js?async=true	907
Total	12,239

TABLE 30: The top 10 URI-Rs that appear as embedded resources in descendants make up 22.4% of all resources added to the crawl frontier.

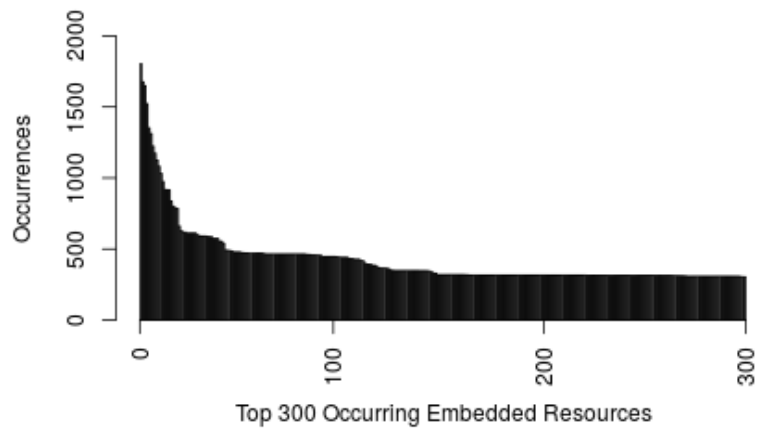


FIG. 100: The occurrence of embedded resources loaded into deferred representation descendants.

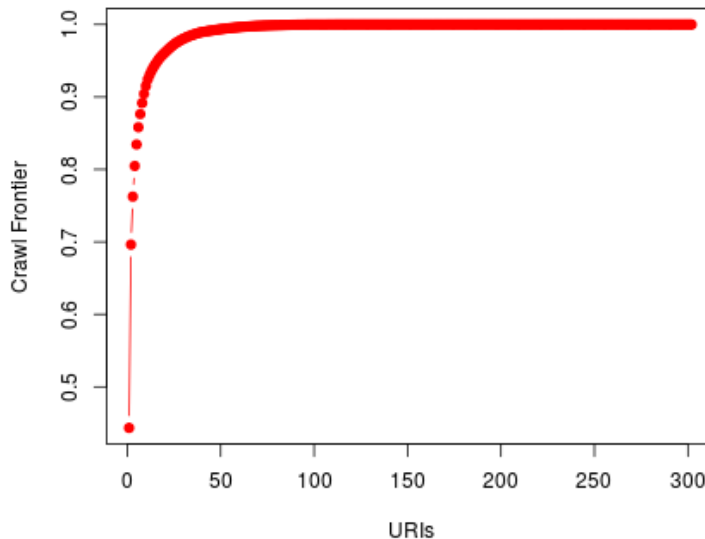


FIG. 101: Contributions of each URI-R and its descendants to RP .

the deduplicated crawl frontier if we crawl all descendants for a URI-R. This shows that the largest 10% of our frontier contributes 91% of RP ; that is, a large portion of the discovered crawl frontier is shared by our seed list.

7.12 STORING DESCENDANTS

Our prior research has shown that representing descendants in the archives is difficult and could even lead to URI-M collisions between mementos (Section 4.4) [145]. Jackson has discussed [134] the challenges with identifying descendants in the archives – particularly those descendants that heavily leverage JavaScript to change the representation without changing the URI-R – may lead to indexing and referencing challenges.

The IIPC proposed an additional set of JSON metadata to better represent representation context and environment variables (e.g., deferred representations and descendants) in WARCs [131, 132]. We adapt the metadata to describe descendants and include the interactions, state transitions, rendered content, and interactive elements (Table 31) and mock-up an example in Figure 102.

Conceptually, the WARC metadata would appear in the metadata section of the WARC. While this specification may be used to represent deferred representations and their descendants, at the authoring of this dissertation, the specification is only

Field Name	Data within field
startedDateTime	Timestamp of interactions (no change from WARC Spec)
id	ID of s_n represented by these interactions and resulting R_n .
title	URI-R of the descendant
pageTimings	Script of interactions (as CSV) to reach s_n from s_0 . E.g., click button A, click button B, then double click image C.
comment	Additional information
renderedContent	The resulting DOM of s_n .
renderedElements	RP from s_0 to s_n .
map	The set of interactions available from s_n that will transition to a new s_{n+1} .

TABLE 31: JSON object representing s_n stored as the metadata of a WARC.

proposed rather than adopted, and may therefore evolve and change. As presented [132], this metadata includes the DOM and descriptions of changes to the DOM as a result of JavaScript and lists of embedded resources.

We present a summary of the storage requirements for descendants (using *Max Coverage*) in Table 32. If we write out the JSON describing the states, transitions, rendered content, and other information, it would add, on average, 16.45 KB per descendant or memento. With 8,691 descendants, a total of 143 MB of storage space for the WARCs will be required just for the metadata, along with the storage space for the representations of the 54,378 new embedded resources.

The embedded resources discovered in our crawl average 2.5 KB in size. The embedded resources at s_0 were 2.6 KB on average, and the newly discovered embedded resources, as a result of deferred representations, were 2.4 KB in size on average. We estimate that nondeferred representations, which have 31.02 embedded resources on average, would require 80.7 KB per URI-R, or 11.1 MB of storage for the 137 URI-Rs in the collection. The storage requirement increases to 13.4 MB with the additional metadata.

Deferred representations have 25.4 embedded resources at s_0 , or 70.0 KB per URI-R. For the 303 URI-Rs in the collection, s_0 would require 21.2 MB of storage. In s_1 , the crawl discovered 45,015 embedded resources which requires 108.0 MB of

```

1  "pages": [
2    {
3      "startedDateTime": "2015-04-20T01:00:00.000",
4      "id": "s_1_testPage.html",
5      "title": "Test Page",
6      "pageTimings": {null},
7      "comment": "state 1 of the test page",
8      "renderedContent": {<html>... <\html>},
9      "renderedElements": ["/20140907_090111.jpg",
10        "/20140903_200818.jpg"],
11      "map": [
12        {
13          "href": "img1",
14          "location": {...}
15        },
16        {
17          "href": "img2",
18          "location": {...}
19        },
20      ]
21    }
22  ]

```

FIG. 102: JSON metadata to be added to WARCS.

Storage Target	Size
JSON Metadata per descendant/memento	16.5 KB
JSON Metadata of all descendants	143 MB
Nondeferred (137 URIs)	
Average Embedded Resource	2.5 KB
Embedded Resources per URI	80.7
Total embedded resource storage	11.1 MB
Total JSON MetaData storage	2.3 MB
Total with JSON Metadata	13.4 MB
Deferred (303 URIs)	
Average Embedded Resource	2.6 KB
Embedded Resources per URI	70.0
Embedded resource storage s_0	21.2 MB
Embedded resource storage s_1	108.0 MB
Embedded resource storage s_2	22.5 MB
Total JSON Metadata Storage	5.0 MB
Total with JSON Metadata	156.7 MB

TABLE 32: The storage impact of deferred representations and their descendants is 5.12 times higher per URI-R than archiving nondeferred representations.

additional storage, and 9,363 embedded resources at s_2 , or an additional 22.5 MB of storage. In all, the 303 deferred representations require 156.7 MB of storage for the entire collection and all crawl levels (8,691 descendants). This is 11.3 times more storage than is required for the nondeferred representations, or 5.12 times more storage per URI-R crawled.

If we consider the July 2015 Common Crawl [189] as representative of what an archive might be able to crawl in one month (145 TB of data for 1.81 billion URIs), an archive would require 597.4 TB of additional storage for descendants (29.9 TB of which is additional storage for metadata) for a total of 742.4 TB to store descendants and metadata for a one-month crawl. If we assume that the July crawl is a representative monthly sample, an archive would need 8.9 PB of storage for a year-long crawl. This is an increase of 7.17 PB per year (including 358.8 TB of storage for metadata) to store the resources from deferred representations. Alternatively, can also say that an archive will miss 6.81 PB of embedded resources per year because of deferred representations.

7.13 CONTRIBUTION TO RESEARCH QUESTION 3

Our investigation of Research Question 3 led us to develop a two-tiered crawling approach with a classifier for mitigating the impact of headless browsing solutions. This recommendation serves as the first part of our answer to Research Question 3.

We measured the differences in crawl speed and frontier size of wget, PhantomJS, and Heritrix. While PhantomJS was the slowest crawler, it provided the largest crawl frontier due to its ability to execute client-side Javascript to discover URIs missed by Heritrix and wget. Heritrix was the fastest crawler we observed, but is not capable of archiving deferred representations. Our two-tiered crawling approach uses a classifier to determine whether to crawl a resource with PhantomJS to reap the URI discovery benefits of the specialized crawler where appropriate.

From the conclusions in Section 7.3, we know that PhantomJS finds 19.70 more embedded resources per URI. Heritrix runs 12.13 times faster than PhantomJS, meaning the crawler should avoid crawling URI-Rs with nondeferred representations to maintain an optimal performance trade-off. We understand that PhantomJS is required to discover the embedded resources needed to complete a deferred representation that Heritrix cannot discover. This increases crawler run time, but offers a benefit of more complete mementos (and likely higher quality mementos) and a larger frontier for crawling. We also found that 53% of the URIs discovered by PhantomJS are duplicates of one another if we remove session-specific URI parameters.

Using DOM features we can accurately predict deferred and nondeferred representations 79% of the time. Using this classification, deferred representations can be crawled by PhantomJS to ensure all embedded resources are added to the crawl frontier.

If using a two-tiered crawling approach, archives can leverage the benefits of PhantomJS and Heritrix simultaneously. That is, using a deferred representation classifier, archives can use PhantomJS for deferred representations and Heritrix for nondeferred representations. Using a two-tiered crawling approach, we showed that crawls will run 5.2 times faster than using only PhantomJS, and create a frontier 1.8 times larger than using only Heritrix. This crawl strategy mitigates the impact of JavaScript on archiving while also mitigating the reduced crawl speed of PhantomJS.

We leverage PhantomJS and adapt the Hypercube model by Dincturk et al. and present a FSM to describe descendants and propose a WARC storage model for describing mementos of descendants.

We show that, despite high standard deviations in our sample, deferred representations have 38.5 descendants per URI-R and that deferred representations are surprisingly shallow, only reaching a depth of 2 levels. This means that deferred representations are shallower than originally anticipated (but also very broad). The URI-Rs we sampled have 8,691 descendants (19.7 per URI-R) that add 54,378 new embedded resources to the crawl frontier.

