Fall 12-2020

MementoMap: A Web Archive Profiling Framework for Efficient Memento Routing

Sawood Alam
Old Dominion University, ibnesayeed@gmail.com

MEMENTOMAP: A WEB ARCHIVE PROFILING FRAMEWORK FOR EFFICIENT MEMENTO ROUTING

by

Sawood Alam
B.Tech. May 2008, Jamia Millia Islamia, India
M.S. August 2013, 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
December 2020

Approved by:

Michael L. Nelson (Director)

Michele C. Weigle (Member)

Jian Wu (Member)

Sampath Jayarathna (Member)

Erika F. Frydenlund (Member)

ABSTRACT

MEMENTOMAP: A WEB ARCHIVE PROFILING FRAMEWORK FOR EFFICIENT MEMENTO ROUTING

Sawood Alam
Old Dominion University, 2020
Director: Dr. Michael L. Nelson

With the proliferation of public web archives, it is becoming more important to better profile their contents, both to understand their immense holdings as well as to support routing of requests in Memento aggregators. A memento is a past version of a web page and a Memento aggregator is a tool or service that aggregates mementos from many different web archives. To save resources, the Memento aggregator should only poll the archives that are likely to have a copy of the requested Uniform Resource Identifier (URI). Using the Crawler Index (CDX), we generate profiles of the archives that summarize their holdings and use them to inform routing of the Memento aggregator's URI requests. Additionally, we use fulltext search (when available) or sample URI lookups to build an understanding of an archive's holdings. Previous work in profiling ranged from using full URIs (no false positives, but with large profiles) to using only top-level domains (TLDs) (smaller profiles, but with many false positives). This work explores strategies in between these two extremes.

For evaluation we used CDX files from Archive-It, UK Web Archive, Stanford Web Archive Portal, and Arquivo.pt. Moreover, we used web server access log files from the Internet Archive's Wayback Machine, UK Web Archive, Arquivo.pt, LANL's Memento Proxy, and ODU's MemGator Server. In addition, we utilized historical dataset of URIs from DMOZ.

In early experiments with various URI-based static profiling policies we successfully identified about 78% of the URIs that were not present in the archive with less than 1% relative cost as compared to the complete knowledge profile and 94% URIs with less than 10% relative cost without any false negatives. In another experiment we found that we can correctly route 80% of the requests while maintaining about 0.9 recall by discovering only 10% of the archive holdings and generating a profile that costs less than 1% of the complete knowledge profile.

We created *MementoMap*, a framework that allows web archives and third parties to express holdings and/or voids of an archive of any size with varying levels of details to fulfil various application needs. Our archive profiling framework enables tools and services to

predict and rank archives where mementos of a requested URI are likely to be present.

In static profiling policies we predefined the maximum depth of host and path segments of URIs for each policy that are used as URI keys. This gave us a good baseline for evaluation, but was not suitable for merging profiles with different policies. Later, we introduced a more flexible means to represent URI keys that uses wildcard characters to indicate whether a URI key was truncated. Moreover, we developed an algorithm to rollup URI keys dynamically at arbitrary depths when sufficient archiving activity is detected under certain URI prefixes. In an experiment with dynamic profiling of archival holdings we found that a *MementoMap* of less than 1.5% relative cost can correctly identify the presence or absence of 60% of the lookup URIs in the corresponding archive without any false negatives (i.e., 100% recall). In addition, we separately evaluated archival voids based on the most frequently accessed resources in the access log and found that we could have avoided more than 8% of the false positives without introducing any false negatives.

We defined a *routing* score that can be used for Memento routing. Using a cut-off threshold technique on our *routing* score we achieved over 96% accuracy if we accept about 89% recall and for a recall of 99% we managed to get about 68% accuracy, which translates to about 72% saving in wasted lookup requests in our Memento aggregator. Moreover, when using top-k archives based on our *routing* score for routing and choosing only the topmost archive, we missed only about 8% of the sample URIs that are present in at least one archive, but when we selected top-2 archives, we missed less than 2% of these URIs. We also evaluated a machine learning-based routing approach, which resulted in an overall better accuracy, but poorer recall due to low prevalence of the sample lookup URI dataset in different web archives.

We contributed various algorithms, such as a space and time efficient approach to ingest large lists of URIs to generate *MementoMaps* and a Random Searcher Model to discover samples of holdings of web archives. We contributed numerous tools to support various aspects of web archiving and replay, such as MemGator (a Memento aggregator), InterPlanetary Wayback (a novel archival replay system), Reconstructive (a client-side request rerouting ServiceWorker), and AccessLog Parser. Moreover, this work yielded a file format specification draft called Unified Key Value Store (UKVS) that we use for serialization and dissemination of *MementoMaps*. It is a flexible and extensible file format that allows easy interactions with Unix text processing tools. UKVS can be used in many applications beyond *MementoMaps*.

Beneath my mother's feet...

ACKNOWLEDGEMENTS

All praises be to the Almighty to bring me this far. I am deeply grateful for the help, support, and contributions from numerous people and organizations in this journey.

My advisor during my masters and doctorate degree programs, Dr. Michael L. Nelson, has been a tremendous support to me in my academic journey and career. If I were to lead a team in the future, punctual weekly progress meetings and rigorous practice sessions before every upcoming presentation are the two lessons I have learned from him that I would replicate. My co-advisor Dr. Michele C. Weigle helped me learn how to express research findings more effectively using suitable information visualization techniques for various types of results and telling stories with properly annotated figures. These two people get the lion share of credits for turning me into a researcher and teaching me techniques of effective scholarly communication.

My doctoral dissertation committee members Dr. Jian Wu and Dr. Sampath Jayarathna were helpful beyond this research. They were available for discussions on many research topics and were supportive in finding job opportunities for me. Dr. Erika F. Frydenlund provided an outsider's perspective and her reflections improved the readability of my dissertation for those unfamiliar with my research discipline.

Prior works of Dr. Robert Sanderson and Dr. Ahmed AlSum formed the basis of this research. I got the opportunity to work with Dr. David S. H. Rosenthal, Dr. Herbert Van de Sompel, Dr. Martin Klein, Dr. Lyudmila L. Balakireva, and Harihar Shankar during the IIPC funded preliminary exploration phase of this archive profiling work. Dr. Rosenthal's blog post reviews after his retirement from Stanford kept this work in check and reassured that I was on the right track. Dr. Sanderson reflected his thoughts and provided constructive feedback on our *MementoMap* framework during JCDL '19. Dr. Edward A. Fox from Virginia Tech reflected his thoughts on our *URI Key* concept and showed interest on potential future collaborations.

Numerous IIPC member organizations and individuals contributed to this work with datasets and feedback. Kris Carpenter and Jefferson Bailey from the Internet Archive helped us with the Archive-It WARC dataset. Joseph E. Ruettgers helped us with the petabytes of storage space setup at ODU. Daniel Gomes and Fernando Melo were very generous in sharing complete dataset of CDX files and access logs from Arquivo.pt web archive. Nicholas Taylor shared Stanford Web Archive Portal's CDX datasets. Dr. Andrew Jackson from the British Library shared CDX files and a sample of access logs from the UK Web Archive

and provided feedback on the IIPC funded archive profiling project. Kristinn Sigurðsson provided feedback during the early days of the project. Garth Stewart shared CDX files from the National Records of Scotland. Ilya Kreymer contributed to the discussion about CDXJ profile serialization format and shared OldWeb.today access logs. Alex Osborne and Dr. Paul Koerbin worked on sharing the web archive index from the National Library of Australia. Olga Holownia from the British Library helped the progress of IIPC funded project in numerous ways. Dr. Ian Milligan gave me the opportunity to attend multiple Archives Unleashed datathons, an event where *InterPlanetary Wayback* was created.

Many people provided early feedback on this work during the Doctoral Consortium events of JCDL and TPDL. Dr. Nattiya Kanhabua offered some really good suggestions and potential related work references.

Dr. M. M. Sufyan Beg, Dr. Mohammad Zubair, and Dr. Hussein Abdel-Wahab were the key people who helped me join the Old Dominion University. Dr. Ravi Mukkamala has always been helpful in administrative matters and beyond. Dr. Stephan Olariu has been available to discuss random interesting research topics every time I approached him. Dr. Steven Zeil helped me with the logistics of teaching when I offered the *Web Server Design* course. Ajay Gupta and many members of the Systems Group ensured that necessary computing resources were available and were in good shape. Ariel Sturtevant and Phyllis Woods took care of all the paper work.

Mark Graham and Brewster Kahle supported me by keeping the workload on me to the minimum while I was finalizing my dissertation and working for the Internet Archive. National Digital Stewardship Alliance recognized my work on the digital preservation, including this work, and honored me with the Future Steward Innovation Award.

The list of past and ongoing collaborations with Dr. Mat Kelly are too long to list here, but the routine of taking a break and going for a campus walk while exchanging ideas was perhaps one of the most valuable memories we shared together. Dr. Alexander Nwala was my go-to person for discussing matters related to language modeling, machine learning, and classification. Dr. Mohamed Aturban and I shared the lab till late night and we collaborated on archival fixity. Dr. Lulwah M. Alkwai and family are the ones I can count on even after they have gone back to their home country. Dr. Justin F. Brunelle and family were my tailgate hosts during Football season. Shawn M. Jones always had crisp and unambiguous ideas and very receptive of feedback, someone I am looking forward to collaborate with on formal publications in the future. John A. Berlin contributed to the *Reconstructive* project. Plinio Vargas and I used to begin our day by solving puzzles and riddles and I enjoyed

collaborating with him on a web archiving issue related to Twitter. Nauman Siddique and Hussam Hallak were my friends I could call for help in person anytime. Himarsha Jayanetti and Kritika Garg were my students and mentees I enjoyed working with and they contributed to this work in the analysis of Arquivo.pt access logs. I have a long list of good memories with every Web Science and Digital Libraries Research Group member I shared the academic timeline with.

The Urdu community on UrduWeb is my home on the internet. I need to get back to it and put fresh energy into numerous projects that are on hold and need to be mobilized.

My mother Sitara Begum, elder brother Masood Alam, maternal uncle Dr. Shabbir Ahmad Khan, and maternal aunt Naseema Khatoon are the four most significant people in my life due to their immense support and sacrifices towards my upbringing and growth during the most unfavorable times. I am determined to continue working towards fulfilling their dreams. I missed seeing my father Sayeed Ahmad all these years of my graduation life. My wife Rehana Khatoon and daughter Fareeha Khan were always there for me and they took care of all my needs silently. Abida, Aqeela, Wadood, Faizan, Raihan, Sumbul, Nabeel, Raed, Manal, Shaima, and the rest of the extended family members, I love you all and I miss you. My thoughts are with my grandparents I will never get to see them again, may you all rest in peace.

This work was supported in part by the International Internet Preservation Consortium, the National Science Foundation, and the Andrew W. Mellon Foundation.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Web archives capture web pages from the live web, index them, and make them available for replay later. The Memento protocol [245] provides a uniform means to lookup archived resources in various web archives in the form of a list of all the captures, or mementos, of a web page or a specific memento closest to a given date and time in the past. The number of public web archives of varying sizes and collection policies supporting this protocol natively or through proxies continues to grow [118, 141, 204, 240].

Figure 1 shows two copies of the early versions of the Smithsonian Institution's home page in two different web archives. Past versions of web pages are called mementos. A memento of a resource can be served from the original server itself (e.g., a revision of a wiki page) or from a web archive (even after the original server is gone).

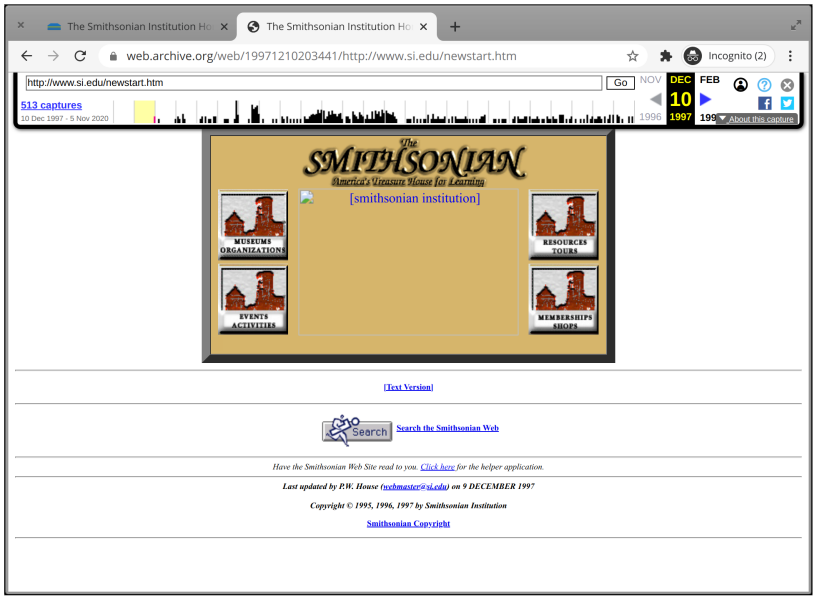
Discoverability of archived resources is in the interest of both users and web archives. Users want to find mementos that accurately represent the state of the resource at a given time in the past which is more likely to be the case if there are many mementos of the same resource (potentially in many different archives) over the period of the life of the web page. Web archives want their collections to be utilized whenever they have a memento that a user might be interested in. Memento aggregators are services that perform lookup for mementos of web pages across many different web archives using the Memento protocol and provide consolidated results. Without an aggregator, small archives will never get the exposure that the Internet Archive (IA)[1] gets (which has an Alexa ranking close to 200 for the last several years[2]).

In this work we establish a framework for *archive profiles* to systematically describe holdings of various web archives whether they are generated by the archives themselves or third parties. Depending on the application, available resources, and the level of access to the archives, one may generate a brief summary of holdings, a detailed ledger, or anything in between. Among many use cases of archive profiles the primary purpose is efficient Memento aggregation. There is a tradeoff between increasing exposure to web archives (hence mementos) and managing load (hence resources and response time). The goal is to

[1] `https://web.archive.org/`

[2] `https://www.alexa.com/siteinfo/archive.org`

(a) First Memento of the Smithsonian Institution in Arquivo.pt from October 1996. `https://arquivo.pt/wayback/19961013204418/http://www.si.edu/`



(b) First Memento of the Smithsonian Institution in the Internet Archive from December 1997 (JavaScript in the home page changes the location to "`newstart.htm`"). `https://web.archive.org/web/19971210203441/http://www.si.edu/newstart.htm`

Fig. 1. Early Mementos of the Smithsonian Institution Home Page

identify a few potential archives among all archives known to the aggregator that are likely to return good results for a given page lookup.

1.1 MOTIVATION

A brief discussion on the following questions can help understanding the purpose and importance of this work:

- Why aggregate web archives anyway?
- Why aggregate small web archives?
- Why profile web archives?
- Why route lookup requests?

1.1.1 WHY AGGREGATE WEB ARCHIVES ANYWAY?

The Internet Archive, the first web archive founded in 1996, has over a trillion mementos of which about half are web pages (IA defines a "web page" as an HTML, text, or PDF file [114]) as illustrated in Figure 2. Currently, it is adding roughly 1.6 billion mementos each week as shown in Figure 3. According to an engineer at IA, every URI (Uniform Resource Identifier) is captured on average about 2.1 times [115]. With that estimate, IA has about 476 billion unique original URIs and 223 billion unique original web page URIs as of October, 2020.

In November 2016, Google's *How Search Works* page[3] estimated the size of the web to be over 130 trillion individual pages [219]. On one hand, this number does not reflect the size of Google's search index which is smaller (a few hundred billion[4] or at least above 50 billion [92, 247]). On the other hand, this also does not include the number of historical pages that are long gone, which, when included, would make the size of the web even larger.

IA's seemingly large numbers, when put in perspective of the size of the web, become small, relatively. This shows that while IA is the oldest and largest web archive, it still holds only a small sample of the historical web. A recent study estimates that about two-thirds of the entire web traffic is not archivable by public archives [126], where personal/private web archiving can play an important role. Aggregating results from many archives helps discovering more archived resources.

[3] `https://www.google.com/search/howsearchworks/`

[4] `https://www.google.com/search/howsearchworks/crawling-indexing/`

https://twitter.com/internetarchive/status/1320801643135287296

Fig. 2. Over a Trillion Mementos and 468 Billion Web Pages in IA as of October 26, 2020

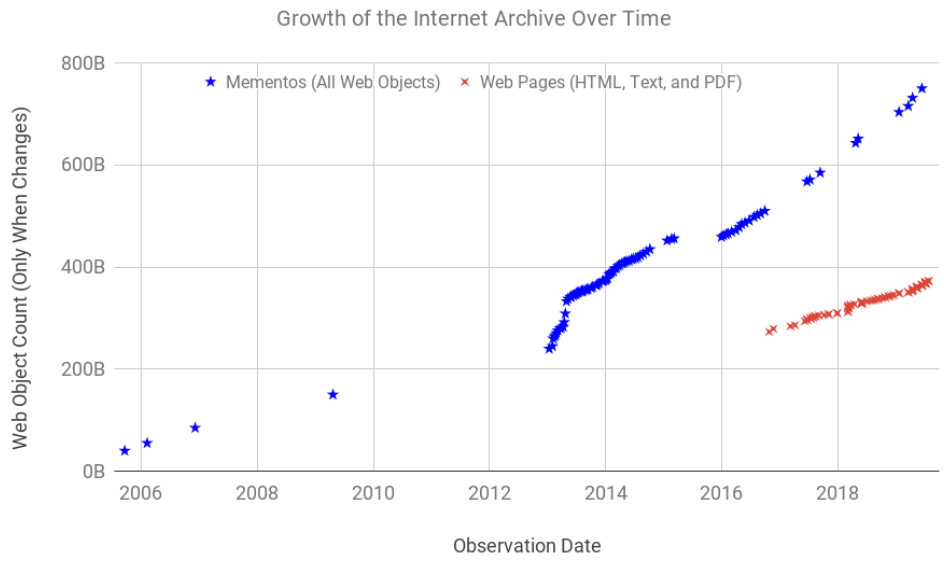


Fig. 3. Growth of Mementos and Web Pages in IA Over Time [16] (IA started reporting web page count instead of memento count on its homepage since October 2016).

IA has been blocked (or occasionally gets blocked) for years or months in many countries, including China [123], Russia [99], Jordan [4, 75], and India [229, 151, 74]. Comcast, the largest home Internet service provider in the United States, once blocked IA [73]. Being the largest web archive, IA is susceptible to many threats, including natural disasters, hacking, and Internet censorship. IA itself recognizes these threats and is trying to create mirrors in different web archives for diversity in jurisdictions, countries, and geography, including the most recent initiative of the Internet Archive of Canada [87]. Aggregation can help make the long tail of smaller web archives available to everyone even when the largest archive is inaccessible.

On June 15, 2016, I publicly demonstrated MemGator[5], our newly created Memento aggregator tool, at the Library of Congress (LC) for the first time. In the live demo I illustrated many useful functionalities using MemGator as a command-line tool. One of those examples was to count the number of mementos of a given URI for every year across various archives (as illustrated in Figure 4). Later we found that IA was down during the demonstration of MemGator due to a denial-of-service (DDoS) attack [148]. However, by the virtue of aggregating from about a dozen other web archives the demonstration worked

[5] `https://github.com/oduwsdl/MemGator`

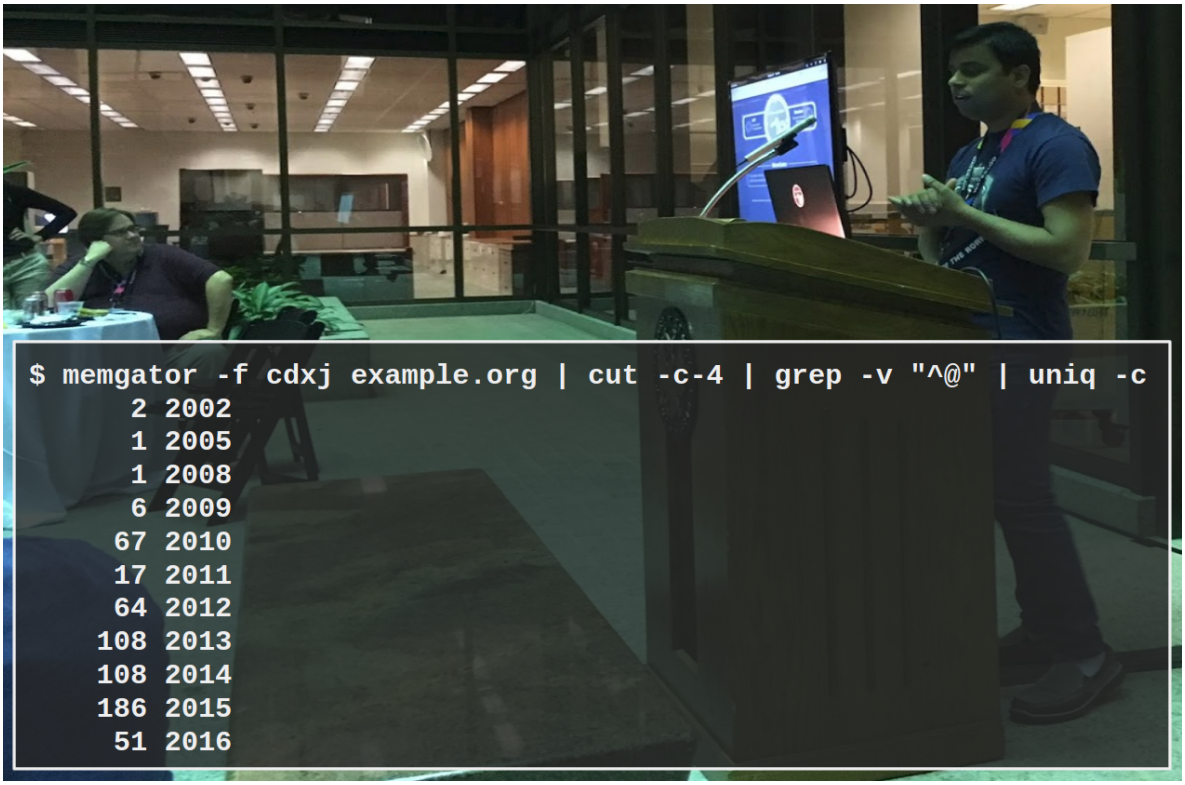


Fig. 4. MemGator Demonstration at the Library of Congress on June 15, 2016, while IA was under a DDoS attack.

without anyone noticing any failures, proving the tool to be more resilient to transient errors at individual archives. This reflects the potential experience of people living in places where certain web archives might not be accessible, if they choose to utilize Memento aggregators.

The Smithsonian Institution unveiled its first homepage (`http://www.si.edu/`) in May 1995. Over time it has gone through many changes, but they did not preserve the history of changes. Later, in 2014 they attempted to find the earliest copies of their homepage in web archives [111]. Among thousands of mementos of their homepage IA had, the earliest copy they found there was from May 1997[6], which redirects to another memento from December 1997 as shown in Figure 1b. Then they utilized Time Travel[7], a Memento aggregator service, which led them to discover a functional copy of their homepage from October 1996 in Arquivo.pt (Portuguese Web Archive) that was last updated in March 1996 as shown in Figure 1a.

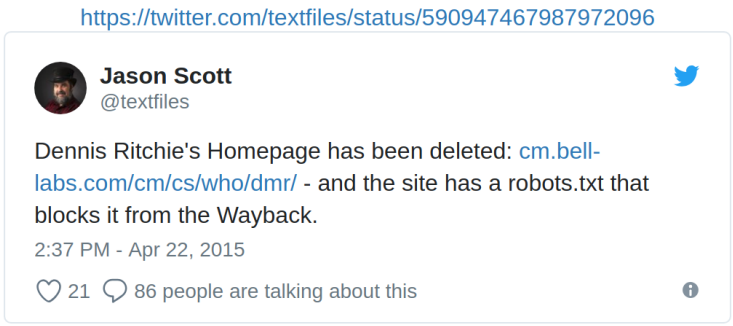
[6] `https://web.archive.org/web/19970502110751/http://si.edu`)
[7] `http://timetravel.mementoweb.org/`

Using `robots.txt` [165], IA allows webmasters to exclude historical copies of their whole site, specific sections, or specific pages from being replayed. The US government and military domains are exempted from this `robots.txt` exclusion as of December 2016 [212, 119]. In April 2015, Bell Labs[8] restructured their web pages. As a result, the home page of Dennis Ritchie, an American computer scientist primarily known as the creator of the C programming language and one of the developers of the Unix operating system, was moved from `http://cm.bell-labs.com/cm/cs/who/dmr/` to `https://www.bell-labs.com/usr/dmr/www/index.html`. People with broken links could not find any copies in the IA (as illustrated in Figure 5a). At that time, the `robots.txt` file of the `cm.bell-labs.com` in effect prevented IA from replaying any pages from that domain[9] (see Figure 81 of Appendix A for the contents of the `robots.txt` file). Later the `robots.txt` file was removed, then we found that IA has over 200 copies of Dennis Ritchie's original home page (as shown in Figure 5c). However, those using aggregators could easily find copies in other archives (as shown in Figure 5b) that do not utilize `robots.txt` for barring replay, such as Bibliotheca Alexandrina Web Archive[10], Archive.is[11], and Archive-It[12].

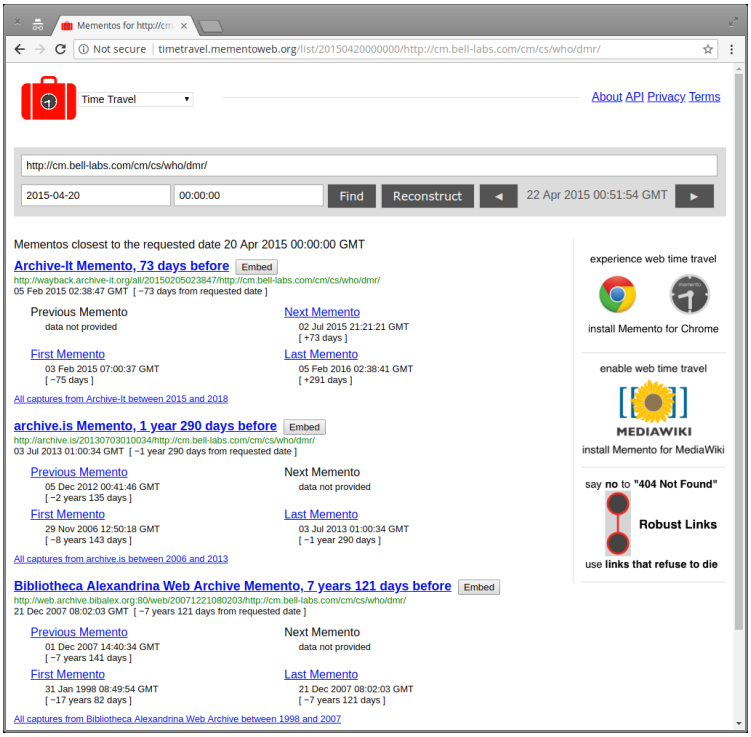
In December 2017, an American journalist Joy-Ann Reid apologized for "insensitive LGBT blog posts" she wrote more than a decade ago [232]. A few months later she claimed that either her blog or the copies of her blog in IA been hacked to fabricate homophobic posts [96]. IA denied the claim of being hacked [72], but there were no technical means deployed to prove fixity and non-repudiation of archived content [31, 47, 86]. We investigated the matter further and looked for copies of her relevant posts in other web archives [187, 186]. We found a few copies of her posts in other web archives, suggesting that the claim of IA being hacked not stand. Aggregating results from multiple independent archives gives confidence about the validity and fixity of the content or lack thereof.

Each web archive has different goals and collection policies that result in a diverse set of mementos across different archives. Some archives strive for broader coverage and run a crawler that tries to traverse the connected graph of the World Wide Web (WWW) while others limit their crawling to a specific set of web pages, such as domain names related to a specific country (for example, the Icelandic Web Archive collects `*.is` pages and a hand-picked selection of other Icelandic websites under some other TLDs[13]). Some individuals

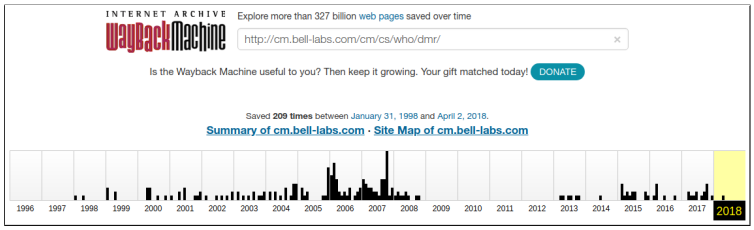
[8] `https://en.wikipedia.org/wiki/Bell_Labs`
[9] `https://web.archive.org/web/20150421154435/http://cm.bell-labs.com/robots.txt`
[10] `http://archive.bibalex.org/`
[11] `http://archive.is/`
[12] `https://archive-it.org/`
[13] `https://vefsafn.is/index.php?page=english`



(a) The Live Page is Missing and Inaccessible in IA



(b) Archives Other Than IA Have Some Copies



(c) Presence in IA Discovered Later After `robots.txt` is Removed

Fig. 5. Dennis Ritchie's Homepage at Bell Labs After Site Restructuring

and organizations build focused collections with curated list of important pages while others try to capture pages related to significant events in a timely manner. Some archives focus on high-fidelity archiving and some provide on-demand archiving of a single page at a time. Many different archives located in different parts of the world also capture the geo-location diversity on sites that serve different content in different places. For example, some of the mementos of the travel booking site Expedia in Arquivo.pt are in Spanish[14,15] (as shown in Figure 6a) while the Internet Archive has mementos of the same site in English[16] (as shown in Figure 6b) around the same time. Aggregation allows discovery of all these variations from a single convenient place.

1.1.2 WHY AGGREGATE SMALL WEB ARCHIVES?

We conducted a preliminary experiment to assess the overlap among different web archives [8, 210]. For that we collected four random samples of 1,000,000 URIs each from four different sources:

1. the historical collection of DMOZ[17]
2. Internet Archive Wayback Machine access logs
3. Memento Proxy access logs, and
4. UK Web Archive Wayback access logs

Now defunct, DMOZ was a human curated directory of web pages in nested categories. We then checked the presence of these four million sample URIs in two large web archives (Archive-It and UK Web Archive) and one small web archive (Stanford Web Archive Portal[18]).

Figure 7 shows the outcome of this experiment. There are two important observations that highlight the need for aggregation: 1) none of the three archives have more than 5% of any sample set archived, and 2) the overlap among archives is insignificant. We observe that the coverage is almost additive in nature as more collections are aggregated together. This means that an unpopular resource that is found in one web archive is less likely to be

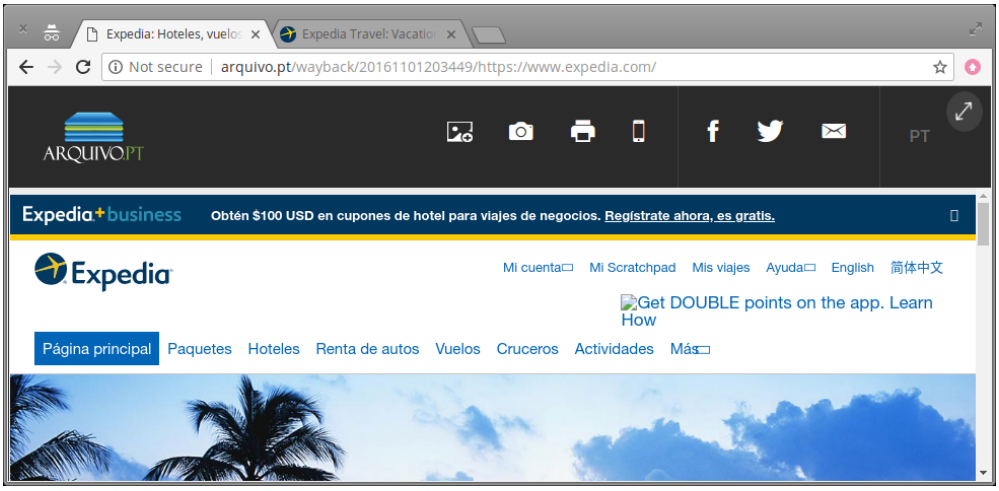
[14]Expedia did not support Portuguese at that time, hence Spanish was the closest language option for that geographical region.

[15]`http://arquivo.pt/wayback/20161101203449/https://www.expedia.com/`

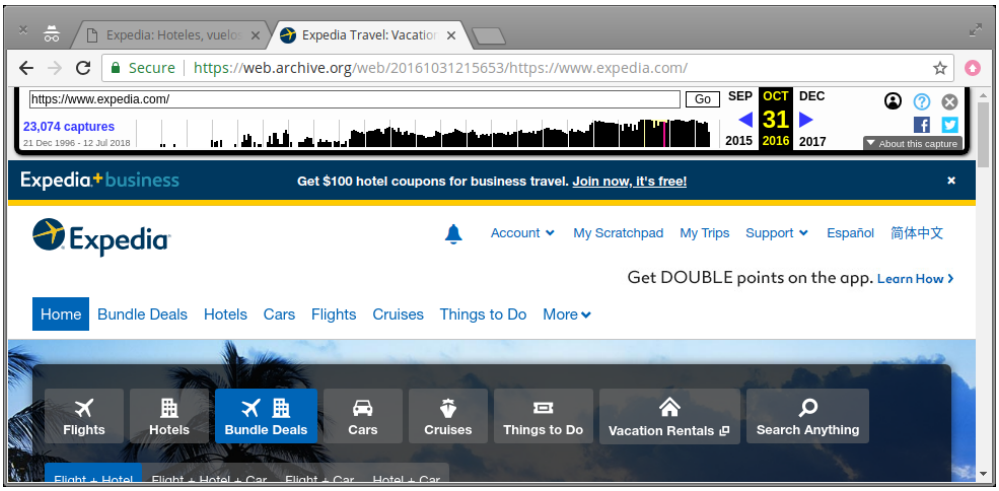
[16]`https://web.archive.org/web/20161031215653/https://www.expedia.com/`

[17]`https://en.wikipedia.org/wiki/DMOZ`

[18]`https://swap.stanford.edu/`



(a) Arquivo.pt Observed the Page in Spanish



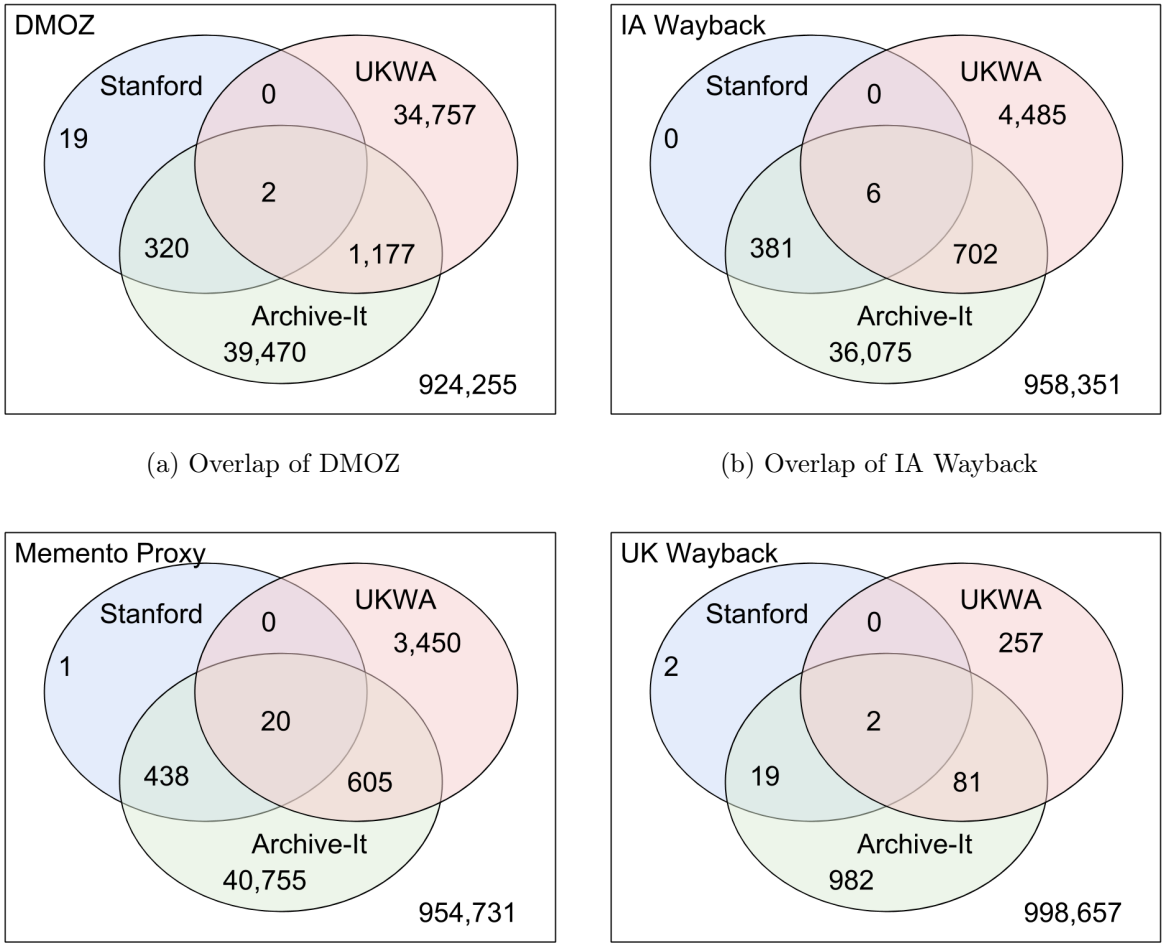
(b) Internet Archive Observed the Page in English

Fig. 6. Expedia Serves Pages in Different Languages to Different Geo-locations

found in any other archive (excluding the Internet Archive). Hence, even small archives may contribute significantly to the effectiveness of the aggregation as they are not mere subsets of large web archives.

Small curated collections often focus on high-fidelity captures that result in a more accurate replay of archived web pages than large-scale automated crawls. Rhizome's webenact[19] is a good example of such collections. Rhizome is a born-digital art institution. Their webenact service has a collection of a few hand-picked sites that have rich and interactive

[19] `http://webenact.rhizome.org/`



(a) Overlap of DMOZ

(b) Overlap of IA Wayback

(c) Overlap of Memento Proxy

(d) Overlap of UK Wayback

Fig. 7. Sample Lookup URI Overlap in Archive-It, UKWA, and Stanford Web Archives

embedded media in them. These collections include properly preserved interactions like infinite scroll, media playback, and slideshows in modal windows that are otherwise difficult to archive using traditional crawlers. Aggregating such small archives can help surface more high-fidelity mementos for users when available.