Crawling all descendants (which we defined as the *Max Coverage* policy) is 38.9 times slower than crawling with only Heritrix, but adds 15.60 times more URI-Rs to the crawl frontier than Heritrix alone. Using the *Max ROI* policy is 27.04 times slower than Heritrix, but adds 13.40 times more URI-Rs to the crawl frontier than Heritrix alone. We do not recommend one policy over the other since the policy selection depends on the archival goals of coverage or speed. However, both will help increase the ability of the crawlers to archive deferred representations.

Most of R_{new} (newly discovered embedded resources by traversing the paths) are unarchived (92% unarchived at s_1 and 96% at s_2). However, after trimming, 22.4% of the newly discovered URI-Rs match one of the top 10 occurring URI-Rs, indicating a high amount of duplication within RP ; mostly, these are ad servers and data-services like Google Analytics.

Our work establishes an understanding of how much Web archives are missing by not accurately crawling deferred representations and presents a process for archiving descendants. We demonstrate that archiving deferred representations is a less daunting task with regards to graph depth than previously thought, with fewer levels of interactions required to discover all descendants. The increased frontier size and associated metadata will introduce storage challenges with deferred representations requiring 5.12 times more storage than crawling nondeferred representations.

CHAPTER 8

FUTURE WORK, CONTRIBUTIONS, AND CONCLUSIONS

When we began our research, the Web archiving community understood that JavaScript makes Web archiving more difficult. However, the impact of, extent of, and potential mitigations to the challenges JavaScript causes were neither measured nor understood by the Web archiving community.

The foundations and motivations of our research began with examples of the challenges JavaScript introduces to the Web archiving community (Chapter 4). We demonstrated that historical events, such as the SOPA protest, were not archivable with today’s technologies (Sections 1.2 and 4.1), and that advertisements – along with other embedded resources loaded via JavaScript – can cause temporal violations in mementos as demonstrated in our presidential campaign example on CNN.com (Section 4.3). We defined mementos that load live Web content and present it as if archived as *zombie resources*, characterizing zombie resource behavior as the live Web *leaking* into the archives. We also defined *deferred representations* to refer to representations that require post-load events (e.g., user interactions, onload events) to fully load all embedded resources and complete a representation (Chapter 2). Finally, we demonstrated that even if we can archive deferred representations, there is no guarantee that we can reliably identify mementos of deferred representations in the archives (Section 4.4).

With this foundational understanding of *how* JavaScript impacts the archives, we measured the pervasiveness of deferred representations and zombie resources in the archives and showed a trend of increased adoption of JavaScript by content authors which leads to more zombie resources over time (Chapter 5). We show that leakage in the archives is increasing over time, as well (Section 5.9).

We wanted to measure how increasing leakage impacted the quality of mementos over time. We developed a metric that more closely matches Web users’ understanding of memento quality than the traditional metric of percent of embedded resources

missing (Chapter 6). We showed that mementos of deferred representations have more damage than nondeferred representations (Section 6.6).

With the impact of JavaScript measured (Chapter 5) and a metric of quality established (Chapter 6), we proposed a two-tiered crawling approach that proposes a change to the way archival crawlers interact with and archive deferred representations (Chapter 7). This area of research has three areas of investigation, the first being an explanation of the differences between legacy crawling approaches and our two-tiered crawling approach (Section 7.2). The second area of investigation compared the performance differences, in terms of wall-clock time to perform a crawl and the size of the crawl frontier, between the legacy crawling approach and the two-tiered crawling approach (Section 7.3). We found that the two-tiered crawling approach is much slower than the legacy approach but discovers a much larger crawl frontier. The third and final area of investigation defines *descendants* and measures the number of descendants of deferred representations that the two-tiered crawling approach may discover (Section 7.6). We also propose using the IIPC WARC metadata to describe and store descendants, and we estimated the additional storage requirements for archiving descendants using the WARC metadata (Section 7.12).

We have presented a body of work in this dissertation that measures the prevalence and impact of deferred representations on the archives and proposes and measures the performance impact of a two-tiered archiving framework that will allow archives to automatically crawl and archive deferred representations at Web scale. We review the future work, contributions, and conclusions of our work in this chapter.

8.1 RESEARCH QUESTIONS REVISITED

In this section, we revisit our research questions posed at the beginning of our research (Section 1.4) and answered in this dissertation.

RQ1. To what extent does JavaScript impact archival tools? Our initial work recognized the phenomenon of zombie resources [54, 47] that *reach* into the live Web and load live Web resources as if they were archived (Section 4.3). We measured the prevalence of deferred representations in the archives (that potentially lead to zombie resources) and show that the adoption of JavaScript over time has led to an increasing number of zombie resources, increasing proportions of missing embedded resources, and increasing archival challenges for the current archival tools

[146] (Section 5).

We performed an experiment establishing that while wget, WebCite, and Heritrix – the current state-of-the-art archival tools at the time – handled JavaScript and deferred representations with varying levels of success, no Web-scale archival tool could perfectly archive a deferred representation [52]. This is due to the difference between a browser’s capability to execute JavaScript and a crawler’s inability to execute JavaScript and discover the embedded resources required to build deferred representations. Further, even if archives could create mementos of deferred representations, they may not be identifiable in the archives [145].

During this investigation, we defined the term *deferred representations* to refer to client-side representations that make use of JavaScript and Ajax to load embedded resources after an initial page load, leading to reduced archivability.

RQ2. How do we measure memento quality? Before discussing how archives can produce better quality mementos of deferred representations, we constructed a metric to quantitatively measure a previously qualitative human assessment (or a misleading calculated metric M_m) of memento quality [51, 50] (Chapter 6). We constructed a metric (D_m) that more closely assesses the quality of a memento according to human perception than current automated metric (M_m). We use this metric to understand how archival quality has changed over time, how different archival services perform relative to one another, and how deferred representations impact the quality of an archive. Using D_m , we demonstrate that deferred representations lead to reduced quality mementos.

RQ3. How can we crawl, archive, and play back deferred representations? The culmination of this body of work is a framework for archiving deferred representations (Chapter 7.2). An intuitive solution that closes the gap between the browser’s and crawler’s capability to handle JavaScript is to use a technology that offers JavaScript support during the crawl [48]. However, we measured the performance trade-offs between a headless browsing client (PhantomJS) and current archival tools (wget and Heritrix) to demonstrate that PhantomJS runs much more slowly than Heritrix and wget but uncovers more embedded resources [57] (Chapter 7).

To limit the runtime penalty, we defined a classifier to only use PhantomJS to

load deferred representations – essentially using PhantomJS for those resources that require a headless browsing client and allowing Heritrix to more quickly crawl those that do not need a headless browsing client. This increases the frontier size but also the overall crawl time.

As part of this work, we define *descendants* as client-side states in the FSM of deferred representations reached by client-side interactions or events that have unique sets of embedded resources. We measured the number of descendants and paths on a client showed that they are only two levels deep and that can be discovered by a crawler equipped with a headless browsing client [58]. From our experimentation, we show that descendants are largely unarchived, significantly increase the crawl time and crawl frontier of a crawler, and crawlers can use JSON metadata to store information about the descendants in WARC files.

This body of work provides a two-tiered crawling approach to archive deferred representations, increase the quality of the archives, and mitigate the impact of JavaScript on mementos.

8.2 FUTURE WORK

Web users will continue to expect the Web and its technologies to advance, become more interactive, and better service their needs. As such, we expect the adoption of JavaScript to continue to increase in the near future. Without adopting a framework for archiving second-order expressions (i.e., representations and their context, including deferred representations and their descendants) and a method for replaying deferred representations, second-order expression will continue to be absent in the archives. Examples of content that will be missing without a mechanism to archive second-order expression include comments on YouTube videos or news articles. Our future work focuses on helping archivists implement our two-tiered crawling approach and understand how the archives can use it to archive deferred representations.

Our future work will incorporate PhantomJS or another headless browsing client into a Web-scale crawler to measure the actual benefits and increased archival coverage realized when crawling deferred representations with a two-tiered crawling approach. We will also work to develop an approach to solve our current edge cases (Chapter 7), including a way to handle applications like mapping applications using our automated approach along with an approach using “canned interactions.” Our goal is to understand how many executions of canned interactions are necessary to

uncover an acceptable threshold of embedded resources (e.g., how many scrolls, pans, and zooms to get the Google Maps tiles for all of Norfolk, VA, USA?). As part of this research, we will study the overlap between the embedded resources in descendants; for example, how many interactions or URI-Rs should we crawl to get all ads served by an ad server? This is effectively the Halting problem for Web archiving.

Similar to our effort of using canned interaction, it would be useful to model user behavior to understand the most likely interactions a user is likely to perform. This will help identify the most likely embedded resources required from a user interaction, as well as the highest value targets for a crawler. Similarly, using a user interaction model, we can perform navigation paths through descendants in the archives using existing mementos (similar to how one might use existing videos to perform video game walk-throughs). We will also investigate using personalized representations from user-archived mementos (e.g., via WARCreate) to improve coverage of deferred representations [144].

8.3 CONTRIBUTIONS

In the below enumerated list, we present the contributions of this dissertation.

1. To better discuss the archival challenges with resources that leverage JavaScript to load embedded resources, we coined the terms *zombie resources*, *deferred representations*, and *descendants*.
2. We identified examples of zombie resources in the archives and demonstrated the social importance of properly archiving deferred representations.
3. Our work measured the missing embedded resources in the archives over time and measured the correlation between the increased missing embedded resources and JavaScript adoption.
4. During our work to archived deferred representations, we constructed a metric (D_m) that uses relative importance of resources to automatically measure the quality of embedded mementos more closely to user perception than the simple measure of completeness (M_m).
5. We used Mechanical Turkers to prove that D_m is an improvement upon M_m .

6. In this dissertation, we showed that resources that do not use JavaScript to load embedded resources have more complete and higher quality mementos.
7. We showed that Heritrix does not archive JavaScript-dependent resources, and that PhantomJS – while it crawls much more slowly – creates a much larger crawl frontier, leading to more complete mementos.
8. Our work demonstrated that descendants are not well archived, but are shallow with only two levels of descendants. From this finding, we propose two crawl policies for archiving descendants.
9. This dissertation culminates in a two-tiered crawling approach for archiving deferred representations and proposed metadata to include in WARC to identify the descendants.

8.4 CONCLUSIONS

Any discussion of the 21st century will have to include the Web because of its importance, prevalence, and ubiquity in today’s society and culture. As such, Web archiving will continue to grow in importance as events such as the SOPA protest manifest themselves solely on the Web. The goal of our research presented in this dissertation is to ensure that historical events manifesting themselves on the Web, when recalled as we do the September 11th terrorist attack through newspaper articles and mementos of CNN.com, create an emotive response rather than a merely factual recount. Our two-tiered crawling approach can help ensure these important events are preserved – as they appear at time of viewing – for future generations, rather than lost due to a technological mismatch between Web browsers and Web crawlers.