1.1.3 WHY PROFILE WEB ARCHIVES?

As web archives grow larger over time, even the curated collections may collect many unintended resources while missing out on many resources that should have been captured. This may happen due to many reasons, including vaguely configured crawling policies, scripts

Fig. 8. Unclear Holdings of Archival Collections

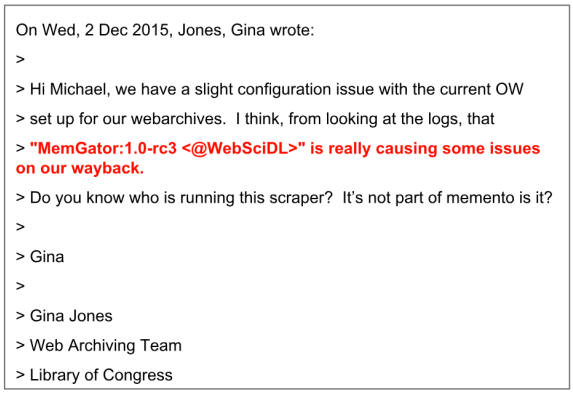
in web pages, and transient failures of web servers or crawlers. According to Andrew Jackson, the Web Archiving Technical Lead at the British Library, "We don't even know what we've got, especially for messy collections like web archives." (Figure 8) [144]. Archive profiles are one way to understand the holdings of an archive at any point in time.

Being able to describe the holdings of web archives by building archive profiles can enable many use cases. One such use case is the ability to efficiently route requests from a Memento aggregator to only archives that might have good results for the lookup URI.

Many archives would like to advertise their holdings even if those mementos are not yet made accessible publicly. The reasons for not being able to access a resource could be an embargo period, legal takedowns, requested exclusions, authorization requirements, geo-spatial restrictions, etc. These archive profiles can even enable web archives to publicly express their holdings about mementos that are only accessible to selected people, at selected places, or at specific time periods.

1.1.4 WHY ROUTE LOOKUP REQUESTS?

In December 2015, soon after the popularity surge of OldWeb.today [167], a service that allows accessing historical web in historical web browsers, many archives struggled with the increased traffic. Library of Congress contacted us about MemGator causing issues on their web archive server (as shown in Figure 9a). They found us because we added our contact information in MemGator's default user-agent request HTTP header. This field is supposed to be customized by users who run MemGator on their own. We were aware that the newly launched OldWeb.today service used MemGator which might not have customized


On Wed, 2 Dec 2015, Jones, Gina wrote:

>

> Hi Michael, we have a slight configuration issue with the current OW

> set up for our webarchives. I think, from looking at the logs, that

> **"MemGator:1.0-rc3 <@WebSciDL>" is really causing some issues on our wayback.**

> Do you know who is running this scraper? It's not part of memento is it?

>

> Gina

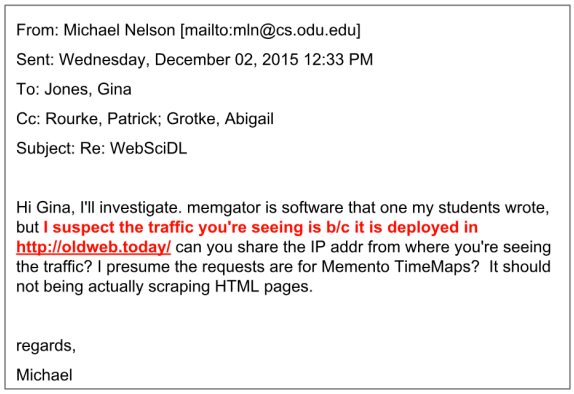
>

> Gina Jones

> Web Archiving Team

> Library of Congress

(a) LC Contacted Us About MemGator Causing Issues


From: Michael Nelson [mailto:mln@cs.odu.edu]

Sent: Wednesday, December 02, 2015 12:33 PM

To: Jones, Gina

Cc: Rourke, Patrick; Grotke, Abigail

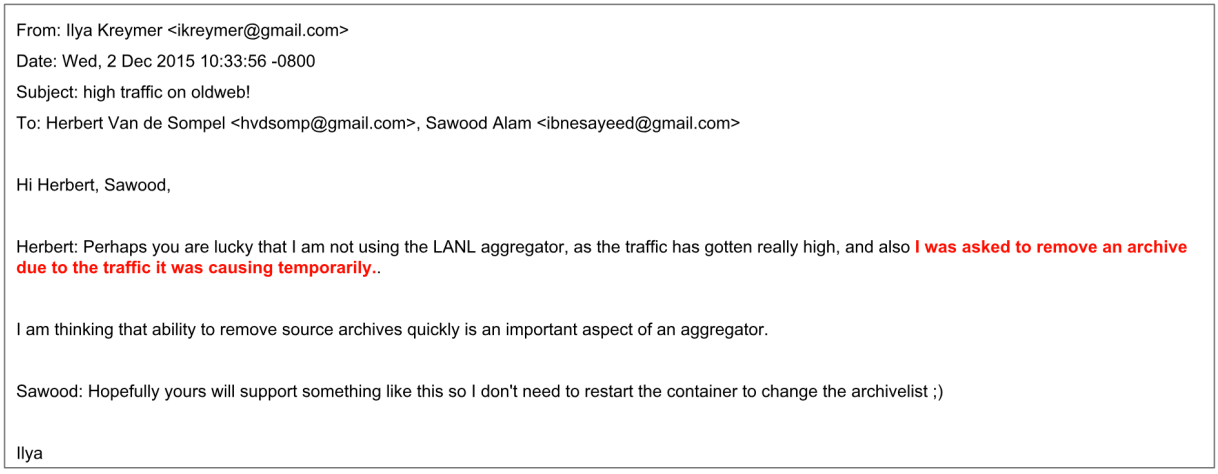
Subject: Re: WebSciDL

Hi Gina, I'll investigate. memgator is software that one my students wrote, but **I suspect the traffic you're seeing is b/c it is deployed in http://oldweb.today/** can you share the IP addr from where you're seeing the traffic? I presume the requests are for Memento TimeMaps? It should not being actually scraping HTML pages.

regards,

Michael

(b) We Responded With Potential Source of Traffic Surge


From: Ilya Kreymer <ikreymer@gmail.com>

Date: Wed, 2 Dec 2015 10:33:56 -0800

Subject: high traffic on oldweb!

To: Herbert Van de Sompel <hvdsomp@gmail.com>, Sawood Alam <ibnesayeed@gmail.com>

Hi Herbert, Sawood,

Herbert: Perhaps you are lucky that I am not using the LANL aggregator, as the traffic has gotten really high, and also **I was asked to remove an archive due to the traffic it was causing temporarily**..

I am thinking that ability to remove source archives quickly is an important aspect of an aggregator.

Sawood: Hopefully yours will support something like this so I don't need to restart the container to change the archivelist ;)

Ilya

(c) OldWeb.today Informed Us About an Exclusion Request

Fig. 9. OldWeb.today Caused Excessive Load at loc.gov

the user-agent. In our response we told LC about the potential source of traffic (as shown in Figure 9b). A while later we received an email from the administrator of OldWeb.today about an exclusion request from an archive from being aggregated (as shown in Figure 9c).

We found that fewer than 5% of the queried URIs are present in any individual archive other than IA. In this case, being able to identify a subset of archives that might return good results for a particular lookup request becomes very important.

Figures 11a and 11b illustrate a naive implementation of a Memento aggregator where each request is broadcasted to all the known archives, but only a few archives return good results. Memento aggregators, such as the Time Travel Service or MemGator, and other

```
$ curl -sL https://git.io//archives \
>      | jq '.[] | select(.id=="pastpages")'
{
  "id": "pastpages",
  "name": "PastPages Web Archive",
  "timemap": "http://www.pastpages.org/timemap/link/",
  "timegate": "http://www.pastpages.org/timegate/",
  "probability": 0.001,
  "ignore": true
}
```

Fig. 10. MemGator's Default Archives' List Explicitly Disabled PastPages

services, need to know which archives to poll when a request for an archived version of a file is received. LANL's Time Travel Service routes requests to only a subset of archives using binary classifiers trained on the observed results from various web archives in the past. The default list of archives in MemGator had explicitly disabled the PastPages Web Archive (as shown in Figure 10) because it was often not returning any good results. As a result PastPages, before it shut down in 2018 (and the content was moved to IA[20]), was never requested even if it had mementos for some lookup URIs. This binary approach of either request a given archive for every lookup request or not request for any at all is not ideal. Efficient Memento routing in aggregators is desired from both aggregators' and archives' perspective. Aggregators can reduce the average response time, improve overall throughput, and save network bandwidth. Archives benefit by the reduced number of requests for which they have no holdings, hence saving computing resources and bandwidth.

Broadcasting becomes practically impossible for an aggregator as the number of aggregation sources grow large. Archive-It (AIT), a hosted web archiving subscription service run by IA, is currently hosting over 15 thousand collections created by about two thousand organizations and individuals[21]. Aggregating mementos from these many collections can be overwhelming and impractical if the aggregator is to return the response to the client in a reasonable amount of time. Luckily, AIT provides a special memento endpoint (called "all") where results from all of the public collections on AIT are combined. However, if an

[20] `https://archive.org/details/pastpages`

[21] As of November 27, 2020, AIT had 1,860 organizations and 15,515 collections and adding hundreds of collections each month.

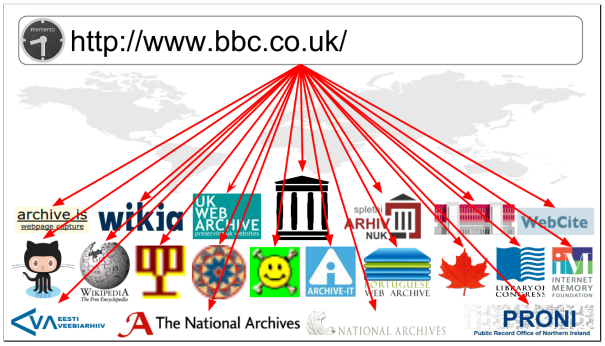
aggregator wants to exclude some collections, it has to then individually aggregate from the rest. There are some other subscription services that have gained popularity such as Webrecorder (now called Conifer)[22], archiveweb.eu[23], and the now defunct Internet Memory Foundation (IMF)[24]. They do not provide a combined endpoint yet, so the aggregator must include their collections individually. Personal web archiving (collected by individuals, potentially from their web browsing sessions) is another interesting dimension in web archiving. It is currently not very popular due to lack of tools and awareness, but a significant amount of research and development is ongoing for personal web archiving [61, 156, 158]. If personal web archiving practices proliferate, a significant number of small, but valuable archival collections might come to life that will be worth aggregating (with necessary access control).

Figure 11c illustrates a setup where a Memento aggregator first queries an archive profile service for the lookup URI. It receives a list of archives sorted in the order of the likelihood of finding archived copies of the queried file in them. The aggregator then chooses the top-k archives from the list that are above certain threshold to route the lookup request.

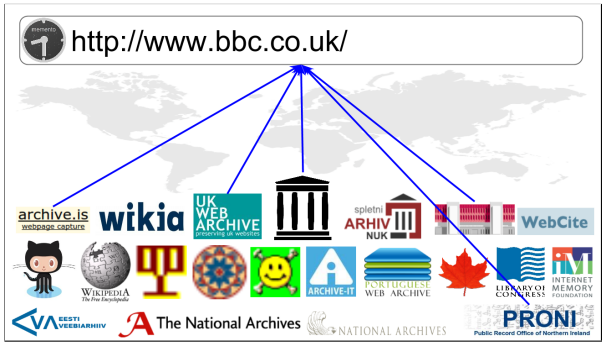
Previous work proved that simple rules are insufficient to accurately model a web archive's holdings [40, 39] (see Section 3.9.2 of Chapter 3 for more details). For example, simply routing requests for `*.uk` URIs to the UK National Archives is insufficient: many other archives hold `*.uk` URIs, and the UK National Archives holds much more than just `*.uk` URIs. This is true for the many other national web archives as well, such as Arquivo.pt is not limited to `*.pt` and Vefsafn.is is not limited to `*.is` URIs. Earlier, we discussed the case of the first copy of the Smithsonian Institution's homepage was found in Arquivo.pt (as shown in Figure 1a), which is not a page of primary interest of Portuguese people.

Using an informed routing at Memento aggregators can avoid overloading archives of any size, where the aggregator only makes requests to a selected subset of archives where a given lookup URI is likely to be found. It can help new archives with limited resources to flourish while still being utilized when people look for archived mementos held by them. This can result in an efficient aggregation, decreased response times, and good user experience while saving resources of users, aggregators, and web archives.

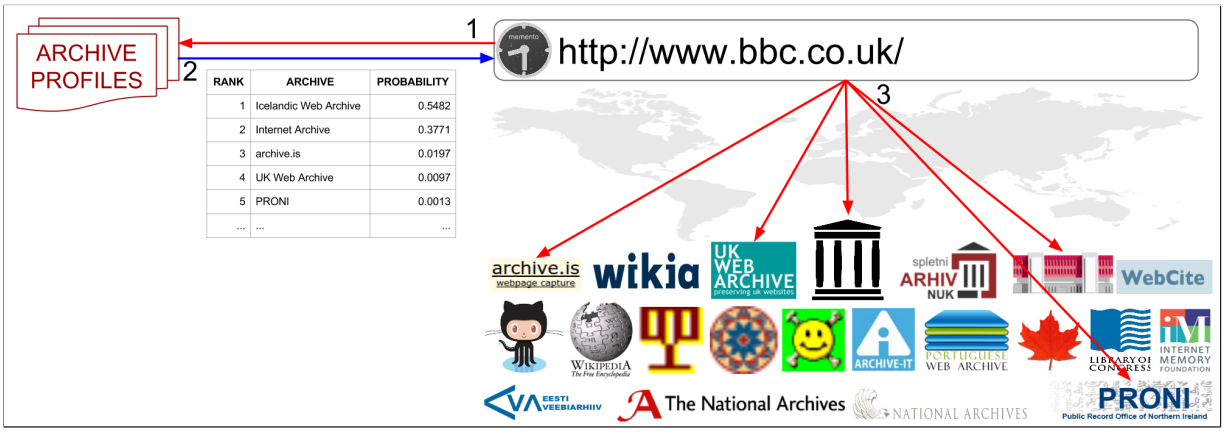
[22] `https://conifer.rhizome.org/`
[23] `https://www.archiefweb.eu/`
[24] `https://collections.internetmemory.org/`



(a) Request to All Known Archives



(b) Response from a Few Archives



(c) Memento Aggregator With Profile Based Routing

Fig. 11. Request-Response Cycle of Memento Aggregator and Various Archives

1.2 RESEARCH QUESTIONS

The above motivational examples helped identify the problem space that is the focus of this research. The process of learning involves gathering information from different sources, organizing the knowledge in a way that is easy to understand and express, and utilizing the knowledge to solve other problems. We explore the following three primary research questions and evaluate their potential solutions.

- **RQ1**: How to learn about the holdings and voids of an archive?
- **RQ2**: How to build an archive profile that will best summarize an archive's holdings/voids and allow for dissemination and exchange?
- **RQ3**: How to utilize archive profiles for the routing of URI lookup requests?

1.2.1 LEARNING ABOUT THE HOLDINGS OF AN ARCHIVE

Different people, based on their association with a web archive (or lack thereof), may have varying levels of access to information of that archive's holdings. Web archives generally have an index of all the URIs they have collected one or more times. Having access to such indexes can give a complete knowledge about their holdings. However, this information is often not available publicly due to some practical problems such as logistics, security, or privacy concerns. Entities that do not have direct access to an archive's index can still learn about its holdings by observing externally. We can perform searches using random keywords in web archives that support fulltext search and note all the URIs returned in their responses. This way, we can learn about a few new URIs from the archive's holdings with each search. In the case of web archives that do not support fulltext search, we can observe their responses as we lookup individual URIs.

Each of these approaches have their own learning rate, associated cost, and usefulness that needs to be studied and analyzed. There are many opportunities of optimization in each of these approaches. For example, we need to identify a mechanism to generate a list of keywords that yields maximum number of unique URIs in search results in the least number of searches in a given web archive. Similarly, in the case where fulltext search is not available and learning rate is one URI per lookup, we need to find a suitable list of diverse URIs that maximizes the understanding of archive's holdings.

The Random Searcher Model, one of our contributions (as discussed in Section 6.4.1 of Chapter 6), provides a means of discovery of an archive's holdings via fulltext search. Access logs collected by the running instances of MemGator, another contribution of this work (described in Chapter 5), provide a longitudinal data of which URI lookups were successful or unsuccessful in various web archives over time.

1.2.2 EXPRESSING AND DISSEMINATING ARCHIVE PROFILES

As we learn about the holdings of various archives, it becomes important to store and organize that knowledge to make it useful. We call this organized knowledge about a web archive's holdings an *Archive Profile*. The purpose of an archive profile is to express the holdings of an archive by the archive owners themselves or by a third party. A simple archive profile can be a list of URIs present in an archive along with the date and time when the archive observed each of those URIs. However, such simple profiles can become significantly large for large archives. On the contrary, if a profile only stores the top-level

domains (TLDs) that are present in an archive, the profile will be small in size, but not very effective for most of the use cases. We can also find a suitable middle ground of a partial URI (such as just the domain name or up to one path depth) which is between the two extremes of full URI and only the TLD. Alternatively, we can have full URIs recorded for sites that are not well archived, but relatively shorter partial URIs for well-archived sites to minimize the repetition and profile size. So, there is a tradeoff among factors like frequency of updates, accuracy, size of the profile, granularity of details, and usefulness.

Depending on the purpose of the profile, one might want to store more metadata about each record in the profile (e.g., MIME type, byte size, HTTP status code, or access restrictions). Another possibility is to only record a unique URI once and not repeat it each time it was observed by the archive; instead, write down the number of observations made to that URI. One might want to organize URIs in buckets of time slots of varying granularity such as yearly, monthly, or daily. These profiles can be generated by different entities at different times with different levels of details.

There is a need to define a suitable format for archive profiles that is flexible enough to express varying levels of details, easy to parse and understand, compact in size, easy to merge with other profiles or split arbitrarily in small chunks, and allows incremental growth and versioning. The format should also be suitable for publishing on the web and easy to exchange with other parties.

We introduced a file format called, Unified Key Value Store (UKVS), which fulfills these needs of archive profiles' serialization and dissemination (as described in Chapter 8). Furthermore, we implemented the *MementoMap* tool (described in Section 5.6 of Chapter 5) that ingests a list of URIs collected from CDX files, search results, access logs, or any other means and produces a *MementoMap* of the archive in the UKVS format based on supplied options to control the level of details.

1.2.3 ARCHIVE PROFILES FOR MEMENTO ROUTING

While there can be many use cases of archive profiles, the primary focus of this work is to efficiently route memento aggregation requests to appropriate archives. We can build algorithms to calculate routing scores for a given lookup URI to be present in different archives based on the information in their corresponding profiles. Alternatively, we can build binary or probabilistic classifiers for each archive (using data in their corresponding profiles) to predict whether a given lookup URI is present in them individually. These approaches can be useful when the number of archives to be aggregated is not significantly

large, as these approaches are not scalable beyond a limit.

Another way to look at this problem is to convert it into an information retrieval (IR) problem and utilize many existing well-established techniques for the task such as language modeling or vector space modeling [91]. In simple terms, IR is the task of searching for resources from a collection that are relevant to a given query. This is generally performed on a collection of documents where search keywords are generally words that appear in those documents. In the case of memento routing, we can have a collection of archive profiles where profiles are analogous to documents and the URIs present in them are analogous to words. Then the task of memento routing converts into the task of finding relevant profiles (hence, related archives) for a given URI lookup. It is worth noting that many IR tasks (such as tokenization and stemming) will be different for URIs than for conventional documents. Also, the document size (both the byte size as well as the word count) can be several orders of magnitude larger for large archive profiles than general text documents.

1.3 CHAPTER SUMMARY

In this chapter we briefly described web archives, the Memento framework, Memento aggregation and archive profiles. With the help of some real life examples we highlighted the need for aggregating archives. We described that the rate of growth of the web will likely always be faster than the ability to archive it, so we will always need techniques to efficiently aggregate as many archives as possible, no matter how large certain archives get. We gave examples of censorship of web archives, transient errors and attacks, lack of timely captures of important events in a single archive, potentially unintentional exclusions, a means for the validity and fixity, and the lack of variety in mementos. Then, we demonstrated the significance of aggregating small archives by showing the little overlap among archives. We showed that even small archives may bring some unique archived resources when aggregated. We also highlighted the significance of small archives due to their focus on niche features such as high-fidelity captures.

Furthermore, we described the usefulness of understanding the holdings of various web archives by creating their archive profiles. We noted that even the curated collections start to include many undesired resources and exclude many desired ones over time where archive profiles can be helpful to understand their holdings. We also described the role of archive profiles in expressing dark or restricted holdings in collections that are otherwise not accessible publicly.

Using the example of an incident where LC was unable to handle a sudden surge of

traffic, we demonstrated how broadcasting lookup requests can be wasteful and problematic. This establishes the need for a smart routing in Memento aggregators to avoid overloading archives and to allow new archives to flourish. We envisioned that archive profiles can be a means to implement an efficient and smart Memento aggregator.

We then established our three primary research questions related to learning holdings of web archives, expressing and disseminating this information, and utilizing it for memento aggregation routing. Finally, we described what we explore about each of these questions and how.

The remainder of this document is organized as follows:

- Chapter 2 gives background information about various related terminologies that will be helpful to understand the problems this work is addressing and their solutions.
- Chapter 3 explores and reviews published scholarly works in various related areas.
- Chapter 4 describes our *MementoMap* framework, research questions, and evaluation plan.
- Chapter 5 describes various tools we built during our research as part of deliverables and to help our research process.
- Chapter 6 describes our work around **RQ1** in which we explore different approaches to learn about holdings of web archives with different levels of access.
- Chapter 7 describes our work around **RQ1** in which we discuss how to learn about resources that are not present in a web archive.
- Chapter 8 describes our work around **RQ2** in which we explore serialization and dissemination methods of archive profiles.
- Chapter 9 describes our work around **RQ3** in which we combine profiles of various web archives and route Memento lookup queries to suitable archives.
- Chapter 10 enumerates our contributions, discusses potential future work, and concludes by summarizing this work.

CHAPTER 2

BACKGROUND

In order to understand web archive profiling it would be helpful to be familiar with some related terminologies. In this chapter we briefly describe Hypertext Transfer Protocol, web archiving, Memento, aggregation, URI transformations, and some file formats.

2.1 HYPERTEXT TRANSFER PROTOCOL (HTTP)

Hypertext Transfer Protocol (HTTP) is a stateless client and server communication protocol [103, 102, 110, 33, 26]. HTTP is a widely adopted protocol on the World Wide Web (WWW). The monolithic specification of HTTP version 1.1 was later split in small pieces, each covering a specific aspect of the protocol [108, 109, 107, 104, 105, 106]. HTTP up to version 1.1 has been a text-based protocol, but HTTP version 2 and later are binary [235, 58]. In this work we primarily illustrate and focus on HTTP/1.1 as the later versions, despite being binary encoded, are conceptually the same, especially, in the aspects of HTTP this work has the focus on.

HTTP is relevant to this work at many levels. For example, Web archives preserve HTTP exchanges and replay them using HTTP later.

2.1.1 HTTP MESSAGE

In HTTP a client user agent (such as a web browser) sends an *HTTP Request Message* to a server using one of the supported methods (e.g., `GET`, `POST`, `PUT`, or `DELETE`), some headers, and an optional payload. The server processes the request, and returns an appropriate *HTTP Response Message* back to the client with a suitable status code (e.g., `200`, `201`, `301`, `404`, or `500`), some headers, and an optional payload.

Figure 12 illustrates a typical HTTP request response exchange. The second line shown in the figure is called a *Request Line* that contains the HTTP method, request URI, and the HTTP version (in this example, "`GET`", "`/hello`", and "`HTTP/1.1`" respectively). Following the *Request Line* there can be zero or more consecutive lines of headers with their names and values separated by a colon sign (in this example, lines 3 and 4). Headers are terminated by an empty line followed by an optional payload, which is absent in this example request

```
1  $ curl -ivs https://example.com/ 2>&1 | grep "^<\|^>"
2  > GET /hello HTTP/1.1
3  > Host: example.com
4  > User-Agent: curl/7.58.0
5  >
6  < HTTP/1.1 200 OK
7  < Date: Thu, 08 Aug 2019 15:47:15 GMT
8  < Server: Apache
9  < Last-Modified: Fri, 02 Aug 2019 03:35:56 GMT
10 < Content-Type: text/plain
11 < Content-Length: 16
12 <
13 < Hello Archives!
```

Fig. 12. HTTP Request and Response Messages (Characters > and < are not part of the messages, they are annotations to identify request and response portions, respectively.)

message. Line 6 is the start of an *HTTP Response Message* which is called a *Status Line* that consists of the HTTP version, status code, and a status message (in this example, "`HTTP/1.1`", "`200`", and "`OK`" respectively). Same as the request message, the response message has consecutive lines of headers, an empty line, and an optional payload (in this example, lines 7–11 are headers and line 13 is the response payload).

2.1.2 HTTP METHOD

An HTTP method is a verb that indicates a generic action a client intends to perform on a resource identified by a URI. It is a single token written using all upper-case letters such as GET, POST, DELETE, etc. The base HTTP specification defines various common methods, but other specifications extend the HTTP method space to include more methods. Table 1 lists common HTTP methods among which the GET and POST are the most utilized methods on the web [18]. A safe HTTP method does not change the state of the resource, hence has no side-effects (with the exception of logging or change in the last access time). An idempotent method means the overall effect of repeating the request is going to be the same as a single successful attempt. Traditional web archiving systems focused only on the GET method, both for capture and replay, but modern systems are accounting for other methods as well.

Table 1. Common HTTP Methods

Method	**Description**	**Safe**	**Idempotent**
GET	Retrieve a representation of a resource	Yes	Yes
HEAD	Same as GET, but omits the payload	Yes	Yes
POST	Create a new resource	No	No
PUT	Create or update a resource at a given URI	No	Yes
PATCH	Partially update a resource	No	Yes
DELETE	Remove a resource	No	Yes
OPTIONS	Query available communication options	Yes	Yes
CONNECT	Create a tunnel	Yes	Yes
TRACE	Echo back the request	Yes	Yes

Table 2. Purpose of Various HTTP Status Code Classes

Status Class	**Code Range**	**Description**
1xx	100–199	Informational responses
2xx	200–299	Successful responses
3xx	300–399	Redirects
4xx	400–499	Client errors
5xx	500–599	Server errors

2.1.3 HTTP STATUS CODE

When a client sends an HTTP request to an HTTP server, the server communicates the overall status of the HTTP exchange using response status codes. These status codes are three-digit decimal numbers ranging from 100 to 599. The leftmost digit describes one of the five response status classes as described in Table 2. Many existing HTTP-related Request for Comments (RFCs) standardize various status codes and their semantics in each class, but many codes in each range are still available for future extension. However, even if a client does not know the exact semantics of a given status code, the class of the code still gives a generic understanding of the response.

Status codes in the `1xx` class are informational in nature such as, `100 Continue` (an

interim response to indicate everything OK so far, wait for the actual response) or `103 Early Hints` (to tell the client what other requests it might want to make in order to preload necessary resources). This class of status codes is not very common in practice.

Status codes in the `2xx` class indicate success. The `200 OK` status is the most popular (and often misused) status code, but other common codes in the class include `201 Created` (when one or more resources are creates as a result of a request), `202 Accepted` (when the request is accepted for asynchronous processing), `204 No Content` (when the request is successfully, but there is no content to send), and `206 Partial Content` (when only a subset of the response payload is returned).

Status codes in the `3xx` class indicate a redirect. The response may contain a `Location` header that the client may choose to follow to access the resource where the server is redirecting. Common status codes of this class include `300 Multiple Choices` (when a content-negotiation cannot be resolved to one unique representation of the resource by the server and results in an agent-driven negotiation), `301 Moved Permanently` (when the resource is moved to a new location, often used when domain names or URI structure of a website are changed), `302 Found` (when the resource is available at a different URI, but the server wants the client to continue to come back to the current URI for content negotiation in the future), `303 See Other` (when a related resources is available at a different URI), and `304 Not Modified` (when the content is not modified as per the conditions provided in the request and the client can leverage the corresponding cached resources instead). Status codes `307 Temporary Redirect` and `308 Permanent Redirect` are similar to the `302` and `301`, respectively, with one difference that these indicate that the client should not change the request method when following the redirect.

Status codes in the `4xx` class indicate a client error. Generally, a client should not attempt to repeat the request that received a status of this class without making any changes to the request. The `404 Not Found` (the server does not have a resource representation for the target resource or is not willing to disclose) is the most common status code in this class. Some other common status codes in the `4xx` class include `400 Bad Request` (when the request message is malformed), `401 Unauthorized` (when authentication is needed to access the resource), `403 Forbidden` (when an authenticated user is not allowed to access the resources), `405 Method Not Allowed` (when the requested HTTP method is not allowed on the resource), `410 Gone` (when the resource identified by the URI is gone forever), `413 Payload Too Large` (when the request payload is larger than configured for the server), and `414 URI Too Long` (when the URI is larger than configured for an HTTP server).

```
$ curl -i https://example.com/absent.html
HTTP/1.1 200 OK
Content-Type: text/plain
Date: Thu, 08 Aug 2019 21:13:04 GMT
Server: Apache
Content-Length: 40

Sorry, the requested page was not found!
```

Fig. 13. A Sample Soft-404 Response

Status codes in the `5xx` class indicate a server error. Common status codes in the class include `500 Internal Server Error` (when the server encounters an unhandled error while processing the request), `501 Not Implemented` (when the requested HTTP method is not implemented), `502 Bad Gateway` (when the server acting as a gateway to another service like a database or an external API received an invalid response), `503 Service Unavailable` (when the server is overloaded or down for maintenance, generally provides a `Retry-After` header), `504 Gateway Timeout` (when the server is acting as a gateway and the upstream service did not return a response in time), and `505 HTTP Version Not Supported` (when the request HTTP version is unsupported by the server).

2.1.4 SOFT-404

As per the HTTP standards, if a resource is accessed that is not present on the requested URI, the server should return the `404 Not Found` status code, and if the resource is present and is accessible then the response should be `200 OK`. However, some poorly written web applications may return `200 OK` status code even for resources that are not present (as illustrated in Figure 13). They often advertise the unavailability of the resource via the response body instead of the status code. This behavior is called Soft-404 [180]. Moreover, the term Soft-404 is commonly used as an umbrella term for any error page that is returned with the `200 OK` response code due to the prevalence of `404 Not Found` errors over other error pages on the web. Soft-404 is relevant to our work as some poorly implemented web archives may return it, As a result, we may need to find workarounds to identify them and isolate them from real `200 OK` responses.

```
172.17.0.1 - - [13/Nov/2020:19:01:18 +0000] "GET / HTTP/1.1" 200 45
↪  "-" "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML,
↪  like Gecko) Chrome/87.0.4280.66 Safari/537.36"
172.17.0.1 - - [13/Nov/2020:19:01:18 +0000] "GET /favicon.ico
↪  HTTP/1.1" 200 238 "http://localhost/" "Mozilla/5.0 (X11; Linux
↪  x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.66
↪  Safari/537.36"
172.17.0.1 - - [13/Nov/2020:19:02:42 +0000] "GET /img/logo.png
↪  HTTP/1.1" 200 2906 "http://localhost/" "Mozilla/5.0 (X11; Linux
↪  x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.66
↪  Safari/537.36"
172.17.0.1 - - [13/Nov/2020:19:03:12 +0000] "GET /img/unicorn.svg
↪  HTTP/1.1" 404 196 "http://localhost/" "Mozilla/5.0 (X11; Linux
↪  x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.66
↪  Safari/537.36"
172.17.0.9 - - [13/Nov/2020:19:05:36 +0000] "POST /login HTTP/1.1" 401
↪  234 "-" "curl/7.58.0"
```

Fig. 14. A Sample Extended Access Log

2.2 HTTP ACCESS LOGS

Many HTTP servers write a log in the standard *Common Log Format* or extended *Combined Log Format* [41]. Figure 14 illustrates a typical access log file. It is a textual file format that contains the following information about all the HTTP accesses, when available (where any missing field is represented with a hyphen):

- IP address of the client
- User identity (often empty)
- Authenticated user's ID (often empty)
- Date and time
- HTTP method, request path, and HTTP version
- HTTP status code
- Size of the response in bytes

Extended log versions may include additional fields as shown below:

- Referrer
- User-agent
- Cookies
- Request processing duration
- Any other headers if configured

However, not many web webmasters configure their server logging to include many useful headers beyond the default fields (which usually only include referrer and user-agent on top of the Common Log Format). For example, it would have been useful to know what language preferences various clients had when accessing different resources from web archives, but the access logs we collected from different web archives did not include this information. In this work, we leverage access logs of web archives to learn about frequently accessed resources that are not present in the archive. To process web archive access logs we have implemented a generic HTTP access log parser with added capabilities for web archives (as discussed in Chapter 5).

2.3 WEB ARCHIVING AND WEB ARCHIVES

Web archiving is the practice of collecting and preserving subsets of the WWW with the intent to make them available in the future as historical records. The scope of web archiving can be anything from a single web page saved by an individual for personal records to the entire WWW captured by an institute or organization.

Institutions that capture the Web for preservation are called web archives. For example, the Internet Archive is a large-scale web archive, founded in 1996 as a non-profit organization. However, there are various other commercial, non-profit, institutional, and national web archives around the world with different policies, goals, and focuses [51, 52, 241]. Some web archives are on-demand (e.g., Archive.is and WebCite [100, 101]) as they capture web pages and all the page requisites when requested explicitly by a client. Other web archives are crawler-based that are configured to crawl certain portions of the WWW using standard web crawling techniques.

The act of web archiving usually involves the following tasks [179] (our work relates to and/or contributes to most of these aspects of web archiving as illustrated in Figure 36 of Chapter 5 on Page 82):

- Collection policy making, seed selection, and crawling frequency configuration
- Capturing/Crawling
- Data storage
- Indexing
- Replay

Usually the coverage of the archival collection depends on the collection policy and frequency of crawling, while the quality of capture depends on the tools used to capture the web. Various open-source tools for capture/crawling, indexing, and replay are available that can be used to set up a small or large-scale personal or institutional web archive.

2.3.1 ARCHIVE COLLECTION POLICY

Different web archives have different archival collection policies that control what gets archived and what gets excluded. These policies shape the holdings and voids of a web archive. For example, a web wide crawler can have more widespread coverage while crawling popular resources more frequently. In contrast a focused crawler may collect many deep URIs from the domains in scope that might be missed in a shallow web wide crawl. On-demand one-page-at-a-time archives (e.g., Perma.cc [95, 205] and Archive.is) collect only pages (and their page requisites) that users show an interest in, which may include deep links that would otherwise not be discovered, but such archives cannot be as dense as crawler-based archives. Moreover, there are various national web archives (e.g., Arquivo.pt [117, 89] and the UK Web Archive [53, 145, 76]) that perform wide crawls, but they scope their crawlers primarily to their national TLDs/gTLDs. Such crawlers may include resources from other parts of the web, but their density will be low. Similarly, there are event-based collections that focus on crawling resources related to specific events (e.g., COVID19 [134, 136, 218, 57, 135]). These types of collections often require manual seed collection, which limits the number of resources individual curators or collaborators can collect for archiving. In some cases a collection can be made from a curated seed list by setting up a wide or deep crawling policy starting from the seed URIs. A good example of this would be the End of Term crawl [113, 125, 49], which includes a curated list of domains/pages and many federal sites under the `.gov` and `.mil` TLDs that is crawled a few times around the US presidential elections.

2.4 MEMENTO

Memento [245, 246, 189] is a protocol that adds the dimension of time on the web. In other words, it is an HTTP framework for time-based access to resource states. The framework is applicable to any web resource for time-based content negotiation (i.e., a client can negotiate with a server to retrieve a historical version of a resource). However, it has a greater focus towards web archives where time-based content negotiation is at the core.

The Memento framework introduced a uniform Application Programming Interface (API) for web archives to interoperate. This interoperability enabled memento aggregation for cross-archive consolidation of archived resources.

The term *Memento* (as a proper noun) refers to the framework itself. However, the common noun form *memento* is used for a prior state or a timestamped version of the representation of an original resource. A memento is identified by a memento URI (or *URI-M*) which often resolves to an archived copy in a web archive. An original resource is identified by a resource URI (or *URI-R*) which used to exist in the past and might still exist.

2.4.1 TIMEGATE

A *TimeGate* is a resource that acts as a gateway for datetime negotiation for a *URI-R*. A *TimeGate* of a *URI-R* is identified by a corresponding *TimeGate URI* (or *URI-G*). The client sends an *Accept-Datetime* request header to a *URI-G* for datetime negotiation. The *TimeGate* then identifies the closest memento with respect to the requested datetime and returns a `302 Found` response with the corresponding *URI-M* in the `Location` header as illustrated in Figure 15. The Memento framework covers various datetime negotiation patterns in detail under Section 4 of its specifications [245]. A valid memento response contains a *Memento-Datetime* response header that represents the datetime of a prior state of the resource.