The context in which today’s users view Web resources is also important to a resource’s understanding, and therefore to its replay as a memento. Second-order expressions are increasingly unarchivable, but our two-tiered crawling approach can help reverse this trend. As descendants of deferred representations become increasingly prevalent, their archiving to preserve second-order expression becomes equally important. Archiving descendants will improve the archives’ ability to replay deferred representations and allow users to interact with mementos in the archives without

losing functionality. In aggregate, descendants create the equivalent of a joiner memento with multiple perspectives and smaller snapshots creating a complete picture of a deferred representation.

In this dissertation, we have measured the current impact JavaScript has on the archives due to the archival tools' inability to automatically crawl deferred representations. As such, there is currently an inability to archive deferred representations using the current Web-scale archival toolsets. We define the term *deferred representation* to describe the representations that are not fully built until after JavaScript executes and loads additional embedded resources. We demonstrate that the increased adoption of JavaScript leads to more missing embedded resources in the archives.

To better measure the impact – as perceived by humans – of JavaScript on the archives, we constructed an metric to more closely assess the quality of the archives than the traditional metrics associated with completeness. We measure the correlation between human perception and our metric, measuring the improved correlation between human perception and our metric over the traditional metric of completeness. Using our metric, we demonstrate that resources that rely on JavaScript to load embedded resources have lower quality mementos than do resources that do not use JavaScript to load embedded resources. That is, when archived, deferred representations are of lower quality than nondeferred representations.

We propose a two-tiered crawling approach that leverages PhantomJS (or other headless browsing utility) to crawl deferred representations, and present a classifier that can limit the negative performance impacts of PhantomJS on the automatic archival tools (e.g., Heritrix). We show that this will add many more URI-Rs to the crawl frontier, but will also crawl much more slowly. We also define a FSM for describing descendants, define a model for crawling descendants, and show that new embedded resources needed to build descendants are nearly entirely unarchived.

Using the two-tiered crawling framework that we proposed, the archives will increase their coverage of deferred representations and their embedded resources, particularly the resources required for descendants. This will provide higher quality mementos and help preserve the Web as experienced by Web users (rather than by Web crawlers) for future generations.

BIBLIOGRAPHY

- [1] E. Abgrall, Y. L. Traon, M. Monperrus, S. Gombault, M. Heiderich, and A. Ribault. XSS-FP: Browser Fingerprinting using HTML Parser Quirks. Technical Report arXiv:1211.4812, Université du Luxembourg, 2012.
- [2] Access Board. The Rehabilitation Act Amendments (Section 508). <http://www.access-board.gov/sec508/guide/act.htm>, 1998.
- [3] E. Adar, J. Teevan, S. Dumais, and J. Elsas. The Web changes everything: Understanding the dynamics of Web content. In *Proceedings of the 2nd ACM International Conference on Web Search and Data Mining*, pages 282–291, 2009.
- [4] Adobe. Create panoramic images with Photomerge. http://help.adobe.com/en_US/photoshop/cs/using/WSfd1234e1c4b69f30ea53e41001031ab64-75e8a.html, 2013.
- [5] S. Ainsworth, A. Alsum, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. How much of the Web is archived? In *Proceedings of the 11th ACM/IEEE Joint Conference on Digital Libraries*, pages 133–136, 2011.
- [6] S. Ainsworth, M. L. Nelson, and H. Van de Sompel. A framework for evaluation of composite memento temporal coherence. Technical Report arXiv:1402.0928, 2014.
- [7] S. G. Ainsworth and M. L. Nelson. Evaluating sliding and sticky target policies by measuring temporal drift in acyclic walks through a web archive. *International Journal on Digital Libraries*, 16(2):129–144, 2014.
- [8] S. Alam. IIPC General Assembly 2015 Trip Report. <http://ws-dl.blogspot.com/2015/05/2015-05-09-iipc-general-assembly-2015.html>, 2015.
- [9] A. AlSum. Harvesting and preserving the future web: Replay and scale challenges. <http://ws-dl.blogspot.com/2012/05/2012-04-30-iipc-2012-ga-week-with.html>, 2012.
- [10] A. AlSum. mcurl - Command Line Memento Client. <http://ws-dl.blogspot.com/2013/05/2013-05-29-mcurl-command-line-memento.html>, 2013.

- [11] A. AlSum and M. L. Nelson. Thumbnail Summarization Techniques for Web Archives. In *Proceedings of 36th European Conference on Information Retrieval Research*, pages 299–310, 2014.
- [12] K. Amiri, S. Park, R. Tewari, and S. Padmanabhan. DBProxy: A dynamic data cache for Web applications. In *Proceedings of the 19th International Conference on Data Engineering*, pages 1–11, 2003.
- [13] A. Anand, S. Bedathur, K. Berberich, R. Schenkel, and C. Tryfonopoulos. EverLast: a distributed architecture for preserving the web. In *Proceedings of the 9th ACM/IEEE Joint Conference on Digital Libraries*, pages 331–340, 2009.
- [14] S. Andrica and G. Candea. WaRR: A tool for high-fidelity web application record and replay. In *Proceedings of the 41st IEEE/IFIP International Conference on Dependable Systems Networks*, pages 403–410, 2011.
- [15] A. Ankolekar, M. Krötzsch, T. Tran, and D. Vrandecic. The two cultures: Mashing up web 2.0 and the semantic web. In *Proceedings of the 16th International Conference on World Wide Web*, pages 825–834, 2007.
- [16] D. Antoniadis, I. Polakis, G. Kontaxis, E. Athanasopoulos, S. Ioannidis, E. P. Markatos, and T. Karagiannis. we.b: The web of short URLs. In *Proceedings of the 20th International Conference on World Wide Web*, pages 715–724, 2011.
- [17] N. Aranda. A brief history of mobile computing, 2013. <http://ezinearticles.com/?A-Brief-History-of-Mobile-Computing&id=505215>.
- [18] Archive.is. Archive.is. <http://archive.is/>, 2013.
- [19] P. Ast, M. Kapfenberger, and S. Hauswiesner. Crawler Approaches And Technology. Technical Report <http://www.iicm.tugraz.at/cguetl/courses/isr/uearchive/uews2008/Ue01%20-%20Crawler-Approaches-And-Technology.pdf>, Graz University of Technology, 2008.

- [20] V. Banos and Y. Manolopoulos. A Quantitative Approach to Evaluate Website Archivability Using the CLEAR+ Method. *International Journal on Digital Libraries*, pages 1–23, 2015.
- [21] V. Banos, K. Yunhyong, S. Ross, and Y. Manolopoulos. CLEAR: a credible method to evaluate website archivability. In *Proceedings of the 9th International Conference on Preservation of Digital Objects*, 2013.
- [22] Z. Bar-Yossef, A. Z. Broder, R. Kumar, and A. Tomkins. Sic transit gloria telae: towards an understanding of the web’s decay. In *Proceedings of the 13th International Conference on World Wide Web*, pages 328–337, 2004.
- [23] A. Bartoli, E. Medvet, and M. Mauri. Recording and replaying navigations on ajax web sites. In *Web Engineering*, volume 7387 of *Lecture Notes in Computer Science*, pages 370–377. 2012.
- [24] A. Bartoli, E. Medvet, and M. Mauri. A tool for registering and replaying web navigation. In *Proceedings of the 2nd International Conference on Information Society*, pages 509–510, 2012.
- [25] J. Behr, P. Eschler, Y. Jung, and M. Zöllner. X3DOM: a DOM-based HTML5/X3D integration model. In *Proceedings of the 14th International Conference on 3D Web Technology*, pages 127–135, 2009.
- [26] M. Ben Saad and S. Gançarski. Archiving the web using page changes patterns: A case study. In *Proceedings of the 11th ACM/IEEE Joint Conference on Digital Libraries*, pages 113–122, 2011.
- [27] M. Ben Saad and S. Gançarski. Archiving the web using page changes patterns: a case study. *International Journal on Digital Libraries*, 13(1):33–49, 2012.
- [28] M. Ben Saad, Z. Pehlivan, and S. Gançarski. Coherence-oriented crawling and navigation using patterns for web archives. In *Proceedings of the 1st International Conference on Theory and Practice of Digital Libraries*, pages 421–433, 2011.
- [29] S. Benford and G. Giannachi. Temporal trajectories in shared interactive narratives. In *Proceeding of the 26th Annual SIGCHI Conference on Human Factors in Computing Systems*, pages 73–82, 2008.

- [30] K. Benjamin, G. von Bochmann, M. Dincturk, G.-V. Jourdan, and I. Onut. A Strategy for Efficient Crawling of Rich Internet Applications. In *Web Engineering*, volume 6757 of *Lecture Notes in Computer Science*, pages 74–89, 2011.
- [31] E. Benson, A. Marcus, D. Karger, and S. Madden. Sync kit: a persistent client-side database caching toolkit for data intensive websites. In *Proceedings of the 19th International Conference on World Wide Web*, pages 121–130, 2010.
- [32] M. K. Bergman. Deep web: Surfacing hidden value. *Journal of Electronic Publishing*.
- [33] R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, E. O’Connor, and S. Pfeiffer. Html 5.1 nightly: History interface. <http://www.w3.org/html/wg/drafts/html/master/single-page.html#history-1>, 2013.
- [34] T. Berners-Lee. Information management: A proposal. <http://www.w3.org/History/1989/proposal.html>, 1990.
- [35] T. Berners-Lee. Generic Resources. <http://www.w3.org/DesignIssues/Generic>, 1996.
- [36] T. Berners-Lee. What do HTTP URIs Identify? <http://www.w3.org/DesignIssues/HTTP-URI.html>, 2002.
- [37] T. Berners-Lee, R. Fielding, and L. Masinter. RFC2396: Uniform Resource Identifiers (URI): Generic Syntax. <http://www.rfc-base.org/rfc-2396.html>, 1998.
- [38] T. Berners-Lee, R. Fielding, and L. Masinter. RFC3986: Uniform Resource Identifier (URI): Generic Syntax. <http://www.ietf.org/rfc/rfc3986.txt>, 2005.
- [39] P. L. Bogen, II, D. Pogue, F. Poursardar, Y. Li, R. Furuta, and F. Shipman. WPv4: a re-imagined Walden’s paths to support diverse user communities. In *Proceeding of the 11th ACM/IEEE Joint Conference on Digital Libraries*, pages 419–420, 2011.

- [40] M. Bossetta and A. D. Segesten. Tracing Eurosceptic Party Networks via Hyperlink Network Analysis and #FAIL!ng: Can Web Crawlers Keep up with Web Design? In *#FAIL! Workshop at the Websci'15 Conference*, 2015.
- [41] M. N. K. Boulos, J. Gong, P. Yue, and J. Y. Warren. Web GIS in practice VIII: HTML5 and the canvas element for interactive online mapping. *International Journal of Health Geographics*, 9(14):1–13, March 2010.
- [42] M. Bragg and S. Rollason-Cass. Archiving Social Networking Sites w/ Archive-It. <https://webarchive.jira.com/wiki/pages/viewpage.action?pageId=3113092>, 2014.
- [43] B. Brewington, G. Cybenko, D. Coll, and N. Hanover. Keeping up with the changing Web. *IEEE Computer*, 33(5):52–58, 2000.
- [44] A. Bright. Web evidence points to pro-Russia rebels in downing of MH17. <http://www.csmonitor.com/World/Europe/2014/0717/Web-evidence-points-to-pro-Russia-rebels-in-downing-of-MH17-video>, 2014.
- [45] J. F. Brunelle. Replaying the SOPA Protest. <http://ws-dl.blogspot.com/2013/11/2013-11-28-replaying-sopa-protest.html>, November 2013.
- [46] J. F. Brunelle. Google and JavaScript. <http://ws-dl.blogspot.com/2014/06/2014-06-18-google-and-javascript.html>, 2014.
- [47] J. F. Brunelle. Fixing Links on the Live Web, Breaking Them in the Archive. <http://ws-dl.blogspot.com/2015/02/2015-02-17-fixing-links-on-live-web.html>, 2015.
- [48] J. F. Brunelle. PhantomJS+VisualEvent or Selenium for Web Archiving? <http://ws-dl.blogspot.com/2015/06/2015-06-26-phantomjsvisualevent-or.html>, 2015.
- [49] J. F. Brunelle, G. T. Jackson, K. Dempsey, C. Boonthum, I. B. Levinstein, and D. S. McNamara. Game-Based iSTART Practice: From MiBoard to Self-Explanation Showdown. In *Proceedings of the 23rd International FLAIRS Conference*, pages 480–485, 2010.

- [50] J. F. Brunelle, M. Kelly, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. Not All Mementos Are Created Equal: Measuring The Impact Of Missing Resources. In *Proceedings of the 14th ACM/IEEE Joint Conference on Digital Libraries*, pages 321 – 330, 2014.
- [51] J. F. Brunelle, M. Kelly, H. SalahEldeen, M. C. Weigle, and M. L. Nelson. Not All Mementos Are Created Equal: Measuring The Impact Of Missing Resources. *International Journal of Digital Libraries*, 16(3):283–301, 2015.
- [52] J. F. Brunelle, M. Kelly, M. C. Weigle, and M. L. Nelson. The Impact of JavaScript on Archivability. *International Journal on Digital Libraries*, 2015.
- [53] J. F. Brunelle, I. B. Levinstein, and C. Boonthum. MiBoard: Metacognitive Training Through Gaming in iSTART. In *Proceedings of the 1st VMASC Capstone Conference*, page 10, 2009.
- [54] J. F. Brunelle and M. L. Nelson. Zombies in the archives. <http://ws-dl.blogspot.com/2012/10/2012-10-10-zombies-in-archives.html>, 2012.
- [55] J. F. Brunelle and M. L. Nelson. An Evaluation of Caching Policies for Memento TimeMaps. In *Proceedings of the 13th ACM/IEEE Joint Conference on Digital Libraries*, pages 267–276, 2013.
- [56] J. F. Brunelle, M. L. Nelson, L. Balakireva, R. Sanderson, and H. Van de Sompel. Evaluating the SiteStory Transactional Web Archive With the ApacheBench Tool. In *Proceedings of the 3rd International Conference on Theory and Practice of Digital Libraries*, pages 204–215. 2013.
- [57] J. F. Brunelle, M. C. Weigle, and M. L. Nelson. Archiving Deferred Representations Using a Two-Tiered Crawling Approach. In *Proceedings of 12th International Conference on Preservation of Digital Objects*, 2015.
- [58] J. F. Brunelle, M. C. Weigle, and M. L. Nelson. Adapting the Hypercube Model to Archive Deferred Representations at Web-Scale. Technical Report arXiv:1601.05142, 2016.
- [59] E. P. Bucy, A. Lang, R. F. Potter, and M. E. Grabe. Formal features of cyberspace: Relationships between web page complexity and site traffic. *Journal*

of the American Society for Information Science, 50(13):1246 – 1256, November 1999.

- [60] P. Cao, J. Zhang, and K. Beach. Active cache: caching dynamic contents on the web. In *Proceedings of the 1st IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 373–388, 1998.
- [61] S. Chakrabarti, S. Srivastava, M. Subramanyam, and M. Tiwari. Memex: A browsing assistant for collaborative archiving and mining of surf trails. In *Proceedings of the Very Large Database Endowment*, 2000.
- [62] K. Z. Chen, G. Gu, J. Zhuge, J. Nazario, and X. Han. Webpatrol: automated collection and replay of web-based malware scenarios. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 186–195, 2011.
- [63] C. Chi, M. X. Zhou, M. Yang, W. Xiao, Y. Yu, and X. Sun. Dandelion: supporting coordinated, collaborative authoring in Wikis. In *Proceedings of the 28th International Conference on Human factors in Computing Systems*, pages 1199–1202, 2010.
- [64] W. Chisholm, G. Vanderheiden, and I. Jacobs. Web Content Accessibility Guidelines 1.0. *Interactions*, 8(4):35–54, 2001.
- [65] J. Cho and H. Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Data Bases*, pages 200–209, 2000.
- [66] J. Cho and H. Garcia-Molina. Estimating frequency of change. *ACM Transactions on Internet Technology*, 3(3):256–290, 2003.
- [67] S. R. Choudhary and A. Orso. Automated client-side monitoring for web applications. In *Proceedings of the 3rd IEEE International Conference on Software Testing, Verification and Validation Workshops*, pages 303–306, 2009.
- [68] S. R. Choudhary, H. Versee, and A. Orso. A cross-browser web application testing tool. In *Proceedings of 28th IEEE International Conference on Software Maintenance*, pages 1–6, 2010.

- [69] S. Chung and A. Pereira. Timed Petri net representation of the Synchronized Multimedia Integration Language (SMIL) of XML. In *Proceedings of 9th IEEE International Conference on Information Technology: Coding and Computing*, pages 711 – 716, 2003.
- [70] J. Conallen. Modeling Web application architectures with UML. *Communications of the ACM*, 42(10):63–70, 1999.
- [71] R. G. Coram. django-phantomjs. <https://github.com/ukwa/django-phantomjs>, 2014.
- [72] E. Crook. Web archiving in a Web 2.0 world. In *Proceedings of the 8th Australian Library and Information Association Biennial Conference*, pages 1–9, 2008.
- [73] A. Cui, L. Yang, D. Hou, M.-Y. Kan, Y. Liu, M. Zhang, and S. Ma. PrEV: Preservation Explorer and Vault for Web 2.0 User-Generated Content. In *Proceedings of the 12th International Conference on Theory and Practice of Digital Libraries*, volume 7489 of *Lecture Notes in Computer Science*, pages 101–112. 2012.
- [74] P. Dave, P. L. Bogen, II, U. P. Karadkar, L. Francisco-Revilla, R. Furuta, and F. Shipman. Dynamically growing hypertext collections. In *Proceedings of the 15th ACM Conference on Hypertext and Hypermedia*, pages 171–180, 2004.
- [75] David Hockney. David Hockney. <http://churchstreetart.net/wp-content/uploads/2011/12/Yosemite-David-Hockney.jpg>, 2013.
- [76] R. C. Davis. Five tips for designing preservable websites. <http://blog.photography.si.edu/2011/08/02/five-tips-for-designing-preserved-websites/>, August 2011.
- [77] D. Denev, A. Mazeika, M. Spaniol, and G. Weikum. SHARC: framework for quality-conscious web archiving. In *Proceedings of the 35th International Conference on Very Large Data Bases*, pages 586–597, 2009.

- [78] M. Dhawan and V. Ganapathy. Analyzing information flow in JavaScript-based browser extensions. In *Proceedings of the 1st Annual Computer Security Applications Conference*, pages 382 – 391, 2009.
- [79] M. E. Dincturk. Model-based Crawling - An Approach to Design Efficient Crawling Strategies for Rich Internet Applications. Ph.D. Dissertation, University of Ottawa, 2013.
- [80] M. E. Dincturk, G.-V. Jourdan, G. V. Bochmann, and I. V. Onut. A Model-Based Approach for Crawling Rich Internet Applications. *ACM Transactions on the Web*, 8(3):1–39, 2014.
- [81] F. Douglass, A. Feldmann, B. Krishnamurthy, and J. Mogul. Rate of change and other metrics: a live study of the World Wide Web. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems on USENIX Symposium on Internet Technologies and Systems*, page 14, 1997.
- [82] C. Duda, G. Frey, D. Kossmann, and C. Zhou. AjaxSearch: crawling, indexing and searching Web 2.0 applications. In *Proceedings of the Very Large Database Endowment*, pages 1440–1443, 2008.
- [83] C. E. Dyreson, H.-l. Lin, and Y. Wang. Managing versions of Web documents in a transaction-time Web server. In *Proceedings of the 13th International Conference on World Wide Web*, pages 422–432, 2004.
- [84] R. Ewing. Wayback Machine, Use in Proxy Mode. <https://webarchive.jira.com/wiki/display/ARIH/Wayback+Machine,+Use+in+Proxy+Mode>, 2015.
- [85] G. Eysenbach and M. Trudel. Going, going, still there: using the WebCite service to permanently archive cited web pages. *Journal of Medical Internet Research*, 7(5):919, 2005.
- [86] D. A. Fahrenthold. SOPA protests shut down Web sites. http://www.washingtonpost.com/politics/sopa-protests-to-shut-down-web-sites/2012/01/17/gIQA4WYl6P_story.html, January 2012.