A *TimeGate* is an intermediate layer that may exist on the same host as the *URI-R*, on a web archive, or on a Memento aggregator that consolidates various archives using their *TimeGate* endpoints. If a web service implements the Memento protocol, ideally all the *URI-Rs* of that service should advertise their *URI-Gs* in the `Link` response header. However, if a *URI-R* does not advertise its *TimeGate* or does not exist anymore then a generic *TimeGate* endpoint of a web archive (or a Memento aggregator) can be used, which

```
1  $ curl -I https://www.w3.org/wiki/Main_Page
2  HTTP/1.1 200 OK
3  Date: Tue, 16 Jul 2019 03:16:01 GMT
4  Link: <https://www.w3.org/wiki/Main_Page>; rel="original latest-version",
5        <https://www.w3.org/wiki/Special:TimeGate/Main_Page>; rel="timegate",
6        <https://www.w3.org/wiki/Special:TimeMap/Main_Page>; rel="timemap";
7          type="application/link-format"; from="Thu, 01 Jan 1970 00:00:00 GMT";
8          until="Fri, 16 Nov 2018 19:10:23 GMT",
9        <https://www.w3.org/wiki/index.php?title=Main_Page&oldid=30366>; rel="first memento";
10         datetime="Thu, 01 Jan 1970 00:00:00 GMT",
11       <https://www.w3.org/wiki/index.php?title=Main_Page&oldid=108148>; rel="last memento";
12         datetime="Fri, 16 Nov 2018 19:10:23 GMT"
13 Content-Language: en
14 Vary: Accept-Encoding,Cookie
15 Cache-Control: s-maxage=18000, must-revalidate, max-age=0
16 Last-Modified: Mon, 15 Jul 2019 22:16:01 GMT
17 Content-Type: text/html; charset=UTF-8
18
19 $ curl -IL -H "Accept-Datetime: Sat, 20 Dec 2014 12:30:00 GMT" \
20       https://www.w3.org/wiki/Special:TimeGate/Main_Page
21 HTTP/1.1 302 Found
22 Date: Tue, 16 Jul 2019 03:16:21 GMT
23 Vary: Accept-Encoding,Cookie,Accept-Datetime
24 Location: https://www.w3.org/wiki/index.php?title=Main_Page&oldid=80125
25 Link: <https://www.w3.org/wiki/Special:TimeMap/Main_Page>; rel="timemap";
26         type="application/link-format"; from="Thu, 01 Jan 1970 00:00:00 GMT";
27         until="Fri, 16 Nov 2018 19:10:23 GMT",
28       <https://www.w3.org/wiki/index.php?title=Main_Page&oldid=30366>; rel="first memento";
29         datetime="Thu, 01 Jan 1970 00:00:00 GMT",
30       <https://www.w3.org/wiki/index.php?title=Main_Page&oldid=108148>; rel="last memento";
31         datetime="Fri, 16 Nov 2018 19:10:23 GMT",
32       <https://www.w3.org/wiki/Main_Page>; rel="original latest-version"
33 Content-Type: text/html; charset=UTF-8
34
35 HTTP/1.1 200 OK
36 Date: Tue, 16 Jul 2019 03:16:23 GMT
37 X-Content-Type-Options: nosniff
38 Memento-Datetime: Sat, 20 Dec 2014 11:34:08 GMT
39 Link: <https://www.w3.org/wiki/Main_Page>; rel="original latest-version",
40       <https://www.w3.org/wiki/Special:TimeGate/Main_Page>; rel="timegate",
41       <https://www.w3.org/wiki/Special:TimeMap/Main_Page>; rel="timemap";
42         type="application/link-format"; from="Thu, 01 Jan 1970 00:00:00 GMT";
43         until="Fri, 16 Nov 2018 19:10:23 GMT",
44       <https://www.w3.org/wiki/index.php?title=Main_Page&oldid=30366>; rel="first memento";
45         datetime="Thu, 01 Jan 1970 00:00:00 GMT",
46       <https://www.w3.org/wiki/index.php?title=Main_Page&oldid=108148>; rel="last memento";
47         datetime="Fri, 16 Nov 2018 19:10:23 GMT"
48 Content-Language: en
49 Vary: Accept-Encoding,Cookie
50 Expires: Thu, 01 Jan 1970 00:00:00 GMT
51 Cache-Control: private, must-revalidate, max-age=0
52 Content-Type: text/html; charset=UTF-8
```

Fig. 15. Content Negotiation Using the Memento TimeGate of the Original Server to Access a Specific Version of a Wiki Page

often allows a template-based *URI-G* formation where the *URI-R* is passed as a path or query parameter as illustrated in Figure 16.

Figure 15 illustrates the case where an original resource itself supports temporal content negotiation. We make a *HEAD* request to a Wiki page (line 1, where "`-I`" flag asks `curl` to make a *HEAD* request). The server responds with the status code `200` and a *Link*

```
1  $ curl -IL -H "Accept-Datetime: Sat, 20 Dec 2014 12:30:00 GMT" \
2        https://web.archive.org/web/https://example.com/
3  HTTP/1.1 302 FOUND
4  Server: nginx/1.15.8
5  Date: Mon, 20 Jul 2020 20:11:04 GMT
6  Vary: accept-datetime
7  Location: https://web.archive.org/web/20141220124831/http://www.example.com/
8  Link: <https://example.com/>; rel="original",
9        <https://web.archive.org/web/20141220124831/http://www.example.com/>;
10         rel="memento"; datetime="Sat, 20 Dec 2014 12:48:31 GMT",
11       <https://web.archive.org/web/timemap/link/https://example.com/>;
12         rel="timemap"; type="application/link-format"
13 Content-Type: text/plain; charset=utf-8
14 Content-Length: 0
15 Connection: keep-alive
16
17 HTTP/1.1 200 OK
18 Server: nginx/1.15.8
19 Date: Mon, 20 Jul 2020 20:11:07 GMT
20 Memento-Datetime: Sat, 20 Dec 2014 12:48:31 GMT
21 Link: <http://www.example.com/>; rel="original",
22       <https://web.archive.org/web/http://www.example.com/>; rel="timegate",
23       <https://web.archive.org/web/timemap/link/http://www.example.com/>;
24         rel="timemap"; type="application/link-format",
25       <https://web.archive.org/web/20020120142510/http://example.com:80/>;
26         rel="first memento"; datetime="Sun, 20 Jan 2002 14:25:10 GMT",
27       <https://web.archive.org/web/20141220115400/http://example.com/>;
28         rel="prev memento"; datetime="Sat, 20 Dec 2014 11:54:00 GMT",
29       <https://web.archive.org/web/20141220124831/http://www.example.com/>;
30         rel="memento"; datetime="Sat, 20 Dec 2014 12:48:31 GMT",
31       <https://web.archive.org/web/20141220135303/http://www.example.com/>;
32         rel="next memento"; datetime="Sat, 20 Dec 2014 13:53:03 GMT",
33       <https://web.archive.org/web/20200720172043/https://example.com/>;
34         rel="last memento"; datetime="Mon, 20 Jul 2020 17:20:43 GMT"
35 Content-Type: text/html; charset=utf-8
36 Content-Length: 4064
37 X-Archive-Orig-Accept-Ranges: bytes
38 X-Archive-Orig-Cache-Control: max-age=604800
39 X-Archive-Orig-Date: Sat, 20 Dec 2014 12:48:31 GMT
40 X-Archive-Orig-Etag: "359670651"
41 X-Archive-Orig-Expires: Sat, 27 Dec 2014 12:48:31 GMT
42 X-Archive-Orig-Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
43 X-Archive-Orig-Server: ECS (rhv/818F)
44 X-Archive-Orig-X-Cache: HIT
45 X-Archive-Orig-x-ec-custom-error: 1
46 X-Archive-Orig-Content-Length: 1270
47 X-Archive-Orig-Connection: close
48 X-Archive-Guessed-Content-Type: text/html
49 X-Archive-Guessed-Charset: utf-8
50 Content-Security-Policy: default-src 'self' 'unsafe-eval' 'unsafe-inline'
51       data: blob: archive.org web.archive.org analytics.archive.org pragma.archivelab.org
52 Connection: keep-alive
```

Fig. 16. Content Negotiation Using a Generic Memento TimeGate of the Internet Archive to Access an Archived Version of a Resource When the Origin Server Is Not Available or Does Not Support Memento

header that contains a link with relation type "`timegate`" (line 5). This response suggests that the original Wiki page that we are interested in has an associated *TimeGate* resource available from where we can access its past states. We then use this newly discovered *URI-G* and make another *HEAD* request with an *Accept-Datetime* header to access a version of

the resource as it was on `Dec 20, 2014`, at `12:30:00` (lines 19–20). The server returns a `302` response code (line 21), which suggests that a resource was found that satisfies the request and it can be accessed from the URI returned in the *Location* header (line 24). The response returned includes *Accept-Datetime* in its *Vary* header (line 23) as an indicator of availability of temporal content negotiation. We follow the `302` redirect automatically (as `curl` is configured to follow the *Location* header in line 19 using "`-L`" flag) to access the matching memento. We receive a `200` status code (line 35) and a *Memento-Datetime* header (line 38) that indicates a version of the resource at `11:34:08` (about 25 minutes prior to the requested time). Since a wiki is a content management system that preserves a history of all of its changes [146], we can believe that the state of the resource at the requested time in the past was the same as in the response, although the response indicated a slightly older version when the resource was changed (i.e., the closest past version).

Figure 16 illustrates the case where we look for historical versions of an original resource using a generic *TimeGate* endpoint of a web archive independent from the original resource. In this example, we performed content negotiation on a resource from `example.com` in the temporal dimension in an independent domain `web.archive.org`, which is a web archive. We skip the first step of discovering a link relation type "`timegate`" for a *URI-R* and use our out of band knowledge to form a *URI-G* (as shown in line 2 where the *URI-R* `https://example.com/` is concatenated to the generic *TimeGate* endpoint of the web archive). The remaining steps are the same as described above in the previous example. It is worth noting that the actual state of the resource at the requested datetime might have been different from what we get from the web archive, which is a version observed after about 19 minutes from the requested datetime (i.e., closest future version).

2.4.2 TIMEMAP

A *TimeMap* is a collection resource that lists links and their relations with an original resource. A *TimeMap* of a *URI-R* is identified by a corresponding *TimeMap URI* (or *URI-T*). The primary purpose of a *TimeMap* is to list all mementos with their *URI-Ms* and corresponding *Memento-Datetime* as illustrated in Figure 17. Generally, the media type of the entity body of a *TimeMap* resource is "`application/web-link`" [192, 221], but recent implementations are experimenting with alternate formats like *JSON*.

2.4.3 NOT-ARCHIVED VS. ARCHIVED-404

A web archival replay system is yet another HTTP server that replays HTTP responses

```
$ curl -i https://web.archive.org/web/timemap/link/http://example.org/index.html
HTTP/1.1 200 OK
Server: nginx/1.15.8
Date: Fri, 09 Aug 2019 21:58:23 GMT
Content-Type: application/link-format
Transfer-Encoding: chunked
Connection: keep-alive

<http://www.example.org:80/index.html>; rel="original",
<https://web.archive.org/web/timemap/link/http://example.org/index.html>; rel="self";
  type="application/link-format"; from="Wed, 16 Oct 2002 10:13:37 GMT",
<https://web.archive.org/web/http://example.org/index.html>; rel="timegate",
<https://web.archive.org/web/20021016101337/http://www.example.org:80/index.html>;
  rel="first memento"; datetime="Wed, 16 Oct 2002 10:13:37 GMT",
<https://web.archive.org/web/20031207031049/http://www.example.org:80/index.html>;
  rel="memento"; datetime="Sun, 07 Dec 2003 03:10:49 GMT",
<https://web.archive.org/web/20040305230707/http://www.example.org:80/index.html>;
  rel="memento"; datetime="Fri, 05 Mar 2004 23:07:07 GMT",
[... TRUNCATED ...]
<https://web.archive.org/web/20190731130056/http://www.example.org/index.html>;
  rel="memento"; datetime="Wed, 31 Jul 2019 13:00:56 GMT",
```

Fig. 17. A TimeMap from the Internet Archive

captured from other HTTP servers in the past with necessary changes. This means there are situations where the client may not be sure whether a status code reflects the state of a resource on the web archival server or the original server corresponding to the URI-R. For example, when accessing URI-M, if the archive returns `404 Not Found`, should this be interpreted as the archive does not have that memento (i.e., not-archived) or the crawler of the archive received a `404 Not Found` response from the origin server when attempting to access the corresponding resource in the past, which is being replayed (i.e., archive-404). Similarly, there can be confusion about other HTTP status codes in the redirect, client error, and server error status classes. To resolve this confusion we look for signatures of a memento in the response headers. If the response contains the `Memento-Datetime` header and a `Link` header with relation type `memento`, then we know it is an archived resource with the corresponding status code, otherwise we consider that the status code reflects the state of the resource in the web archive server. Figure 18 illustrates the two cases in which the `example.com/absent.html` is not archived, but the `example.com/404.html` is archived with the `404 Not Found` status code.

2.5 MEMENTO AGGREGATOR

A *Memento Aggregator* is a service that consolidates various web archives using the *Memento* protocol. The primary function of a Memento aggregator is to provide HTTP

```
1  $ curl -I https://web.archive.org/web/20201109011314/http://example.com/absent.html
2  HTTP/1.1 404 Not Found
3  Server: nginx/1.15.8
4  Date: Wed, 11 Nov 2020 00:02:04 GMT
5  Content-Type: text/html; charset=utf-8
6  Connection: keep-alive
7
8  $ curl -I https://web.archive.org/web/20201109011314/http://example.com/404.html
9  HTTP/1.1 404 Not Found
10 Server: nginx/1.15.8
11 Date: Wed, 11 Nov 2020 00:03:16 GMT
12 Memento-Datetime: Mon, 09 Nov 2020 01:13:14 GMT
13 Link: <http://example.com/404.html>; rel="original",
14       <https://web.archive.org/web/http://example.com/404.html>; rel="timegate",
15       <https://web.archive.org/web/timemap/link/http://example.com/404.html>;
16         rel="timemap"; type="application/link-format",
17       <https://web.archive.org/web/20020401195232/http://www.example.com:80/404.html>;
18         rel="first memento"; datetime="Mon, 01 Apr 2002 19:52:32 GMT",
19       <https://web.archive.org/web/20200912032554/http://example.com/404.html>;
20         rel="prev memento"; datetime="Sat, 12 Sep 2020 03:25:54 GMT",
21       <https://web.archive.org/web/20201109011314/http://example.com/404.html>;
22         rel="memento"; datetime="Mon, 09 Nov 2020 01:13:14 GMT",
23       <https://web.archive.org/web/20201109011314/http://example.com/404.html>;
24         rel="last memento"; datetime="Mon, 09 Nov 2020 01:13:14 GMT"
25 Content-Type: text/html; charset=UTF-8
26 Content-Length: 2885
27 Connection: keep-alive
28 X-Archive-Orig-Age: 257886
29 X-Archive-Orig-Cache-Control: max-age=604800
30 X-Archive-Orig-Date: Mon, 09 Nov 2020 01:13:14 GMT
31 X-Archive-Orig-Expires: Mon, 16 Nov 2020 01:13:14 GMT
32 X-Archive-Orig-Last-Modified: Fri, 06 Nov 2020 01:35:08 GMT
33 X-Archive-Orig-Server: ECS (nyb/1D35)
34 X-Archive-Orig-Vary: Accept-Encoding
35 X-Archive-Orig-X-Cache: 404-HIT
36 X-Archive-Orig-Content-Length: 1256
37 X-Archive-Guessed-Content-Type: text/html
38 X-Archive-Guessed-Charset: utf-8
39 Content-Security-Policy: default-src 'self' 'unsafe-eval' 'unsafe-inline'
40       data: blob: archive.org web.archive.org analytics.archive.org pragma.archivelab.org
```

Fig. 18. Not-Archived vs. Archived-404

API endpoints for consolidated *TimeGate* and *TimeMap*. When a client performs datetime negotiation with a Memento aggregator's *TimeGate* endpoint, the aggregator performs the same negotiation with all the known web archives and returns `302 Found` response by selecting the closest *URI-M* with respect to the requested datetime among various responses received from upstream web archives as shown in Figure 19. The matching Memento in this case is coming from the Internet Archive (as shown in lines 30–65). The actual state of the resource at the requested datetime might have been different from what we get from the web archive, which is a version observed after about 19 minutes from the requested datetime (i.e., closest future version), but we do not have any other mementos closer to the requested datetime in any of the aggregated web archives.

When a client asks an aggregator for the *TimeMap* of a *URI-R*, the aggregator fetches

```
$ curl -IL -H "Accept-Datetime: Sat, 20 Dec 2014 12:30:00 GMT" \
      https://memgator.cs.odu.edu/timegate/https://example.com/
HTTP/1.1 302 Found
Date: Mon, 20 Jul 2020 20:32:00 GMT
Vary: Accept-Datetime
Location: https://web.archive.org/web/20141220124831/http://www.example.com/
Link: <https://example.com/>; rel="original",
      <http://web.archive.bibalex.org:80/web/20020120142510/https://example.com/>;
        rel="first memento"; datetime="Sun, 20 Jan 2002 14:25:10 GMT",
      <https://wayback.archive-it.org/all/20141220085500/https://example.com/>;
        rel="prev memento"; datetime="Sat, 20 Dec 2014 08:55:00 GMT",
      <https://web.archive.org/web/20141220124831/http://www.example.com/>;
        rel="memento"; datetime="Sat, 20 Dec 2014 12:48:31 GMT",
      <https://wayback.archive-it.org/all/20141220171028/https://example.com/>;
        rel="next memento"; datetime="Sat, 20 Dec 2014 17:10:28 GMT",
      <https://wayback.archive-it.org/all/20200719170744/https://example.com/>;
        rel="last memento"; datetime="Sun, 19 Jul 2020 17:07:44 GMT",
      <https://memgator.cs.odu.edu/timemap/link/https://example.com/>;
        rel="timemap"; type="application/link-format",
      <https://memgator.cs.odu.edu/timemap/json/https://example.com/>;
        rel="timemap"; type="application/json",
      <https://memgator.cs.odu.edu/timemap/cdxj/https://example.com/>;
        rel="timemap"; type="application/cdxj+ors",
      <https://memgator.cs.odu.edu/timegate/https://example.com/>; rel="timegate"
Access-Control-Allow-Origin: *
Access-Control-Expose-Headers: Link, Location, X-Memento-Count, Server
Content-Type: text/html; charset=utf-8
Server: MemGator/1.0-rc8

HTTP/1.1 200 OK
Server: nginx/1.15.8
Date: Mon, 20 Jul 2020 20:32:17 GMT
Memento-Datetime: Sat, 20 Dec 2014 12:48:31 GMT
Link: <http://www.example.com/>; rel="original",
      <https://web.archive.org/web/http://www.example.com/>; rel="timegate",
      <https://web.archive.org/web/timemap/link/http://www.example.com/>;
        rel="timemap"; type="application/link-format",
      <https://web.archive.org/web/20020120142510/http://example.com:80/>;
        rel="first memento"; datetime="Sun, 20 Jan 2002 14:25:10 GMT",
      <https://web.archive.org/web/20141220115400/http://example.com/>;
        rel="prev memento"; datetime="Sat, 20 Dec 2014 11:54:00 GMT",
      <https://web.archive.org/web/20141220124831/http://www.example.com/>;
        rel="memento"; datetime="Sat, 20 Dec 2014 12:48:31 GMT",
      <https://web.archive.org/web/20141220135303/http://www.example.com/>;
        rel="next memento"; datetime="Sat, 20 Dec 2014 13:53:03 GMT",
      <https://web.archive.org/web/20200720172043/https://example.com/>;
        rel="last memento"; datetime="Mon, 20 Jul 2020 17:20:43 GMT"
Content-Type: text/html; charset=utf-8
Content-Length: 4064
X-Archive-Orig-Accept-Ranges: bytes
X-Archive-Orig-Cache-Control: max-age=604800
X-Archive-Orig-Date: Sat, 20 Dec 2014 12:48:31 GMT
X-Archive-Orig-Etag: "359670651"
X-Archive-Orig-Expires: Sat, 27 Dec 2014 12:48:31 GMT
X-Archive-Orig-Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
X-Archive-Orig-Server: ECS (rhv/818F)
X-Archive-Orig-X-Cache: HIT
X-Archive-Orig-x-ec-custom-error: 1
X-Archive-Orig-Content-Length: 1270
X-Archive-Orig-Connection: close
X-Archive-Guessed-Content-Type: text/html
X-Archive-Guessed-Charset: utf-8
Content-Security-Policy: default-src 'self' 'unsafe-eval' 'unsafe-inline'
      data: blob: archive.org web.archive.org analytics.archive.org pragma.archivelab.org
Connection: keep-alive
```

Fig. 19. Content Negotiation Using a Generic Memento TimeGate of a Memento Aggregator to Access an Archived Version of a Resource from a Web Archive When the Origin Server Is Not Available or Does Not Support Memento

```
$ curl -i https://memgator.cs.odu.edu/timemap/link/http://example.org/index.html
HTTP/1.1 200 OK
Content-Type: application/link-format
Date: Fri, 09 Aug 2019 21:53:10 GMT
X-Generator: MemGator:1.0-rc7
X-Memento-Count: 162
Transfer-Encoding: chunked

<http://example.org/index.html>; rel="original",
<https://memgator.cs.odu.edu/timemap/link/http://example.org/index.html>; rel="self";
    type="application/link-format",
<https://memgator.cs.odu.edu/timemap/link/http://example.org/index.html>; rel="timemap";
    type="application/link-format",
<https://memgator.cs.odu.edu/timemap/json/http://example.org/index.html>; rel="timemap";
    type="application/json",
<https://memgator.cs.odu.edu/timegate/http://example.org/index.html>; rel="timegate",
<http://web.archive.org/web/20021016101337/http://www.example.org:80/index.html>;
    rel="first memento"; datetime="Wed, 16 Oct 2002 10:13:37 GMT",
<http://web.archive.org/web/20031207031049/http://www.example.org:80/index.html>;
    rel="memento"; datetime="Sun, 07 Dec 2003 03:10:49 GMT",
<http://archive.md/20070621200621/http://example.org/index.html>;
    rel="memento"; datetime="Thu, 21 Jun 2007 20:06:21 GMT",
<http://wayback.archive-it.org/all/20170717191800/http://www.example.org/index.html>;
    rel="memento"; datetime="Mon, 17 Jul 2017 19:18:00 GMT",
[... TRUNCATED ...]
<http://web.archive.org/web/20190731130056/http://www.example.org/index.html>;
    rel="last memento"; datetime="Wed, 31 Jul 2019 13:00:56 GMT"
```

Fig. 20. An Aggregated TimeMap from MemGator Server

TimeMaps for the same *URI-R* from all known web archives and combines all successful responses to return a more complete *TimeMap*. This is illustrated in Figure 20 where an aggregated *TimeMap* from *MemGator* includes *URI-Ms* from `web.archive.org`, `archive.md`, and `wayback.archive-it.org` web archive domains. This response was consolidated from many different *TimeMaps* fetched from individual upstream web archives that support Memento. Individual web archives may be sparse in capturing a rapidly changing resource adequately, hence missing many updates. Depending on the crawling policy, web archives may sample a small subset of the WWW. *TimeMaps*, when aggregated across various archives, give a more complete picture of the past web.