- [87] T. Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, 2006.
- [88] K. D. Fenstermacher and M. Ginsburg. Mining client-side activity for personalization. In *Proceedings of the 4th IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, pages 205–212, 2002.
- [89] K. D. Fenstermacher and M. Ginsburg. Client-side monitoring for web mining. *Journal of the American Society for Information Science and Technology*, 54(7):625–637, 2003.
- [90] E. Fersini, E. Messina, and F. Archetti. Enhancing web page classification through image-block importance analysis. *Information Processing & Management*, 44(4):1431 – 1447, 2008.
- [91] Fetch Technologies. Internet Data Opportunity: Utilizing New Sources of Real-Time Data to Stay Competitive and Informed. <http://www.fetch.com/wp-content/uploads/2011/01/Whitepaper.pdf>, 2012.
- [92] D. Fetterly, M. Manasse, M. Najork, and J. Wiener. A large-scale study of the evolution of web pages. *Software: Practice and Experience*, 34(2):213–237, 2004.
- [93] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616 Hypertext Transfer Protocol. <http://tools.ietf.org/html/rfc2616>, 1999.
- [94] R. T. Fielding and R. N. Taylor. Principled design of the modern web architecture. *ACM Transactions on Internet Technology*, 2:115–150, 2002.
- [95] Firefox. Firefox. <http://www.mozilla.org/en-US/firefox/new/>, 2013.
- [96] J. E. Fischer. Studying and tackling temporal challenges in mobile HCI. In *Proceedings of the 28th of the International Conference on Human Factors in Computing Systems*, pages 2927–2930, 2010.
- [97] K. Fitch. Web site archiving: an approach to recording every materially different response produced by a Website. In *Proceedings of the 9th Australasian World Wide Web Conference*, pages 5–9, 2003.

- [98] D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, 2001.
- [99] B. Fleiss. SEO in the Web 2.0 Era: The Evolution of Search Engine Optimization. <http://www.bkv.com/redpapers-media/SEO-in-the-Web-2.0-Era.pdf>, 2007.
- [100] L. Francisco-Revilla, F. Shipman, R. Furuta, U. Karadkar, and A. Arora. Managing change on the web. In *Proceedings of the 1st ACM/IEEE Joint Conference on Digital Libraries*, pages 67–76, 2001.
- [101] L. S. Fuhrig. The Smithsonian: Using and Archiving Facebook. <http://blog.photography.si.edu/2011/05/31/smithsonian-using-and-archiving-facebook/>, May 2011.
- [102] R. Furuta, F. M. Shipman, III, C. C. Marshall, D. Brenner, and H.-w. Hsieh. Hypertext paths and the World-Wide Web: experiences with Walden's Paths. In *Proceedings of the 8th ACM conference on Hypertext*, pages 167–176, 1997.
- [103] J. J. Garrett. Ajax: A New Approach to Web Applications. <http://www.adaptivepath.com/ideas/ajax-new-approach-web-applications>, 2005.
- [104] GNU. Introduction to GNU Wget. <http://www.gnu.org/software/wget/>, 2013.
- [105] J. Goettsch. py-wayback. <https://bitbucket.org/jgoettsch/py-wayback>, 2015.
- [106] Google. Getting Started with Ajax Crawling. <https://developers.google.com/webmasters/ajax-crawling/docs/getting-started>, 2012.
- [107] Google. Making AJAX Applications Crawlable. <https://developers.google.com/webmasters/ajax-crawling/>, 2012.
- [108] Google. AJAX crawling: Guide for webmasters and developers. <http://support.google.com/webmasters/bin/answer.py?hl=en&answer=174992>, 2013.
- [109] Google. Google chrome. <http://www.google.com/chrome>, 2013.
- [110] Google. Google Maps, 2013. <http://maps.google.com/>.

- [111] Google Web Toolkit. GBST Uses Google Web Toolkit to Improve Productivity and Create a Rich User Experience. <http://google-web-toolkit.googlecode.com/files/CaseStudy-GBST-Uses-GWT.pdf>, January 2011.
- [112] P. Gorniak and D. Poole. Predicting future user actions by observing unmodified applications. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence*, pages 217–222, 2000.
- [113] G. Gray and S. Martin. Choosing a sustainable web archiving method: A comparison of capture quality. *D-Lib Magazine*, 19(5), May 2013.
- [114] B. L. Gross and J. N. Sheth. Time-Oriented Advertising: A Content Analysis of United States Magazine Advertising, 1890-1988. *Journal of Marketing*, 53(4):76–83, 1989.
- [115] A. Gulli and A. Signorini. The indexable web is more than 11.5 billion pages. In *Special Interest Tracks and Posters of the 14th International Conference on World Wide Web*, pages 902–903, 2005.
- [116] K. Gyllstrom, C. Eickhoff, A. P. de Vries, and M.-F. Moens. The downside of markup: Examining the harmful effects of css and javascript on indexing today’s web. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, pages 1990–1994, 2012.
- [117] L. Ha and E. L. James. Interactivity reexamined: A baseline analysis of early business web sites. *Journal of Broadcasting & Electronic Media*, 42(4):457–474, 1998.
- [118] S. Hackett, B. Parmanto, and X. Zeng. Accessibility of Internet Websites Through Time. *Proceedings of the 6th International ACM SIGACCESS Conference on Computers and Accessibility*, (77-78):32–39, 2003.
- [119] K. Hagedorn and J. Sentelli. Google Still Not Indexing Hidden Web URLs. *D-Lib Magazine*, 14(7), August 2008.
- [120] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten.

- The WEKA Data Mining Software: An Update. In *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining Explorations*, pages 10–18, 2009.
- [121] O. Hallarakker and G. Vigna. Detecting malicious JavaScript code in mozilla. In *Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems*, pages 85–94, 2005.
 - [122] Hanzo Archives. Hanzo: Authenticity Delivered. <http://www.hanzoarchives.com/>, 2015.
 - [123] D. Hauger. Enhancing user models by client-side user-monitoring. In *Proceedings of the UMAP-09 Workshop on Personalization in Mobile and Pervasive Computing*, 2009.
 - [124] D. Hauger. Using asynchronous client-side user monitoring to enhance user modeling in adaptive e-learning systems. In *Proceedings of the UMAP-09 Workshop on Personalization in Mobile and Pervasive Computing*, 2009.
 - [125] B. He, M. Patel, Z. Zhang, and K. C.-C. Chang. Accessing the Deep Web. *Communications of the ACM*, 50(5):94–101, 2007.
 - [126] Y. He, D. Xin, V. Ganti, S. Rajaraman, and N. Shah. Crawling Deep Web Entity Pages. In *Proceedings of the 6th ACM International Conference on Web Search and Data Mining*, pages 355–364, 2013.
 - [127] J. I. Hong, J. Heer, S. Waterson, and J. A. Landay. Webquilt: A proxy-based approach to remote web usability testing. *ACM Transactions on Information Systems*, 19(3):263–285, 2001.
 - [128] J. I. Hong and J. A. Landay. WebQuilt: a framework for capturing and visualizing the web experience. In *Proceedings of the 10th International Conference on World Wide Web*, pages 717–724, 2001.
 - [129] Hugin. Hugin - Panorama photo stitcher. <http://hugin.sourceforge.net/>, 2013.
 - [130] S. Ihm and V. S. Pai. Towards understanding modern web traffic. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, pages 295–312, 2011.

- [131] IIPC. Minutes of the WARC revision workshop. <http://iipc.github.io/warc-specifications/specifications/warc-format/meetings/2015-05-01-IIPC-GA-WARC-Meeting-Minutes/>, 2015.
- [132] IIPC. Proposal for Standardizing the Recording Rendered Targets. <http://nlevitt.github.io/warc-specifications/specifications/warc-rendered-targets/recording-screenshots.html>, 2015.
- [133] P. Jack. Extractorhtml extract-javascript. <https://webarchive.jira.com/wiki/display/Heritrix/ExtractorHTML+extract-javascript>, 2014.
- [134] A. Jackson. Archiving the Dynamic Web. <http://web.archive.org/web/20150217044640/http://anjackson.github.io/keeping-codes/practice/archiving-the-dynamic-web.html>, 2013.
- [135] I. Jacobs and N. Walsh. Architecture of the world wide web, volume one. Technical Report W3C Recommendation 15 December 2004, W3C, 2004.
- [136] A. Jindal, C. Crutchfield, S. Goel, R. Kolluri, and R. Jain. The mobile web is structurally different. In *INFOCOM Workshops 2008*, pages 1–6, 2008.
- [137] W. Jordan, M. Kelly, J. F. Brunelle, L. Vobrak, M. C. Weigle, and M. L. Nelson. Mobile Mink: Merging Mobile and Desktop Archived Webs. In *Proceedings of the 15th ACM/IEEE Joint Conference on Digital Libraries*, pages 243–244, 2015.
- [138] Kapow Software. Kapow Software. <http://kapowsoftware.com/>, 2013.
- [139] U. Karadkar, L. Francisco-Revilla, R. Furuta, H. wei Hsieh, and F. Shipman. Evolution of the Walden’s Paths Authoring Tools. In *Proceedings of WebNet World Conference on the World Wide Web and Internet*, pages 299–304, 2000.
- [140] R. Karri. Client-side page element web-caching. http://scholarworks.sjsu.edu/etd_projects/137, December 2009.
- [141] M. Kelly. An Extensible Framework For Creating Personal Archives Of Web Resources Requiring Authentication. Master’s thesis, Old Dominion University, 2012.
- [142] M. Kelly. WARCreate. <http://warcreate.com/>, 2013.

- [143] M. Kelly. WARCreate and WAIL: WARC, Wayback and Heritrix Made Easy. <http://ws-dl.blogspot.com/2013/07/2013-07-10-warcreate-and-wail-war.html>, July 2013.
- [144] M. Kelly. A Framework for Aggregating Private and Public Web Archives. *Bulletin of the IEEE TCDL*, 11(3), 2015.
- [145] M. Kelly, J. F. Brunelle, M. C. Weigle, and M. L. Nelson. A method for identifying dynamic representations in the archives. *D-Lib Magazine*, 19(11/12), November/December 2013.
- [146] M. Kelly, J. F. Brunelle, M. C. Weigle, and M. L. Nelson. On the Change in Archivability of Websites Over Time. In *Proceedings of the 3rd International Conference on Theory and Practice of Digital Libraries*, pages 35–47, 2013.
- [147] M. Kelly, M. L. Nelson, and M. C. Weigle. Mink: Integrating the Live and Archived Web Viewing Experience Using Web Browsers and Memento. In *Proceedings of the 14th ACM/IEEE Joint Conference on Digital Libraries*, pages 469–470, 2014.
- [148] M. Kelly, M. L. Nelson, and M. C. Weigle. The archival acid test: Evaluating archive performance on advanced HTML and JavaScript. In *Proceedings of the 14th ACM/IEEE Joint Conference on Digital Libraries*, pages 25 – 28, 2014.
- [149] M. Kelly and M. C. Weigle. WARCreate - Create Wayback-Consumable WARC Files from Any Webpage. In *Proceedings of the 12th ACM/IEEE Joint Conference on Digital Libraries*, pages 437–438, 2012.
- [150] A. R. Kenney, N. Y. McGovern, P. Botticelli, R. Entlich, C. Lagoze, and S. Payette. Preservation risk management for web resources. *D-Lib Magazine*, 8(1), January 2002.
- [151] E. Kiciman and B. Livshits. AjaxScope: A Platform for Remotely Monitoring the Client-Side Behavior of Web 2.0 Applications. In *The 21st ACM Symposium on Operating Systems Principles*, pages 17–30, 2007.
- [152] A. Kittur, E. H. Chi, and B. Suh. Crowdsourcing user studies with mechanical turk. In *Proceedings of the 26th SIGCHI Conference on Human Factors in Computing Systems*, pages 453–456, 2008.

- [153] W. Koehler. Web page change and persistence-a four-year longitudinal study. *Journal of the American Society for Information Science and Technology*, 53(2):162–171, 2002.
- [154] C. Kohlschütter, P. Fankhauser, and W. Nejdl. Boilerplate Detection Using Shallow Text Features. In *Proceedings of the 3rd ACM International Conference on Web Search and Data Mining*, pages 441–450, 2010.
- [155] I. Kreymer. Browsertrix: Browser-Based On-Demand Web Archiving Automation. <https://github.com/ikreymer/browsertrix>, 2015.
- [156] I. Kreymer. Webrecorder.io. <https://webrecorder.io/>, 2015.
- [157] K.Vikram, A. Prateek, and B. Livshits. Ripley: Automatically Securing Web 2.0 Applications Through Replicated Execution. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, pages 173–186, 2009.
- [158] A. LaFrance. Raiders of the Lost Web. <http://www.theatlantic.com/technology/archive/2015/10/raiders-of-the-lost-web/409210/>, 2015.
- [159] J. P. Lage, A. S. da Silva, P. B. Golgher, and A. H. Laender. Automatic generation of agents for collecting hidden web pages for data extraction. *Data & Knowledge Engineering*, 49(2):177–196, 2004.
- [160] C. Lagoze, H. Van de Sompel, P. Johnston, M. Nelson, R. Sanderson, and S. Warner. Open Archives Initiative Object Reuse and Exchange. 2008. <http://www.openarchives.org/ore>.
- [161] J. Lanier. *You Are Not a Gadget: A Manifesto*. Penguin Books, 2010.
- [162] S. Lawrence, D. Pennock, G. Flake, and R. Krovetz. Persistence of web references in scientific research. *Computer*, 34(2):26–31, 2001.
- [163] B. Lazorchak. 2012-04-30: IIPC 2012 GA, A Week with Archivists! <http://blogs.loc.gov/digitalpreservation/2012/06/harvesting-and-preserving-the-future-web-replay-and-scale-challenges/>, 2012.

- [164] P. LePage. Responsive Web Design Basics. <https://developers.google.com/web/fundamentals/layouts/rwd-fundamentals/>, 2015.
- [165] J. Lepore. The Cobweb: Can the Internet be Archived? *The New Yorker*, January 26, 2015.
- [166] I. Li, J. Nichols, T. Lau, C. Drews, and A. Cypher. Here’s what I did: sharing and reusing web activity with ActionShot. In *Proceedings of the 28th SIGCHI Conference on Human Factors in Computing Systems*, pages 723–732, 2010.
- [167] P. Likarish and E. Jung. A targeted web crawling for building malicious javascript collection. In *Proceedings of the ACM 1st International Workshop on Data-Intensive Software Management and Mining*, pages 23–26, 2009.
- [168] L. Lim, M. Wang, S. Padmanabhan, J. Vitter, and R. Agarwal. Characterizing web document change. volume 2118 of *Lecture Notes in Computer Science*, pages 133–144, 2001.
- [169] Linden Research. Second Life. <http://www.secondlife.com>, 2015.
- [170] G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan. Koala: capture, share, automate, personalize business processes on the web. In *Proceedings of the 25th SIGCHI Conference on Human Factors in Computing Systems*, pages 943–946, 2007.
- [171] B. Livshits and S. Guarnieri. Gulfstream: Incremental Static Analysis for Streaming JavaScript Applications. Technical Report MSR-TR-2010-4, Microsoft, January 2010.
- [172] B. Livshits and E. Kiciman. Doloto: Code splitting for network-bound web 2.0 applications. Technical Report MSR-TR-2007-159, Microsoft Research, December 2007.
- [173] D. Lowet and D. Goergen. Co-browsing dynamic web pages. In *Proceeding of the 18th International Conference on World Wide Web*, pages 941–950, 2009.
- [174] J. Madhavan, L. Afanasiev, L. Antova, and A. Y. Halevy. Harnessing the Deep Web: Present and Future. Technical Report arXiv:0909.1785, University of Oxford, 2009.

- [175] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy. Google's Deep Web crawl. In *Proceedings of the Very Large Database Endowment*, pages 1241–1252, 2008.
- [176] G. S. Manku, A. Jain, and A. Das Sarma. Detecting near-duplicates for web crawling. In *Proceedings of the 16th International Conference on World Wide Web*, pages 141–150, 2007.
- [177] J. Maras, J. Carlson, and I. Crnkovi. Extracting client-side web application code. In *Proceedings of the 21st International Conference on World Wide Web*, pages 819–828, 2012.
- [178] C. C. Marshall and F. M. Shipman. On the Institutional Archiving of Social Media. In *Proceedings of the 12th ACM/IEEE Joint Conference on Digital Libraries*, pages 1–10, 2012.
- [179] J. Masanès. *Web Archiving*. Springer, 2006.
- [180] N. Matthijssen, A. Zaidman, M.-A. Storey, I. Bull, and A. van Deursen. Connecting Traces: Understanding Client-Server Interactions in Ajax Applications. In *Proceedings of the IEEE 18th International Conference on Program Comprehension*, pages 216 –225, 2010.
- [181] F. McCown and J. F. Brunelle. Warrick. <http://warrick.cs.odu.edu/>, 2013.
- [182] F. McCown, N. Diawara, and M. L. Nelson. Factors affecting Website reconstruction from the Web infrastructure. In *Proceedings of the 7th ACM/IEEE Joint Conference on Digital Libraries*, pages 39–48, 2007.
- [183] F. McCown, X. Liu, M. L. Nelson, and M. Zubair. Search Engine Coverage of the OAI-PMH Corpus. *IEEE Internet Computing*, 10(2):66–73, 2006.
- [184] F. McCown, C. C. Marshall, and M. L. Nelson. Why Web sites are lost (and how they're sometimes found). *Communications of the ACM*, 52(11):141–145, 2009.
- [185] F. McCown, M. Yarbrough, and K. Enlow. Tools for discovering and archiving the mobile web. *D-Lib Magazine*, 21(3/4), March/April 2015.

- [186] N. Y. McGovern, A. R. Kenney, R. Entlich, W. R. Kehoe, and E. Buckley. Virtual remote control. *D-Lib Magazine*, 10(4), April 2004.
- [187] H. Meng, R. Kommineni, Q. Pham, R. Gardner, T. Malik, and D. Thain. An invariant framework for conducting reproducible computational science. *Journal of Computational Science*, 9:137–142, 2015.
- [188] H. Meng, M. Wolf, P. Ivie, A. Woodard, M. Hildreth, and D. Thain. A Case Study in Preserving a High Energy Physics Application with Parrot. In *Journal of Physics: Conference Series*, pages 1–8, 2015.
- [189] S. Merity. July 2015 Crawl Archive Available. <http://blog.commoncrawl.org/2015/08/july-2015-crawl-archive-available/>, 2015.
- [190] A. Mesbah. Analysis and Testing of Ajax-based Single-page Web Applications. Ph.D. Dissertation, Delft University of Technology, 2009.
- [191] A. Mesbah, E. Bozdag, and A. van Deursen. Crawling Ajax by inferring user interface state changes. In *Proceedings of the 8th International Conference on Web Engineering*, pages 122 –134, 2008.
- [192] A. Mesbah and A. van Deursen. An Architectural Style for Ajax. In *Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture*, pages 1–9, 2007.
- [193] A. Mesbah and A. van Deursen. Migrating multi-page web applications to single-page ajax interfaces. In *Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, pages 181–190, 2007.
- [194] A. Mesbah and A. van Deursen. Invariant-based automatic testing of Ajax user interfaces. In *Proceedings of the 31st International Conference on Software Engineering*, pages 210–220, 2009.
- [195] A. Mesbah, A. van Deursen, and S. Lenselink. Crawling Ajax-Based Web Applications Through Dynamic Analysis of User Interface State Changes. *ACM Transactions on the Web*, 6(1):3:1–3:30, 2012.
- [196] L. A. Meyerovich and B. Livshits. ConScript: Specifying and Enforcing Fine-Grained Security Policies for JavaScript in the Browser. In *Proceedings of the 30th IEEE Symposium on Security and Privacy*, pages 481–496, 2010.

- [197] J. Mickens, J. Elson, and J. Howell. Mugshot: deterministic capture and replay for JavaScript applications. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, page 11, 2010.
- [198] Microsoft. Internet Explorer. <http://windows.microsoft.com/en-US/internet-explorer/products/ie/home>, 2013.
- [199] J. Mills. Hockney Style Joiner, 2013. <http://mhsdigitalimagingmillsj.blogspot.com/2010/12/david-hockney.html>.
- [200] B. Mobasher, R. Cooley, and J. Srivastava. Automatic personalization based on web usage mining. *Communications of the ACM*, 43:142–151, 2000.
- [201] G. Mohr, M. Kimpton, M. Stack, and I. Ranitovic. Introduction to Heritrix, an archival quality web crawler. In *Proceedings of the 4th International Web Archiving Workshop*, 2004.
- [202] D. S. Myers, J. N. Carlisle, J. A. Cowling, and B. H. Liskov. MapJAX: data structure abstractions for asynchronous web applications. In *Proceedings of the 15th USENIX Annual Technical Conference*, pages 1–14, 2007.
- [203] National Archives and Records Administration. NARA Code of Federal Regulations - 36 CFR Subchapter B - Records Management. <http://www.archives.gov/about/regulations/subchapter/b.html>, 2011.
- [204] National Archives and Records Administration. NARA Code of Federal Regulations - 36 CFR Subchapter B Part 1236 – Electronic Records Management. <http://www.archives.gov/about/regulations/part-1236.html>, 2011.
- [205] K. C. Negulescu. Web Archiving @ the Internet Archive. Presentation at the 2010 Digital Preservation Partners Meeting, 2010 <http://www.digitalpreservation.gov/meetings/documents/ndiipp10/NDIIPP072110FinalIA.ppt>.
- [206] M. L. Nelson. Memento-Datetime is not Last-Modified. <http://ws-dl.blogspot.com/2010/11/2010-11-05-memento-datetime-is-not-last.html>, 2011.
- [207] M. L. Nelson. Archive.is Supports Memento. <http://ws-dl.blogspot.com/2013/07/2013-07-09-archiveis-supports-memento.html>, 2013.