Popular Memento aggregators such as Memento Time Travel[1] and MemGator [10] may generate a lot of traffic for the upstream web archives. Aggregating responses from all known web archives is useful, but broadcasting each request to every web archive is wasteful, because for a given request very few archives return a useful response. Aggregator services often use local caches to reduce the amount of traffic they cause to upstream web archives. Caching is helpful for popular resources that are requested very often (e.g., CDN-hosted

[1] `http://timetravel.mementoweb.org/guide/api/`

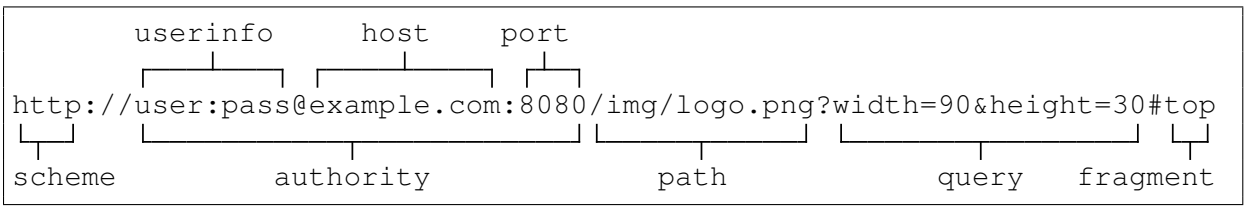


Fig. 21. A Generic URI Example

jQuery[2] that is commonly used in many web pages). However, caching failed responses (such as `404 Not Found`) or caching frequently archived resources may cause the freshness issue. Additionally, less frequently requested resources may spoil the cache (depending on the caching policy) while being ineffective in reducing upstream traffic. Our web archive profiling work can solve this issue by making a prediction of top-K potential web archives where mementos of a *URI-R* might be present to avoid aggregation request broadcasting.

2.6 URI AND URI TRANSFORMATIONS

A Uniform Resource Identifier (URI) [62] is a systematically formatted compact string that is used to identify a resource (as illustrated in Figure 21). A Uniform Resource Locator (URL) is a subset of URIs that has an additional capability of locating a resource (such as on a network). URIs are at the core of web archives in many stages such as crawling, indexing, analysis, and replay. While there are hundreds of registered URI schemes [139], web archives primarily only focus on "`http`" and "`https`" because these are the most common ones on the web.

A URI cannot identify multiple resources at the same time, but multiple URIs can identify a single resource. Moreover, a single URI can be represented in more than one ways due to the following reasons:

- `http` and `https` schemes generally point to the same resource
- Scheme and host segments are case-insensitive
- Percent-encoded hexadecimal digits are case-insensitive
- Some web services treat their path and/or query segments as case-insensitive as well

[2] `http://jquery.com/download/#using-jquery-with-a-cdn`

- `www` sub-domain is generally optional
- Generally, an `index.*` file is loaded when the path portion of the URI ends with a forward slash (i.e., a directory path) without an explicit filename
- Default ports 80 and 443 (of `http` and `https` respectively) are optional
- Paths can have single or double dots (e.g., `./..`) for relative paths and multiple consecutive forward slashes (e.g., `//...`) as path segment separator (like in Unix file systems [202])
- Individual name-value pairs of query parameters can appear in any order
- Some well-known query parameters such as `utm_*` and `jsessionid` are used for tracking or session management, not to identify a resource

2.6.1 URI NORMALIZATION/CANONICALIZATION

When an entity can have many different representations, it is sometimes desired to recognize one of those representations as the standard form. Suppose, we are given the task to find whether a given substring is present in a base string of characters irrespective of the case of letters (i.e., case-insensitive search). One possible approach to attempt this task is to search for all possible variations of the lookup substring, but the cost of such lookup will grow exponentially as the number of alphabet letters grow in the lookup substring (i.e., 2^N for a lookup string with N letters that can either be in upper or lower case). An alternate approach is to decide an application-specific standard representation, such as all lower case letters, and transform both the lookup and base strings to that form before performing the search. After such transformation, this search is performed only once, irrespective of the size of the lookup substring. This process of transformation of an entity to a standard representation is called *Normalization* or *Canonicalization* (i.e., transforming to a normal or canonical form).

Since a URI can be represented in many different ways, search engines, caching proxies, web browsers, and web archives try to normalize URIs when they are used as storage keys [54]. This way multiple variations of a URI are transformed into a unique representation. For example, while loading a web page, a web browser encounters two image elements with their source attributes set to `http://example.com:80/logo.png` and `HTTP://EXAMPLE.COM/logo.png` respectively, which are the same URI represented in two

different forms. Making two different requests for these will be a network bandwidth wastage and caching those responses under two different keys would be a storage wastage. This is why web browsers (and many other components of the web infrastructure) normalize URIs using the rules standardized under Section 6 of the URI RFC [62]. The URI normalization [62] process involves one or more of the following steps:

- Remove scheme, "`www`", port number, fragment identifier, and some well-known unnecessary query parameters
- Remove the "`?`" when the query is empty
- Remove directory index file name (e.g., `index.html`)
- Add trailing slash for directories
- Normalize path (such as removing dots and duplicate consecutive forward slashes) [202]
- Downcase hostname (path and query params are case sensitive)
- Upcase percent-encoded hexadecimal digits
- Sort name-value pairs of query params lexicographically

Some of these normalization steps (such as removing scheme, "`www`", or directory index file) change the semantics of the URI, but are commonly performed for practical reasons. Figure 22 illustrates the URI normalization process. For example, from the URI representation shown in line 4 we removed "`http://`", changed "`NEWS.BBC.CO.UK`" to lower case, removed "`:80`", collapsed duplicate forward slashes in the path (i.e., "`//images//Logo.png`"), sorted query parameters (i.e., "`height`", "`width`", and "`rotate`"), changed percent-encoded characters to upper case (i.e., "`%c2%b0`", which represents the "°" symbol), and removed the segment identifier "`#top`".

It is worth noting that the terms *Normalization* and *Canonicalization* have slightly different semantics in the context of URIs. A canonical URI can be something that we may not be able to deduce using standard URI normalization rules and may need some out of band information. For example, a company may register domain names under many different TLDs to preserve its brand (e.g., `example.com`, `example.org`, `example.us`, and `example.co.uk`) and serve the same resources on all these domains. However, it is equally possible that some of these domain names are registered by different entities. Similarly,

```
1. https://news.bbc.co.uk/images/Logo.png?width=200&height=80&rotate=90%C2%B0#top
2. http://www.news.BBC.co.uk/images/Logo.png?width=200&height=80&rotate=90%c2%b0#top
3. http://www.news.bbc.co.uk/images/Logo.png?rotate=90%c2%B0&width=200&height=80
4. http://NEWS.BBC.CO.UK:80//images//Logo.png?height=80&width=200&rotate=90%c2%b0#top
# And many other variations of the URI...
    ↓ NORMALIZATION ↓
* Remove scheme, "www", port number, fragment identifier, and some unnecessary query parameters
* Normalize path (such as removing dots and duplicate consecutive forward slashes)
* Downcase hostname (path and query params are case sensitive)
* Upcase percent-encoded hexadecimal digits
* Sort name-value pairs of query params lexicographically
=> news.bbc.co.uk/images/Logo.png?height=80&rotate=90%C2%B0&width=200
```

Fig. 22. URI Normalization Process

```
# A normalized/canonicalized URI
news.bbc.co.uk/images/Logo.png?height=80&rotate=90%C2%B0&width=200
    ↓ SURT ↓
* Downcase everything
* Reverse the hostname segments and separate them by commas
* Separate authority and path by a closing parenthesis
=> uk,co,bbc,news,)/images/logo.png?height=80&rotate=90%c2%b0&width=200
```

Fig. 23. Sort-friendly URI Reordering Transform (SURT) Process

sometimes webmasters introduce different URIs (under the same domain or a different one) for the same resource for purposes like testing, cache bursting, or tracking. Search engines do not like duplicate resources associated with many different URIs, so for the sake of search engine optimization webmasters may include a link with relation type "`canonical`" to indicate that the resource should be associated with the advertized canonical URI as the single origin of truth [198].

However, in the web archiving community and in this work the terms *Normalization* and *Canonicalization* are used interchangeably (unless specified otherwise) where the latter is more common [226]. *Canonicalization* is very important in web archiving to minimize the effort of a crawler and to improve the discovery of archived resources at the time of replay. For this reason, many web archives use more aggressive canonicalization rules than standard web browsers and proxy caches. However, canonicalization based on per-domain rules, multi-domain ownership, and URI reorganization over time is an open research area in the context of web archiving, which is out of scope for this work. Such a canonicalization

can reduce false positives and false negatives in web archive replay and discovery systems, but will require some architectural changes.

2.6.2 SORT-FRIENDLY URI REORDERING TRANSFORM (SURT)

In addition to *URI* normalization/canonicalization *SURT (Sort-friendly URI Reordering Transform)* [227] is used to place related *URIs* together when sorted. This spatial locality is important for efficient indexing of large *URI* collections. In a traditional URI the hostname parts are organized differently than paths. In the hostname section, the root of the Domain Name System (*DNS*) chain (i.e., the Top-Level Domain, or *TLD*) comes at the end towards the right hand side while registered domain name portion and subdomain sections are placed towards the left hand side. In contrast, in the path section, the root path comes first followed by deeper nodes of the path tree towards the right side. As a consequence, if a list of three domain names `example.com`, `foo.example.com`, and `example.net` are sorted, the latter with a different *TLD* will sit in between the other two. As opposed to this the *SURT* reverses the order of domain name tokens (originally, separated by dots) and uses a comma as the new delimiter. *SURTs* are commonly used in archival index files and many other places where a *URI* is used as a lookup key field, including *MementoMap* (which is an outcome of this research work).

Figure 23 illustrates how a normalized/canonicalized *URI* is converted to its corresponding *SURT*. Due to the aggressive downcasing, *SURTs* are lossy (i.e., not completely reversible), hence only suitable as lookup keys in an index. This decision was made with a practical assumption that there will be very rare occasions where a site will have files/directories at a given path depth with names that only differ in their letter casing or will have query parameter values that collide due to case-sensitively (See Appendix B for comments in the original Java implementation of *SURT*). The advantage of this decision is that mementos of `twitter.com/BarackObama` and `twitter.com/barackobama` are consolidated (which point to the same resource), but as a consequence `bit.ly/A` and `bit.ly/a` collide (which are two different resources).

We can extend *SURT* and utilize it for archive profile serialization and dissemination. Our extension includes support for wildcards inspired by "`robots.txt`" (discussed in Section 2.8.3).

2.7 ARCHIVE FILE FORMATS

Web archives usually use special file formats to store the data, collection analysis, and

index for lookup. Following are some of the commonly used file formats by web archives that are well supported in web archiving tools and can be used for archive profiling.

2.7.1 WARC

WARC (Web ARChive) file format [142, 15] is the de facto standard for web archives to store their output including DNS (Domain Name System) lookup, HTTP requests, HTTP responses (including headers and payload), and some other things. These files are the default output of the commonly used Heritrix crawler [184], but it can also be generated using WGet [220] or other tools such as WARCreate [158]. Figure 24 illustrates a sample WARC file with two related WARC records of types `request` and `response` respectively. A typical WARC file contains an arbitrary number of records which reduces the number of inodes [207] on the file system, stores inline metadata, and eliminates the possibility of name collisions in contrast to storing archival data in plain files. In the early days of web archiving the Internet Archive introduced ARC file format that formed the basis of the WARC file format [71].

WARC/ARC files can be used for archive profiling. However, it will be more efficient to use their index instead. Web Archive Collection Zipped (WACZ) is a related file format that bundles WARC files with their indexes and other metadata files [166]. It is still in the draft phase, but is a potential candidate of an archive profiling source.

2.7.2 WEB BUNDLES

Web Bundles (also known as, Web Packaging) is an emerging web standard that bundles multiple HTTP transactions (request and response pairs) in a single file for transportation [243, 251, 250, 249, 208, 164, 209]. The primary purpose of Web Bundles is to deliver them to a user-agent (such as a web browser) via a third party (such as content delivery network) and make the user-agent believe that the bundle was originally prepared and signed by the main origin. This way, web pages can be shared offline and other means such as store and forward on demand. It shares a lot in common with WARC files, but is optimized for transportation while WARC is optimized for lossless long-term preservation of resources. Unlike WARC, Web Bundles is a binary format file. Moreover, Web Bundles have a built-in index, so each bundle is self-sufficient. It is more likely that bundled resources will be closely related (such as, all the page requisites of one or more pages from the site), while WARC records are put together more arbitrarily.

This is still a work in progress, but when ready, it is expected to benefit web archives in

```
WARC/1.0
WARC-IP-Address: 93.184.216.34
WARC-Type: request
WARC-Record-ID: <urn:uuid:9b1f7afd-c251-4cb3-a7e9-3690925dc462>
WARC-Concurrent-To: <urn:uuid:57ffb7a4-996b-47e8-8078-769362cbcbd3>
WARC-Target-URI: https://example.com/hello
WARC-Date: 2019-08-08T15:47:15Z
WARC-Payload-Digest: sha1:HI7YB5JPCEVFF54CDGIDSB3IUM75WU3U
WARC-Block-Digest: sha1:OGJKUHULUFYCBFFUT5Z5AKKRWUHXRKKN
Content-Type: application/http; msgtype=request
Content-Length: 67

GET /hello HTTP/1.1
Host: example.com
User-Agent: curl/7.58.0


WARC/1.0
WARC-IP-Address: 93.184.216.34
WARC-Type: response
WARC-Record-ID: <urn:uuid:57ffb7a4-996b-47e8-8078-769362cbcbd3>
WARC-Target-URI: https://example.com/hello
WARC-Date: 2019-08-08T15:47:15Z
WARC-Payload-Digest: sha1:LDEVQLI2LNVAXZU2JUJMUXYTG3YDXASI
WARC-Block-Digest: sha1:NHNS5CS6LO7FFJKDSVFSI4WQLBBE6SSU
Content-Type: application/http; msgtype=response
Content-Length: 180

HTTP/1.1 200 OK
Date: Thu, 08 Aug 2019 15:47:15 GMT
Server: Apache
Last-Modified: Fri, 02 Aug 2019 03:35:56 GMT
Content-Type: text/plain
Content-Length: 16

Hello Archives!
```

Fig. 24. A WARC File With a Request and Corresponding Response Records

capturing and replaying in a more coherent manner [31, 211]. This affects archive profiling in a way that we might profile only the HTML pages of a website, leaving other page requisites come from the same archive from where the HTML page is coming.

2.7.3 WAT, WANE, AND WET

Web Archive Transformation (WAT), Web Archive Named Entities (WANE), and WARC Encapsulated Text (WET) files are derived formats from WARC [236, 182, 233, 85, 50]. A WARC file is usually generated at the crawl time, which is post-processed for optimization and creation of derivatives. WAT files contain JSON-formatted metadata about selected WARC records such as title and outlinks of an HTML page. WANE files contain JSON-formatted named entities (such as people, places, organizations, etc.) extracted from a

```
Envelope
  WARC-Header-Metadata
    WARC-Target-URI
    WARC-Type
    WARC-Date
    ...
  Payload-Metadata
    HTTP-Response-Metadata
      Headers
        Content-Language
        ...
      HTML-Metadata
        Head
          Title
          ...
        Links [list]
      Headers-Length
      Entity-Length
      ...
Container
  Gzip-Metadata
  Compressed?
  Offset
```

Fig. 25. WAT File Structure

WARC record. WET files contain the plain text version of a WARC record after stripping off all the markup and other structures. These derived files are helpful in tasks like understanding collections, text analysis, knowledge graph building, machine learning, fulltext search, etc.

Figure 25 illustrates the structure of WAT record. These derived records are generally placed in a WARC file using `metadata` or `conversion` *WARC-Type* records. All these derived formats contain the original URI and the datetime when the corresponding WARC record was created which can be utilized for archive profiling.

2.7.4 CDX/CDXJ

CDX (Capture inDeX) [138] is a *CSV*-like text file-based index format (as shown in Figure 26) that has traditionally been used by the IA and was one of the primarily supported index formats of OpenWayback[3]. It is very rigid in nature and has a predefined list of fields that are not extendable. Each block of WARC files is indexed as a CDX line. Each entry in the CDX file stores the canonical URI and observation time (Memento-Datetime) as lookup keys and associated data such as the status code, content-type, content digest, record offset

[3] `https://github.com/iipc/openwayback`

```
1  CDX N b a m s k r M S V g
2 com,example)/ 20140103030321 http://example.com text/html 200 B2LTWWPUOYAH7UIPQ7ZUPQ4VMBSVC36A
  ↪  - - 1043 333 example.warc.gz
3 com,example)/ 20140103030341 http://example.com warc/revisit - B2LTWWPUOYAH7UIPQ7ZUPQ4VMBSVC36A
  ↪  - - 553 1864 example.warc.gz
4 org,iana)/domains/example 20140128051539 http://www.iana.org/domains/example text/html 302
  ↪  JZ622UA23G5ZU6Y3XAKH4LINONUEICEG - - 577 2907 example.warc.gz
```

Fig. 26. Sample CDX File

```
1 com,example)/ 20140103030321 {"url": "http://example.com", "digest":
  ↪  "B2LTWWPUOYAH7UIPQ7ZUPQ4VMBSVC36A", "length": "1043", "offset": "333", "filename":
  ↪  "example.warc.gz"}
2 com,example)/ 20140103030341 {"url": "http://example.com", "mime": "warc/revisit", "digest":
  ↪  "B2LTWWPUOYAH7UIPQ7ZUPQ4VMBSVC36A", "length": "553", "offset": "1864", "filename":
  ↪  "example.warc.gz"}
3 org,iana)/domains/example 20140128051539 {"url": "http://www.iana.org/domains/example",
  ↪  "digest": "JZ622UA23G5ZU6Y3XAKH4LINONUEICEG", "length": "577", "offset": "2907",
  ↪  "filename": "example.warc.gz"}
```

Fig. 27. Sample CDXJ File

in the WARC file, content length, and the WARC file name. The latter three are generally used to locate the capture in a WARC file.