- [208] M. L. Nelson. Game Walkthroughs As A Metaphor for Web Preservation. <http://ws-dl.blogspot.com/2013/05/2013-05-25-game-walkthroughs-as.html>, 2013.
- [209] M. L. Nelson. “Refresh” For Zombies, Time Jumps. <http://ws-dl.blogspot.com/2014/07/2014-07-14-refresh-for-zombies-time.html>, 2014.
- [210] @NesbittBrian. Play framework sample application with JWebUnit and synchronous Ajax, 2011. <http://nesbot.com/2011/10/16/play-framework-sample-app-JWebUnit-synchronous-ajax>.
- [211] NetPreserve.org. IIPC Future of the Web Workshop – Introduction & Overview. <http://netpreserve.org/sites/default/files/resources/OverviewFutureWebWorkshop.pdf>, 2012.
- [212] B. Neuberg. Really Simple History. <http://code.google.com/p/reallysimplehistory/wiki/ReallySimpleHistoryLinks>, 2012.
- [213] I. Nino, B. de la Ossa, J. Gil, J. Sahuquillo, and A. Pont. Carena: a tool to capture and replay web navigation sessions. In *Workshop on End-to-End Monitoring Techniques and Services*, pages 127 – 141, 2005.
- [214] A. Ntoulas, P. Zerfos, and J. Cho. Downloading textual hidden web content through keyword queries. In *Proceedings of the 5th ACM/IEEE Joint Conference on Digital Libraries*, pages 100–109, 2005.
- [215] A. Nwala. Carbon dating the web, version 2.0. <http://ws-dl.blogspot.com/2014/11/2014-11-14-carbon-dating-web-version-20.html>, 2014.
- [216] K. Ogura, Y. Matsumoto, Y. Yamauchi, and K. Nishimoto. Kairos chat: a novel text-based chat system that has multiple streams of time. In *Proceedings of the 28th of the International Conference on Human Factors in Computing Systems*, pages 3721–3726, 2010.
- [217] C. Olston, A. Ailamaki, C. Garrod, B. M. Maggs, A. Manjhi, and T. C. Mowry. A scalability service for dynamic web applications. In *Proceedings of the 2nd Conference on Innovative Data Systems Research*, pages 56–69, 2005.

- [218] C. Olston and S. Pandey. Recrawl scheduling based on information longevity. In *Proceedings of the 17th International Conference on World Wide Web*, pages 437–446, 2008.
- [219] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *Proceedings of the 32nd ACM SIGMOD International Conference on Management of Data*, pages 73–84, 2002.
- [220] S. Oney and B. Myers. Firecrystal: Understanding interactive behaviors in dynamic web pages. In *Proceedings of the 25th IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 105–108, 2009.
- [221] Open Ajax alliance. Introducing Ajax and OpenAjax. <http://www.openajax.org/whitepapers/Introducing%20Ajax%20and%20OpenAjax.php>, 2013.
- [222] P. L. Bogen, II, F. M. Shipman, III, and R. Furuta. Distributed Collections of Web Pages in the Wild. Technical Report arXiv:1101.0613, Center for the Study of Digital Libraries, Department of Computer Science and Engineering, Texas A&M University, 2011.
- [223] PageFreezer Software, Inc. Pagefreezer product overview: Website archiving. Technical report, PageFreezer Software, Inc., 2011.
- [224] B. Parmanto and X. Zeng. Metric for Web Accessibility Evaluation. *Journal of the American Society for Information Science and Technology*, 56(13):1394–1404, 2005.
- [225] S. Periyapatna. Total recall for Ajax applications - Firefox extension. http://scholarworks.sjsu.edu/etd_projects/139/, 2009.
- [226] PhantomJS. PhantomJS. <http://phantomjs.org/>, 2013.
- [227] M. E. Pierce, G. Fox, H. Yuan, and Y. Deng. Cyberinfrastructure and Web 2.0. *High Performance Computing and Grids in Action*, 16(3):265–287, 2008.
- [228] A. Potter. @iipc12: A week of web archiving. <http://blogs.loc.gov/digitalpreservation/2012/05/iipc12-a-week-of-web-archiving/>, 2012.
- [229] N. Potter. Wikipedia Blackout: Websites Wikipedia, Reddit, Others Go Dark Wednesday to Protest SOPA, PIPA. <http://abcnews.go.com/Technology/>

- wikipedia-blackout-websites-wikipedia-reddit-dark-wednesday-protest/story?id=15373251, January 2012.
- [230] PTgui. PTgui: Create high quality panoramic images. <http://www.ptgui.com/>, 2013.
 - [231] A. Puerta and J. Eisenstein. XIML: a common representation for interaction data. In *Proceedings of the 7th International Conference on Intelligent User Interfaces*, pages 214–215, 2002.
 - [232] P. Rademacher, J. Lengyel, E. Cutrell, and T. Whitted. Measuring the Perception of Visual Realism in Images. In *Proceedings of the 12th Eurographics Workshop*, Eurographics, pages 235–247. 2001.
 - [233] K. Radinsky and P. N. Bennett. Predicting content change on the web. In *Proceedings of the 6th International Conference on Web Search and Web Data Mining*, pages 414–424, 2013.
 - [234] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. Technical Report 2000-36, Stanford InfoLab, 2000.
 - [235] S. Reed. Introduction to Umbra. <https://webarchive.jira.com/wiki/display/ARIH/Introduction+to+Umbra>, 2014.
 - [236] S. Reed and K. Hawley. Using the Archive-It Proxy Mode Firefox Add-on. <http://ws-dl.blogspot.com/2015/02/2015-02-17-fixing-links-on-live-web.html>, 2015.
 - [237] C. Reis, J. Dunagan, H. J. Wang, O. Dubrovsky, and S. Esmeir. BrowserShield: Vulnerability-driven filtering of dynamic HTML. *ACM Transactions on the Web*, 1(3), 2007.
 - [238] B. Reyes Ayala, M. E. Phillips, and L. Ko. Current Quality Assurance Practices in Web Archiving. Technical Report <http://digital.library.unt.edu/ark:/67531/metadc333026/>, UNT Digital Library, 2014.
 - [239] S. Rhodes. “Link Rot” and Legal Resources on the Web: A 2010 Analysis by the Chesapeake Project. <http://cdm266901.cdmhost.com/custompages/linkrot2010.php>, 2011.

- [240] RickWeiss. On the web, research work proves ephemeral. <http://www.washingtonpost.com/ac2/wp-dyn/A8730-2003Nov23>, 2003.
- [241] Robots.txt. The Web Robots Page. <http://www.robotstxt.org/>, 2014.
- [242] J. J. Rodriguez, L. I. Kuncheva, and C. J. Alonso. Rotation forest: A new classifier ensemble method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1619–1630, 2006.
- [243] P. Rodriguez and E. W. Biersack. Dynamic parallel access to replicated content in the internet. *ACM/IEEE Transactions on Networking*, 10(4):455–465, 2002.
- [244] D. R. Rosenthal. “The Prostate Cancer of Preservation” Re-examined . <http://blog.dshr.org/2015/09/the-prostate-cancer-of-preservation-re.html>, 2015.
- [245] D. S. H. Rosenthal. Moonalice plays Palo Alto. <http://blog.dshr.org/2011/08/moonalice-plays-palo-alto.html>, August 2011.
- [246] D. S. H. Rosenthal. Harvesting and Preserving the Future Web. <http://blog.dshr.org/2012/05/harvesting-and-preserving-future-web.html>, May 2012.
- [247] D. S. H. Rosenthal. Talk on Harvesting the Future Web at IIPC2013. <http://blog.dshr.org/2013/04/talk-on-harvesting-future-web-at.html>, 2013.
- [248] D. S. H. Rosenthal, D. L. Vargas, T. A. Lipkis, and C. T. Griffin. Enhancing the LOCKSS Digital Preservation Technology. *D-Lib Magazine*, 21(9/10), September/October 2015.
- [249] A. Rossi. 80 Terabytes of Archived Web Crawl Data Available For Research. <http://blog.archive.org/2012/10/26/80-terabytes-of-archived-web-crawl-data-available-for-research/>, 2012.
- [250] A. Rossi. Fixing Broken Links on the Internet. <https://blog.archive.org/2013/10/25/fixing-broken-links/>, 2013.
- [251] H. SalahEldeen. Carbon dating the web. <http://ws-dl.blogspot.com/2013/04/2013-04-19-carbon-dating-web.html>, 2013.

- [252] H. M. SalahEldeen and M. L. Nelson. Losing my revolution: how many resources shared on social media have been lost? In *Proceedings of the 2nd International Conference on Theory and Practice of Digital Libraries*, pages 125–137, 2012.
- [253] H. M. SalahEldeen and M. L. Nelson. Carbon Dating the Web: Estimating the Age of Web Resources. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1075–1082, 2013.
- [254] H. M. SalahEldeen and M. L. Nelson. Resurrecting My Revolution: Using Social Link Neighborhood in Bringing Context to the Disappearing Web. In *Proceedings of the 3rd International Conference on Theory and Practice of Digital Libraries*, pages 333–345, 2013.
- [255] R. Sanderson and A. AlSum. MementoFox 0.9.52.1. <https://addons.mozilla.org/en-us/firefox/addon/mementofox/>, 2011.
- [256] R. Sanderson, H. Shankar, M. Klein, and H. Van de Sompel. Sitestory. <http://mementoweb.github.com/SiteStory/>, 2013.
- [257] C. R. Sastry, D. P. Lewis, and A. Pizano. Webtour: a system to record and playback dynamic multimedia annotations on web document content. In *Proceedings of the 7th ACM International Conference on Multimedia*, pages 175–178, 1999.
- [258] R. Schneider and F. McCown. First Steps in Archiving the Mobile Web: Automated Discovery of Mobile Websites. In *Proceedings of the 13th ACM/IEEE Joint Conference on Digital Libraries*, pages 53–56, 2013.
- [259] C. Shah. Tubekit: A query-based youtube crawling toolkit. In *Proceedings of the 8th ACM/IEEE Joint Conference on Digital Libraries*, pages 433–433, 2008.
- [260] H. Shankar. Memento Time Travel. <https://chrome.google.com/webstore/detail/memento-time-travel/jgbfpjledahoajcppakbgilmojkagghm>, 2015.
- [261] R. Shannon, A. Quigley, and P. Nixon. Time sequences. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, pages 4615–4620, 2009.