CDXJ [25] is an evolution of the classic *CDX* format. In this file format, lookup key fields (*URI-R* and *Datetime*) are placed at the beginning of each line which is followed by a single-line compact JSON [90] block that holds other fields that can vary in number and be extended as needed (as shown in Figure 27). This format is primarily used by archival replay systems including PyWB[4] and our InterPlanetary Wayback (IPWB) [152].

Both of these formats are sort-friendly to enable binary search on file when performing lookups. These CDX/CDXJ files are just indexes, hence these are significantly smaller than WARC files. Having access to an archive's CDX files gives complete knowledge of its holdings in terms of what URIs it captured, when, and how often. This information is sufficient to build a lightweight profile for the archive. Additionally, we can use a CDXJ-like format for web archive profile serialization.

[4] `https://github.com/webrecorder/pywb`

2.8 SYNDICATION AND DISCOVERY

WWW is a decentralized publishing platform where webmasters or content creators/owners can publish, update, or redact their resources independently. This decentralized nature, with all its virtues, poses the challenge of content discovery for users. Search engines try to solve the discovery problem by indexing the web at large scale and make relevant resources available via keyword searching. However, this means users needs to know in advance what they are looking for, but this may not be the case if, for example, someone is interested in the latest news. In the recent years after the rise of social media, content creators can promote their content to subscribers directly, which is somewhat equivalent to newsletter subscription of the pre-social media era. People have explored different means to make their content discoverable by search engines and users, which emerged as various standards or widely adopted practices on the web. In this section we explore some well-established protocols to identify how they can inspire web archive summarization and discovery of web archive holdings.

2.8.1 RSS/ATOM FEED

Really Simple Syndication (RSS) or Atom Feed are popular and competing standards for web feeds [213, 194]. Atom tries to improve on some limitations of RSS [214]. A web feed is a way to summarize recent changes in a web site and make it available to clients on demand. Web sites that provide an RSS/Atom feed usually add their feed URIs in their site's markup with "`alternate`" link relation type for the discovery of feed sources. Web browsers in the past used to highlight the presence of a feed link in a page in the toolbar and give users easy controls to add the link in their feed readers, but in the recent years some browsers like Google Chrome and Mozilla Firefox have removed it [201, 83]. However, it is still possible to enable feed detection and subscription using browser add-ons.

Before the popularity of *Web Push Notifications* [63], periodic polling for recent changes was a common practice for users to keep up with topics they are interested in. For example, a user can subscribe to various blogs and news sites by adding their web feed URIs in a feed reader software, these sites can add new content independently at any time and update their Atom/RSS feed to reflect new changes, user's feed reader can periodically poll all the feed documents from all the sources it tracks to identify new content (that are not present in feed reader's cache or have timestamps past the last check), and make the new content available for the user to read later. Figure 28 illustrates an Atom feed that contains metadata for

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <feed xmlns="https://www.w3.org/2005/Atom">
3    <title>An Example Feed</title>
4    <subtitle>A blog that does not exist</subtitle>
5    <link href="https://example.org/blog/atom.xml" rel="self" />
6    <link href="https://example.org/blog/" rel="alternate" type="text/html" />
7    <id>urn:uuid:7a839982-a407-45f4-bf80-be7eb9ff7da3</id>
8    <updated>2020-06-21T09:24:28Z</updated>
9    <entry>
10     <title>Lorem Ipsum</title>
11     <link href="https://example.org/2020/06/21/lorem-ipsum" />
12     <id>urn:uuid:44bd656a-5cd8-4e0c-916b-1c3ed09f1821</id>
13     <updated>2020-06-21T09:24:28Z</updated>
14     <summary>
15       Lorem ipsum dolor sit amet, consectetur adipiscing elit.
16       Integer vel ligula blandit dolor mattis varius id nec magna.
17     </summary>
18     <author>
19       <name>Jane Doe</name>
20       <email>jdoe@example.com</email>
21     </author>
22   </entry>
23   <entry>
24     <!-- Second entry -->
25   </entry>
26   <entry>
27     <!-- Third entry -->
28   </entry>
29 </feed>
```

Fig. 28. Example Atom Feed

the website (lines 3–8) and individual resources (lines 9–22). There are entries for each individual resource that the website wants to summarize in the recent changes (lines 23–28). These entries are usually present in the reverse chronological order of the resource update time with pagination support.

2.8.2 SITEMAPS

HTML is a markup language that used to generate static documents, but later many executable environments (such as Adobe Flash, Silverlight, Java Applets, and JavaScript) emerged that can be embedded in an HTML document to change its state and manipulate the markup after the web page is loaded in a capable client (such as a web browser). This posed a serious challenge to web crawlers that parse web pages to extract external links and embedded resources to be added in their frontier queue without necessarily rendering those pages. Rendering pages for crawling (for example, in a headless browser or in a virtual machine) to discover all the out-links and necessary resource URIs is difficult, costly, and sometimes impossible [70]. This affects search engines as they fail to discover and index many important pages of a website. This also affects web archives as they may fail to

archive some webpages with all their page requisites, resulting in a damaged replay [69].

To overcome this discovery problem Google introduced *Sitemaps* protocol as a *Search Engine Optimization (SEO)* technique [224, 191, 230]. Webmasters can make better decisions about which resources on their website are non-trivial and can automate the process of listing important resources to be harvested by crawlers [188]. A sitemap is an XML document, often placed at the web root of a domain as "`/sitemap.xml`", that lists links to resources that webmasters want search engines to index from their websites. Web crawlers usually check for the presence of a sitemap at this well-known location before starting to crawl a domain. This saves crawlers from doing extra work to parse each page they download to extract more links. Sitemaps can also list resources that are otherwise disconnected and not linked from other pages. In addition to listing resources, a sitemap can annotate each entry with its last modified time, relative importance value, as well as change frequency to further optimize crawler efforts. Moreover, sitemaps can have links to other sitemaps of the domain to allow large sitemaps to be split in smaller parts for pagination and easier management and update.

Figure 29 illustrates a typical sitemap where the "`url`" XML element is repeated for each entry under which only the "`loc`" element is mandatory; other fields are optional. Figure 30 illustrates an index sitemap where the "`sitemap`" XML element is repeated to link to two compressed sitemap pages.

We considered using sitemaps to summarize web archive holdings along the lines of how it formed the basis for the ResourceSync framework [82, 130, 244]. However, we realized the following drawbacks:

- It is good for listing all the resources, but not the summary of holdings at various path depths, which means archives exposing their entire index.
- It is impractical for large archives to generate sitemaps of their collections and costly for the consumers to download them and keep them synchronized.
- It is inherently verbose due to XML format.

However, we can still leverage some sitemaps concepts such as pagination in our *MementoMap* framework.

```
<?xml version="1.0" encoding="utf-8"?>
<urlset xmlns="https://www.sitemaps.org/schemas/sitemap/0.9">
  <url>
    <loc>https://example.com/</loc>
    <lastmod>2020-06-21T09:24:28+00:00</lastmod>
    <changefreq>daily</changefreq>
    <priority>0.9</priority>
  </url>
  <url>
    <loc>https://example.org/2020/06/21/lorem-ipsum</loc>
    <lastmod>2020-06-21T09:24:28+00:00</lastmod>
  </url>
</urlset>
```

Fig. 29. Example Sitemap

```
<?xml version="1.0" encoding="UTF-8"?>
<sitemapindex xmlns="https://www.sitemaps.org/schemas/sitemap/0.9">
  <sitemap>
    <loc>https://example.com/sitemap-1.xml.gz</loc>
    <lastmod>2020-06-21T09:24:34+00:00</lastmod>
  </sitemap>
  <sitemap>
    <loc>https://example.com/sitemap-2.xml.gz</loc>
    <lastmod>2020-06-21T09:24:58+00:00</lastmod>
  </sitemap>
</sitemapindex>
```

Fig. 30. Example Index Sitemap

```
User-agent: Googlebot
User-agent: bingbot
Disallow: /admin/
Disallow: /data/
Allow: /data/summary/*.html
Crawl-delay: 20

User-agent: *
Disallow: /
```

Fig. 31. Example robots.txt File

2.8.3 ROBOTS EXCLUSION PROTOCOL

Robots Exclusion Protocol (REP) (often called as "`robots.txt`") is a means to tell crawler bots which sections of a website they should not attempt to crawl [165, 253]. It is suggestive in nature and does not prevent a crawler from accessing excluded sections of the website, but

a good crawler should respect the advice of webmasters to avoid being blocked later [225]. There can be many reasons why a webmaster would want to make certain resources accessible online, but not want a search engine crawler to spend time on them, for example, large binary files, admin interfaces, infinite pagination (such as calendars), and other crawler traps [133]. Usually, web crawlers check for the presence of "`/robots.txt`" before starting to crawl a domain, which is a well-known location for it.

The format of the file is very simple. It starts with specifying one or more user-agents, followed by one or more allow/disallow path directives that will be applicable to specified user-agents above them. Both the user-agent directive and allow/disallow path directives support wildcards. Furthermore, some non-standard extensions also added additional directives to specify crawl delay and the location of the primary "`sitemap.xml`" file (if it is not present at the root of the website).

Figure 31 shows an example `robots.txt` file in which lines 1–2 suggest that "`Googlebot`" and "`bingbot`" should respect directives that follow. Lines 3–4 suggests them to not index any URIs that start with "`/admin/`" or "`/data/`", but line 5 allows them to index any HTML file that is present under "`/data/summary/`" path. Line 6 suggests them to pause for at least 20 seconds before making another request to the domain. Finally, lines 8–9 suggest every other bot to not crawl anything from the website.

Unlike a `sitemap.xml` file that always expect full URIs, `robots.txt` support path prefixes and wildcards that can be very helpful in summarizing specific sections of a website. This behavior aligns well with the objective of web archive summarization. However, due to the following reasons we cannot use `robots.txt` for archive profiling in its current form:

- It is an exclusion document, which means it suggests what not to look for rather than summarizing what is present under a domain.
- Its allow/disallow directives usually refer to paths under the same domain rather than full URIs, which in the case of web archives, there are numerous domains that they collect resources from and replay.
- It does not allow additional attributes to be associated with each specified path prefix/directive (e.g., we would want to report an estimate of number of resources that are present under a domain or a path depth).

A `sitemap.xml` file allows expressing holdings of a website, but it is comprehensive in nature. On the contrary, a `robots.txt` file allows summarized representation, but it is

not meant to express holdings. However, by combining the desired features of these two protocols we can come up with something that can allow summarizing holdings of web archives in an efficient manner.

2.8.4 WELL-KNOWN URIS

In the last two sections we mentioned that web crawlers look for special files under the root of the domain at "`/sitemap.xml`" and "`/robots.txt`". These are well-know locations for these special files that provide metadata about the origin. Over time the need for more such special metadata files emerged, but placing every special file under the root of the domain has some drawbacks. For example, such file names may collide with other special files or existing resources on certain domains and limit the control of the origin over its URI space.

To avoid polluting the web root of origins and to minimize collisions a separate namespace "`/.well-known/`" was proposed and a registry was created [193, 140]. This means all special files in the future can be namespaced under this special path. Namespace is a common means to solve such problems where many independent entities work in a shared space and may cause collisions. A similar problem was identified in GNU/Linux desktops where various applications create/expect application-specific files (such as, user configurations and caches) under the "`$HOME`" directory of the user, which results in a growing number of hidden files in users' home directory as they install more applications over time. This was solved by introducing special hidden folder like "`$HOME/.config`" and "`$HOME/.cache`", so that individual applications can create (and look for) their special files and folders under these namespaces [56]. Many new applications now utilize these namespaces while existing applications are gradually updating to support them.

We plan to register "`/.well-known/mementomap`" as a well-known URI for web archives to server their primary *MementoMap* at. This will allow automatic discovery of the summary of archive holdings by applications like Memento aggregators.

2.9 INVERTED INDEX

An *Inverted Index* is a simple data structure that is commonly used in fulltext searching [254]. In a simple term, if a document is considered to be a collection of words, then an *Inverted Index* lists documents containing each word. Depending on the use case, an *Inverted Index* may also contain frequency, relevance score, or other weight and indicators of each word in each document.

```
1 D1: W1 W1 W1 W4 W4
2 D2: W1 W3 W3 W3 W3
3 D3: W1 W1 W1 W1 W2 W2 W4 W4 W4
4 D4: W4 W4 W4 W4 W4 W4
```

Fig. 32. A Sample Document Index (where each row represents a separate document)

```
1 D1: [(W1, 3), (W4, 2)]
2 D2: [(W3, 4), (W2, 1)]
3 D3: [(W1, 4), (W4, 3), (W2, 2)]
4 D4: [(W4, 6)]
```

Fig. 33. A Sample Document Index

Suppose we have a collection of documents ($\{D_1, D_2, D_3, ...\}$) and each document is made of a set of words ($\{W_1, W_2, W_3, ...\}$) where each word may appear multiple times in the same document as shown in Figure 32. Suppose we want to answer questions like, "what are the top-k words in a given document of the collection?" (i.e., most relevant words to a given document), without the need of scanning each document each time. This is a common question for classification tasks. For this purpose we can create an index. An index in this case can have a key field containing the name or the identifier of each document in the collection and the value can have a sorted list of tuples containing words and their corresponding term frequencies. To make the lookup in the index fast, the index is sorted by its key field as shown in Figure 33.

Now, suppose we want to answer questions like, "what are the top-k documents of the collection that contain a given word?" (i.e., most relevant documents to a given word), without the need of scanning each document each time. This is a common question for fulltext search tasks. For this purpose we can create an inverted index. As opposed to the document index described above, words are being used as keys in an inverted index and the documents those words appear in become part of the value field. To make the lookup in the inverted index fast, the index is sorted by its key field (i.e., words) as shown in Figure 34. Inverted indexes have traditionally been used in the *Information Retrieval* discipline. In situations where case-sensitive search is not needed, words are canonicalized (e.g., changing