- [262] F. Shipman, C. Marshall, R. Furuta, D. Brenner, H. wei Hsieh, and V. Kumar. Creating educational guided paths over the World-Wide Web. *Ed-Telecom 96*, pages 326–331, 1996.
- [263] F. M. Shipman, III, R. Furuta, and C. C. Marshall. Generating web-based presentations in spatial hypertext. In *Proceedings of the 2nd International Conference on Intelligent User Interfaces*, pages 71–78, 1997.
- [264] K. Sigurðsson. Incremental crawling with Heritrix. In *Proceedings of the 5th International Web Archiving Workshop*, 2005.
- [265] K. Sigurðsson. The results of URI-agnostic deduplication on a domain crawl. <http://kris-sigur.blogspot.com/2014/12/the-results-of-uri-agnostic.html>, 2014.
- [266] K. Sigurðsson. URI agnostic deduplication on content discovered at crawl time. <http://kris-sigur.blogspot.com/2014/12/uri-agnostic-deduplication-on-content.html>, 2014.
- [267] R. Singh and B. D. Bhatarai. Information-theoretic identification of content pages for analyzing user information needs and actions on the multimedia web. In *Proceedings of the 2009 ACM Symposium on Applied Computing*, pages 1806–1810, 2009.
- [268] S. Sivasubramanian, G. Pierre, M. van Steen, and G. Alonso. Analysis of caching and replication strategies for web applications. *IEEE Internet Computing*, 11:60–66, 2007.
- [269] A. Sivell. The History of Mobile Computing, 2013. <http://adamsivell.blogspot.com/2013/06/the-history-of-mobile-computing.html>.
- [270] SoapTeam: Extreme Computing Lab. Remote procedure calls in SOAP, August 2000. http://www.extreme.indiana.edu/xgws/papers/sc00_paper/node5.html.
- [271] H. Song, H.-h. Chu, N. Islam, S. Kurakake, and M. Katagiri. Browser state repository service. In *Pervasive Computing*, volume 2414 of *Lecture Notes in Computer Science*, pages 647–653, 2002.

- [272] M. Song and C. Quintana. TAVR: temporal-aural-visual representation to convey imperceptible spatial information. In *Proceedings of the 28th of the International Conference on Human Factors in Computing Systems*, pages 3967–3972, 2010.
- [273] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning block importance models for web pages. In *Proceedings of the 13th International Conference on World Wide Web*, pages 203–211, 2004.
- [274] M. Spaniol, D. Denev, A. Mazeika, G. Weikum, and P. Senellart. Data quality in web archiving. In *Proceedings of the 3rd Workshop on Information Credibility on the Web*, pages 19–26, 2009.
- [275] M. Spaniol, A. Mazeika, D. Denev, and G. Weikum. Catch me if you can: Visual Analysis of Coherence Defects in Web Archiving. In *Proceedings of The 9th International Web Archiving Workshop*, pages 27–37, 2009.
- [276] J. Srivastava, R. Cooley, M. Deshpande, and P.-N. Tan. Web usage mining: discovery and applications of usage patterns from web data. *Knowledge Discovery and Data Mining Explorations Newsletter*, 1:12–23, 2000.
- [277] D. Stenberg. cURL. <http://curl.haxx.se/>, 2015.
- [278] Y. Sun, Z. Zhuang, and C. L. Giles. A large-scale study of robots.txt. In *Proceedings of the 16th International Conference on World Wide Web*, pages 1123–1124, 2007.
- [279] T. V. Raman and A. Malhotra. Identifying application state. <http://www.w3.org/2001/tag/doc/IdentifyingApplicationState>, 2011.
- [280] J. Teevan, S. T. Dumais, D. J. Liebling, and R. L. Hughes. Changing how people view changes on the web. In *Proceedings of the 22nd annual ACM Symposium on User Interface Software and Technology*, pages 237–246, 2009.
- [281] J. Tennison. Hash URIs. <http://www.jenitennison.com/blog/node/154>, 2011.
- [282] D. Thain, P. Ivie, and H. Meng. Techniques for Preserving Scientific Software Executions: Preserve the Mess or Encourage Cleanliness? In *Proceedings of*

- 12th International Conference on Preservation of Digital Objects*, pages 1–10, 2015.
- [283] The Library of Congress Web Archives. September 11th Web Archives. <http://lcweb2.loc.gov/diglib/lcwa/html/sept11/sept11-overview.html>, 2012.
- [284] The MITRE Corporation. Rebuilding the Pentagon DATA NETWORKS: Coming Through Loud and Clear. http://web.archive.org/web/20130310073245/http://mitre.org/news/digest/archives/2002/9_11pentagon.html, 2002.
- [285] K. Thibodeau. Building the Archives of the Future: Advances in Preserving Electronic Records at the National Archives and Records Administration. *D-Lib Magazine*, 7(2), February 2001.
- [286] P. Timmins, S. McCormick, E. Agu, and C. Wills. Characteristics of mobile web content. In *Proceedings of the 1st IEEE Workshop on Hot Topics in Web Systems and Technologies*, pages 1–10, 2006.
- [287] B. Tofel. ‘Wayback’ for Accessing Web Archives. In *Proceedings of the 7th International Web Archiving Workshop*, 2007.
- [288] H. Tweedy, F. McCown, and M. L. Nelson. A Memento Web Browser for iOS. In *Proceedings of the 13th ACM/IEEE Joint Conference on Digital Libraries*, pages 371–372, 2013.
- [289] H. Van de Sompel. Memento framework introduction. <http://mementoweb.org/guide/quick-intro/>, 2015.
- [290] H. Van de Sompel, C. Lagoze, M. L. Nelson, S. Warner, R. Sanderson, and P. Johnston. Adding eScience Assets to the Data Web. In *Proceedings of the 2nd Linked Data on the Web Workshop*, 2009.
- [291] H. Van de Sompel, M. L. Nelson, C. Lagoze, and S. Warner. Resource harvesting within the OAI-PMH framework. *D-Lib Magazine*, 10(12), 2004.
- [292] H. Van de Sompel, M. L. Nelson, R. Sanderson, L. L. Balakireva, S. Ainsworth, and H. Shankar. Memento: Time Travel for the Web. Technical Report arXiv:0911.1112, Los Alamos National Laboratory, 2009.

- [293] H. Van de Sompel, M. L. Nelson, and R. D. Sanderson. RFC 7089: HTTP Framework for Time-Based Access to Resource States – Memento. <http://mementoweb.org/guide/rfc/>, 2013.
- [294] H. Van de Sompel, R. Sanderson, M. L. Nelson, L. L. Balakireva, H. Shankar, and S. Ainsworth. An HTTP-Based Versioning Mechanism for Linked Data. In *Proceedings of the 3rd Linked Data on the Web Workshop*, 2010.
- [295] R. Voorburg. robustify.js. <http://digitopia.nl/robustify/example.html>, 2015.
- [296] R. Voorburg. robustify.js: Returning a Memento iso a “404”. datatopia.blogspot.nl/2015/02/robustifyjs-returning-memento-iso-404.html, 2015.
- [297] W3C. HTTP request fields. <http://www.w3.org/Protocols/HTTP/HTRQ-Headers.html>, 1994.
- [298] W3C. Dereferencing HTTP URIs, May 2007. <http://www.w3.org/2001/tag/doc/httpRange-14/2007-05-31/HttpRange-14>.
- [299] W3C. Simple Object Access Protocol (SOAP) 1.1. Technical Report W3C Recommendation (Second Edition) 27 April 2007, W3C, 2007.
- [300] W3C Interest Group. HTML5: A vocabulary and associated APIs for HTML and XHTML. <http://www.w3.org/TR/html5/browsers.html#the-history-interface>.
- [301] W3C staff and working group participants. Hash URIs. http://www.w3.org/QA/2011/05/hash_uris.html, December 2011.
- [302] B. V. Wakode, D. N. Choudhari, and V. M. Thakare. Accessing Web Content on Mobile Devices: Relevance, State of the Art and Future Direction. *International Journal of Computer Science & Applications*, 1(4):16–21, 2012.
- [303] J. Wang. A survey of web caching schemes for the Internet. *ACM SIGCOMM Computer Communication Review*, 29(5):36–46, 1999.
- [304] K. E. Warner. Tobacco industry response to public health concern: A content analysis of cigarette ads. *Health, Education, & Behavior*, 12(1):115–127, 2012.

- [305] S. J. Waterson, J. I. Hong, T. Sohn, J. A. Landay, J. Heer, and T. Matthews. What did they do? Understanding clickstreams with the WebQuilt visualization system. In *Proceedings of the 10th Working Conference on Advanced Visual Interfaces*, pages 94–102, 2002.
- [306] E. Weiner. Will Future Historians Consider These Days The Digital Dark Ages? <http://www.npr.org/2016/01/04/461878724/will-future-historians-consider-these-times-the-digital-dark-ages>, 2016.
- [307] R. Weiss. On the Web, Research Work Proves Ephemeral . http://web.archive.org/web/20140317093104/http://stevereads.com/cache/ephemeral_web_pages.html, 2003.
- [308] D. Wessels. *Squid: the Definitive Guide*. O'Reilly Media, Incorporated, 2004.
- [309] Wikipedia. Ajax (programming). [http://en.wikipedia.org/wiki/Ajax_\(programming\)](http://en.wikipedia.org/wiki/Ajax_(programming)), 2013.
- [310] Wikipedia.com. Stop Online Piracy Act. http://en.wikipedia.org/wiki/Stop_Online_Piracy_Act, January 2012.
- [311] Wikipedia.org. David Hockney. http://en.wikipedia.org/wiki/David_Hockney, 2013.
- [312] L. Yi, B. Liu, and X. Li. Eliminating noisy information in web pages for data mining. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 296–305, 2003.
- [313] A. Yildiz, B. Aktemur, and H. Sozer. Rumadai: A plug-in to record and replay client-side events of web sites with dynamic content. In *Proceedings of the 2nd Workshop on Developing Tools as Plug-ins*, pages 88 –89, 2012.
- [314] YouTube. YouTube. <http://www.youtube.com/>, 2013.
- [315] D. Yu, A. Chander, N. Islam, and I. Serikov. JavaScript instrumentation for browser security. In *Proceedings of the 34th annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 237–249, 2007.

- [316] X. Zhang, W. Lin, and P. Xue. Just-noticeable difference estimation with pixels in images. *Journal of Visual Communication and Image Representation*, 19(1):30–41, 2008.
- [317] Y. Zheng, T. Bao, and X. Zhang. Statically Locating Web Application Bugs Caused by Asynchronous Calls. In *Proceedings of the 20th International Conference on World Wide Web*, pages 805–814, 2011.
- [318] H. Zhu and T. Yang. Class-based cache management for dynamic web content. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1215–1224, 2000.
- [319] D. F. Zucker. What does Ajax mean for you? *Interactions*, 14:10–12, 2007.