```
1 W1: [(D3, 4), (D1, 3)]
2 W2: [(D3, 2), (D2, 1)]
3 W3: [(D2, 4)]
4 W4: [(D4, 6), (D3, 3), (D1, 2)]
```

Fig. 34. A Sample Inverted Index

the word to all lowercase letters in languages where applicable) before they are added to the inverted index. To associate variations of the same root word, stemming is performed on words [173] before adding them to the inverted index so that a lookup for "book" also matches words like "books", "booked", "booking", etc.

A regular document index grows linearly as more documents are added to the collection. This means for each new document added in the collection, there is one added entry or row in the index. On the contrary, the growth of an inverted index follows Heaps' Law, which initially grows rapidly, but the growth slows down over time as each new document does not introduce as many new words. The index is capped by the maximum size of the vocabulary (or stems) of the language. However, the value field of the inverted index becomes denser as more documents are added to the collection.

We can use an *Inverted Index* of archive profiles for Memento routing. Here, we consider an archive profile serialized as a *MementoMap* (see Chapter 4 for details) to be a document that describes holdings of the corresponding web archive and lookup *SURT* keys in the *MementoMap* as words. While the number of profiles is small, it would be easy to perform lookup in each profile individually, but as the number of archives (and corresponding *MementoMaps*) grow, it would become an impractical approach. Creating an inverted index of *MementoMaps* can solve the scaling problem of Memento aggregators.

2.10 CHAPTER SUMMARY

In this chapter we briefly described various terminologies and technologies that will be helpful in understanding problems this work addresses and the solutions we propose. We first described *HTTP* and its components, such as HTTP methods and status codes, *Soft-404*, and access logs. After that we discussed web archiving, web archives, and collection policies. Then we discussed *Memento* and related terminologies such as *TimeGate* and *TimeMap* along with a brief discussion on HTTP status codes in the context of web archives. Then we

talked about *Memento Aggregators*. Furthermore, we described *URIs* and transformations of *URIs* such as normalization/canonicalization and *SURT* that are necessary for indexing. After that we discussed various web archiving related file formats including: *WARC*, *WAT*, *WANE*, *WET*, *CDX*, and *CDXJ*. Then we described various protocols used for syndication and discovery including: *RSS/Atom Feed*, *Sitemap*, *REP* (or `robots.txt`), and *Well-Known URIs*. Finally, we described *Inverted Index* which is primarily used in fulltext searching. We included necessary illustrations for each terminology and established the purpose, relevance, or usefulness of each in the context of this work.

CHAPTER 3

RELATED WORK

Archive profiling and Memento routing are niche fields that are not explored by many researchers beyond a small community. In this chapter we describe work published by us and a small number of researchers in the web archiving community. We also describe some closely related works from the information retrieval community. Some of the relevant fields of research to this work include: hidden/deep web, query routing, and profiling web archives.

3.1 SURFACE WEB CRAWLING

Estimating the size of the *Surface Web* that is publicly accessible, interlinked, and easily indexable by web crawlers has been of great interest both for researchers and web crawling practitioners. However, it is a difficult problem to estimate it accurately, which results in different numbers reported by different researchers.

The `WorldWideWebSize.com` is a well-known site that is making a longitudinal data available about the size of search indexes of various independent large-scale web search engines [92, 247]. They select an unrelated pair of search keywords and use them to perform a lookup in each search engine. They extract the number of matching documents for the search terms as reported on the result page of each search engine. They use these numbers to estimate the size of the index of one search engine relative to the other. Their data suggests that for the past many years the average size of Google search index (the most dominating one) was steady between 45 and 50 billion pages. At the start of the year 2019 it jumped up and averaged between 55 and 60 billion pages. Empirical evidence such as, a growing number of open-source software documentation, wiki pages, online news, videos, social media, etc. suggest that both private and public web is growing in size over time, even after we subtract the number of pages that disappear from the web (i.e., there are probably more pages added to the web than removed from it). However, their numbers do not indicate any linear or exponential growth patterns. We can think of two possible reasons behind it: 1) number of matching documents reported by search engines are not updated frequently, and/or 2) search engines limit their index size to something that is sufficient for users' search needs, leaving any low quality pages out of the index. This means their

estimates reflect the size of search engine indexes, which might not be an indicator of the size of the public web.

Lawrence and Giles estimated the size of the indexable web to be at least 320 million pages in 1997 [169]. They estimated index sizes of six different search engines and analyzed pairs of search engines to estimate the size of the indexable web. They reported that no single search engine has indexed more than one third of their estimated indexable web. In the same year Bharat and Broder estimated the lower bound of static pages on the public web to be 200 million [64]. Later in 2004, Dobra and Fienberg estimated the site of the web with a lower bound of 520 million [94]. They argued that the size of the web was at least twice as big in 1997 as it was reported by the above two papers.

More recently, in 2012, Alarifi et al. estimated over 2 billion Arabic web pages indexed in search engines [34]. They used more recent search engines (like Google, Yahoo, and Bing) that operate on a much larger scale than those used in the past researches (like Alta Vista, Excite, HotBot, and Lycos), many of which are not operational anymore. It is worth nothing that their estimate is only about Arabic web pages, the overall indexed web is arguably much larger than 2 billion pages.

In 2014, Khabsa and Giles estimated the number of English scholarly documents accessible on the web over 140 million [160]. They used Google Scholar and Microsoft Academic Search for estimation and reported that about 100 million are indexed in Google Scholar. They further reported that about one quarter of these indexed scholarly documents are freely accessible. Furthermore, they estimated the distribution of these documents in 15 different research fields and found that some fields have more freely available documents (as big as 50%) than others (as low as 12%). There is a growing interest in bringing the two disciplines of web archiving and scholarly communications closer to work as complements to each other. These efforts are done from both ways, web archives trying to index and preserve publicly accessible scholarly documents [190] and digital libraries enriching their collections with scholarly documents found in public web archives [48, 200].

Size of the indexable web is of our interest as it gives an upper bound on how much of the web can be archived, if we exclude personal web archiving behind session walls. These research works on estimating the size of the web give the motivation to aggregate mementos from many web archives as no single web archive is big enough to capture the scale of the ever-growing web.

3.2 DEEP/HIDDEN/DARK WEB CRAWLING

The *Surface Web* is the portion of public WWW that is interconnected with links as a graph [67] and can easily be crawled and indexed by search engine crawlers. In contrast, the *Deep/Hidden Web* is the part of WWW that is kept behind paywalls, login walls, or accessible by filling out a search form. Moreover, the *Dark Web* is portion of the web that is intentionally kept inaccessible by standard web browsers such as the content in the Tor network [116]. However, some of these terminologies are often used interchangeably [228].

Raghavan and Garcia-Molina developed a hidden Web crawler called the Hidden Web Exposer (HiWE) and described its architecture [203]. The system was built to explore electronic datasets and web resources that are only accessible via search forms or after authorization/login.

Ntoulas et al. built a framework of query generation to effectively discover resources from the hidden/deep web [195]. Unlike the surface web, hidden web is usually explored by filling out HTML search forms with one or more fields instead of following hyperlinks. For efficient discovery of resources it is critical to identify suitable values for each field and their suitable combinations otherwise the search might not return any useful results. Ntoulas et al. reported that one of their experiments discovered more than 90% of resources from a large web site using less than 100 queries using one of their policies.

Wu et al. explored the query selection problem for discovering hidden web datasets [248]. The issue is not just to form queries that are likely to return good results, but also target to get more unique results that were not discovered in any previous queries. To optimize the query plan they converted hidden dataset discovery into a graph problem known as *Minimum Weighted Dominating Set* [222] by modeling structured datasets as nodes connected with relational links. This is a known *NP-Complete* problem [121] for which they proposed a heuristically optimized approach and evaluated against real web datasets.

Sheng et al. proposed algorithms to find dataset tuples from the hidden web and established theoretical upper and lower bounds for their proposed algorithms [223]. They stated that their suggested algorithms are asymptotically optimal. To establish their claim they evaluated their proposed techniques against real datasets.

Hernández et al. surveyed deep web crawling [132]. They discussed various aspects of deep web crawling, such as discovery of sites with deep web resources, form filling, crawling path learning, and evaluation datasets. They concluded that the field of crawler evaluation needs more research and standardized datasets/metrics. Moreover, they suggested that future deep web crawlers should account for ever-evolving web technologies.

While some small web archives allow browsing their collections, others mostly rely on a *URI* lookup or fulltext search for the discovery of their holdings. Additionally, archived contents span over a long period of time, which causes a disconnect between old and contemporary pages. As a result, hyperlink based shallow crawling might only discover a temporal sub-graph of the holdings. This means archived content is mostly in the *Deep Web* and cannot be indexed using *Surface Web* crawlers for profiling an archive by a third party. Techniques used in hidden web crawling can be leveraged to query web archives using search keywords when fulltext searching is available, otherwise using sample lookup URIs as search keywords. Our Random Searcher Model (RSM), described in Section 6.4.1 of Chapter 6, is closely related to prior work in the field of Deep Web crawling. In RSM we query a web archive that supports fulltext searching with some seed keywords, collect links from returned results as part of that archive's holdings sample, fetch a page using one of the returned links, extract keywords from that page, and use these keywords to perform more searches until the URI sampling need is fulfilled.

3.2.1 DESCRIBING TEXTUAL DATABASES

As described above, many textual databases on the web are part of the hidden web. One approach to understand and describe a textual database relies on a fulltext search sample. In this approach a handful of fulltext search queries are sent to the database and returned responses are analyzed to both describe the dataset as well as finding more query terms suitable for further exploration. The process initially learns about the database rapidly, but the learning rate slows down as it starts to see more results that have already been seen in previous queries. The learning rate often follows Heaps' law [97] and asymptotically reaches the complete knowledge.

Callan and Connell sampled text databases using fulltext queries to generate description of databases [77]. They found that the resource description created by their sampling-based technique was sufficiently similar to the one created with complete knowledge. The sampling-based approach works independent of whether a database is willing to cooperate and is requires less resources. Apart from using a query-based sampling technique, our work is closely related to this in another aspect that we are not interested in the actual resources held by web archives, but establishing only a way to summarize and describe their holdings.

Agichtein et al. discuss the problem of discovering textual databases that are not crawlable, but expose a fulltext search interface [2, 1]. In this process, generally the crawler is initialized with a small set of query terms that are used to perform fulltext search in the

database. From there, new search terms are discovered from the returned results to perform further queries to discover more resources. If the system fails to produce any new terms, then the search comes to a halt, irrespective of whether all the documents are discovered. Agichtein et al. introduced a reachability metic to assess the completeness of the discovery according to the need of an application.

These works on modeling textual databases inspire our fulltext search based archived content discovery work [30]. After collecting a significant sample of holdings of a web archive, we create archive profiles that act as a description of the archives and can be used for Memento routing.

3.2.2 SEARCH FORM DETECTION

Deep web databases generally make their data searchable via an HTML search form. However, HTML forms can be of many other types and purposes such as login forms, comment posts, subscription, shopping carts, etc. Identifying an HTML form as a search form usually requires textual and structural analysis of the form elements and their surroundings.

Cope et al. described a technique for distributed search application to detect HTML search forms automatically [88]. To classify whether a form is a search form, they used certain HTML attributes like *name*, *value*, and *type* of input elements, *name* of the *form* element itself, and tokens present in the *action* attribute of the form (that sets the target URI where the form is to be submitted). Using decision trees on these features they were able to achieve about 85% accuracy.

Barbosa and Freire proposed a hierarchical form classifier architecture in which they leverage both textual and structural features of form elements and the surroundings of input elements [55]. In the first level of classification they identify generic forms using structural attributes and then in the second layer they identify a domain-specific form using the textual content of the form. They reported that their modular approach resulted in high accuracy, precision, and recall.

Khare et al. wrote a survey paper on automatic detection of web search forms [161]. They described various research papers and approaches of search form detection and concluded that a significant progress has been made in data extraction from deep web sources in about a decade. They also noted that over time a shift from the rule-based form detection to the model-based search interface identification is visible.

In our work, being able to identify a URI lookup or fulltext search form on web archives in many different languages automatically can be helpful in profiling web archives. Currently,

there are only a limited number of web archives with search interfaces, so automating their discovery is not worth the trouble, but it could be done if/when we need to. However, this may not be a necessity if majority of web archives were to expose their indexes via a uniform standard API [127].

3.3 FOCUSED CRAWLING

Unlike traditional general purpose search engines, a focused crawler identifies resources that belong to a specific topic and downloads them selectively. This practice optimizes resource and network usage while discovering documents related to a specific topic.

Micarelli and Gasparetti presented an overview of focused crawling [183]. They described systems that are adaptive in nature and can change their behavior during the search process according to the environment and given input parameters.

Bergmark et al. described that web crawling was traditionally been used for indexing the web for searching, but later digital libraries recognized its potential for collection building [60]. Instead of using an open-ended crawl, a focused crawling on a specific topic can be useful in keeping the collection in scope. They utilized a combination of focused crawling and tunneling for collection building, where tunneling is a technique to estimate the value of a link in addition to its relevance score before adding it to the frontier queue of the crawler.

Li et al. combined the properties of deep web crawlers and focused crawlers [171]. They proposed a multi-layered intelligent crawler caller *iCrawler* that classifies pages that potentially belong to the search domain, links that point to such pages, and HTML forms that search in a relevant database. A link is added to the frontier queue only when it surpasses certain threshold of relevance and a search form is filled in with relevant values when it is identified as a potential candidate for deep web search in a relevant database. They reported that their harvest rate and coverage rate was better than existing deep web form-based focused crawlers.

Focused crawling is relevant to our work in the limited scope of situations when a third party is profiling an archive and the web archive returns some resources that are not part of its archival collections. For example, the Internet Archive has collections of games, music, television news, books, and many other types are artifacts that are not served via their Wayback Machine. Profiling these collections can be useful in some applications, but mixing them with mementos has limited value for memento aggregators. Generally, web archives serve their mementos under a separate URI scope, so filtering non-memento resources is easy and it does not require looking into the content itself. However, if for some web archives this

is not the case then using focused crawling techniques can be helpful in identifying resources that are in scope from those that should be excluded from their profiles.

3.4 ON-PREMISE INDEXING

When dealing with a limited number of servers, it is possible to perform on-premise indexing and summarization of the holdings of a database and transfer only the summary to interested parties. In this approach, code for processing data is sent and executed where data resides instead of bringing data to the code for processing. However, this approach would require cooperation and willingness of the database owners to run code shipped by a search engine on their infrastructure, which is not possible without sufficient review and scrutiny, close partnership, and convincing advantages to the database owners. As a result, this is not a scalable approach on the independently running web, but may prove helpful for a handful of large databases.

Hammer and Fiedler proposed an approach to index hidden web by sending the crawler to the database and running it in close proximity to where the data lives [128] This greatly reduces the network cost as only finalized indexes are sent back to the search engine instead of downloading actual documents, indexing them, and throwing them away. Kumar and Bhatia also tried a similar approach to move indexing process to where the data resides and sending back indexes to the search engine [168].

In the case of web archive profiling it is possible and more practical to ask a finite number of large web archives to run profiling code on their servers while relatively smaller archives can be profiled remotely. During our work we have experienced both logistical and technical challenges in transferring large amounts of index files or running fulltext queries over large collections to create a representative sample of holdings. Being able to run profiling code on the servers of the cooperating web archives and only transferring the summary would have been much easier.

3.5 QUERY ROUTING

Query routing is the task of identifying suitable sources of information from a larger set of sources for a given query. Suppose there are a large number of libraries that collect books on specific topics and there is a meta-search engine that indexes them all based on their collection topics. When a user performs a lookup, the search engine classifies the query and identifies the topic(s) the query may belong to, then it routes the search query to a subset of libraries that collect books on those topics. It avoids broadcasting, saves resources, and

makes the lookup efficient. Query routing is a rigorously researched topic in various fields including networked databases, meta searching, and search aggregation.

Gravano et al. described protocol and system called STARTS [122]. This was a project steered by Stanford Digital Library in association with about a dozen organizations and companies. The aim of this system was to enable searching across multiple document sources with different interfaces and query models. The system was proposed at time when many organizations used external search engines to get their data indexed (i.e., they outsourced the search capability). These external search engines had different query models, incompatible to each other, and did not expose adequate metadata. This means merging search results from multiple document sources was difficult. The STARTS aimed to tackle this issue of allowing search across organizations and document sources on the Internet.

Liu describes a query routing system to better handle multiple relevant results for a given keyword query [172]. They system builds profiles of data sources as well as user queries. By combining these two independent profiles a better relevance can be achieved. They reported that their system performs a more fine-grained user interest matching than those query routing systems that only rely on keyword search.

Callan et al. describe an approach of query-based language model creation for query routing [78]. Traditional systems of query routing for textual databases required that each database provides its language model that can be used to determine appropriate databases for a given query. However, they found that such cooperation is not needed because their query sampling-based system can construct language models for each database. They reported that running about one hundred queries and retrieving a few hundred documents is sufficient to create a reasonably accurate language model of a textual database for query routing.

Jie Lu and Jamie Callan described a federated search query routing systems in hierarchical hybrid peer-to-peer networks [174, 175]. In this system there are some directory nodes that construct content models of neighboring nodes for query routing while leaf nodes perform content-based retrieval of relevant document. While such a hierarchical system is out of scope of this work, we think it will be a good future work to combine our work with this federated routing when the number of public web archive grow beyond a limit. We envision different regional or special-purpose Memento aggregators (e.g., an aggregator for european web archives and one run by Webrecorder/Conifer for its subscription-based archives) that can be queried individually or be aggregated as needed while each aggregator is responsible of understanding the holdings of only a small subset of web archives.

Sugiura and Etzioni described the architecture of an automatic query routing system called *Q-Pilot* [237]. They built topic models of 144 specialized search engines to dynamically identify the best subset of candidate search engines for given search queries. They reported a query category identification accuracy of 70% and about 40% of the time the best search engine was one of the top three from the result set.

Tran and Zhang described a query routing system from structured and linked datasets [239]. Their system returns top-k potential data sources or combinations for a given query keyword based on a multi-level scoring mechanism. They performed experiments on 150 public databases on the web and reported a P@1 of 0.92 and mean reciprocal rank of 0.89.

Meng et al. surveyed meta searching techniques [181]. Unlike query routing systems a metasearch engine, in addition to identifying relevant search engines, retrieves results from identified subset of relevant search engines, combines results, and ranks the aggregated result before returning it to the user. They compared and contrasted various approaches and algorithms used to solve these underlying aspects of metasearch engines.

Klusch et al. briefly reviewed semantic web service search [163]. In this paper they looked for state or the art systems for identifying and composing relevant web services via formal ontology-based representations such as Web Service Description Language, Web Application Description Language, or REST APIs.

Greengrass extensively surveyed information retrieval [124]. They survey covers a great deal of techniques and systems related to classification and query routing among many other IR tasks.

Query routing would be a critical component of a Memento aggregator that aggregates a large set of web archives. A Memento aggregator needs to identify a subset of candidate web archives that are likely to return good results for a given lookup URI. However, query routing has not been explored in the context of Memento routing extensively. In this work we build a high-level understanding of holdings of various web archives and route URI lookup requests at a Memento aggregator to a subset of web archives that are likely to return any mementos for the given lookup URI.

In our work we used two terms "*query*" and "*lookup*" with different semantics. A query to a web archive refers to fulltext (or partial URI) searching that may result in references to multiple (i.e., zero or more) resources ranked by their relevance to the query terms. In the case of a lookup, the resource is predetermined and we are only checking whether it is present. The response to a lookup, if successful, may return multiple versions of the same resource, but not multiple resources. Results of lookups are binary (yes/no) in nature and

have no ranking or concepts of relevance. It is worth noting that Memento routing is a lookup routing where the lookup URI acts as the identifier of the resources in every archive.

In traditional information retrieval systems it is easy to route queries when given query terms/phrases have enough signals to identify their membership to certain topics or collections. However, in URI lookup routing given URIs can be opaque, resulting in lack of signals for classification. For example, the lookup URI `https://cdc.gov/coronavirus/2019-ncov/` has sufficient tokens to identify that it may be present in *COVID19*-related web archival collections, but `https://youtube.com/watch?v=QNo5ZDvKuHg` does not[1]. This is why in this work we only rely on the structural features of a URI, not the natural language semantics, both for profiling and lookup routing.

3.6 BLOOM FILTERS

A Bloom filter is randomized space-efficient probabilistic membership set data structure. When performing a lookup Bloom filters allow false positives, but no false negatives. Bloom filters are useful in situations where an attempt to access a record is costly, especially, when often the record may not be present in the system. In such situations Bloom filters allow a quick and efficient way of knowing the absence of the record in a given system and short-circuit the lookup. However, they report false positives with some probability, in which case the lookup will be performed that will eventually fail to retrieve the record.

In the Bloom filter data structure a bit-array of size m is used that is initialized with every bit set to "0". As items are indexed one by one, they go through k number of different hash functions that each yield a number up to m (the size of the bit-array) and bits of corresponding positions are set to "1", if not already. During lookup, the key goes through the same k hash functions and if all the returned positions are already set to "1" in the bit-array, the item is predicted to be present. However, it is possible that all the positions corresponding to the lookup key were set to "1" by some other existing items even if the lookup item itself is not present, which causes a false positive result. The probability p of such collisions causing false positives increases as the number of indexed items n grows with the finite size of the bit-array.

Bloom proposed the idea of hash coding with allowable errors in 1970, which was later named after him as Bloom filters [65]. Bloom filters are being used in many applications, such as network routing, database lookup, search query routing, spell checking, cache lookup,

[1] `https://youtube.com/watch?v=QNo5ZDvKuHg` points to the official "CDC Briefing Room: COVID-19 Update and Risks" video.

and peer-to-peer networks.

Majkowski explored Bloom filters while analyzing IP spoofing in Cloudflare networks [177]. He wanted to quickly identify whether the source IP address of a packet arriving to a given data center belongs to the corresponding geo-locations. The article describes various issues in using Bloom filters for the purpose and introduces "mmuniq-hash" [176], which helped address the problem more efficiently while reducing the probability of false positives.

Broder and Mitzenmacher surveyed many different network applications of Bloom filters [68]. First, they described Bloom filters in detail and briefly mentioned their usage in application other than networks, such as databases and spell checking. After that, they covered many different network applications such as resources routing, peer-to-peer networks, caching, CDNs, etc. and how Bloom filters we modified to fit certain application needs.

While Bloom filters cannot describe holdings of a web archive, they do look promising for Memento routing, but they have some drawbacks that limits their usefulness. A Bloom filter is an in-memory data structure, which is costly to scale. While adding an item in the filter is easy, deleting them is not possible because some bits of the bit-array corresponding to an deletion-candidate item may belong to some other items still in the system. One might think that adding a secondary Bloom filter for deleted items can solve the issue, but a collision in that filter may yield false negatives, which defeats the purpose. Additionally, adding a deleted item back will pose the same challenge which introduced the secondary filter. Another drawback of Bloom filters is the inability to scale the bit-array later when collision probability p increases, as it would require rebuilding the filter from scratch and indexing all the items again. Our *MementoMap* framework achieves the benefits of Bloom filters while avoiding most of their drawbacks.

3.7 ARCHIVAL COVERAGE OF THE WEB

Ainsworth et al. attempted to answer the question, "How much of the web is archived?", in 2011 [3]. Their results showed the answer to this question depends on how we sample the web. This means the accuracy of the answer will depend on how well a sample represents the web. They sampled URIs from DMOZ (a curated directory of web pages), Delicious (a social bookmarking service), Bitly (a URL shortner server), and search engine indexes and queried them against web archives to see how many copies each URI in their samples has. They found that the variance in resulting numbers for each sample was too large to generalize the outcome. For example, they found that the number of URIs that had at least one archived copy in any of the web archive were as low as 35% in one sample and as high as

90% in another sample. They reported that about 15% to 31% (depending on the sample) URIs are archived at least once per month.

Alkwai revisited the archival rate question in 2015, but for web pages of specific languages [36, 35]. They collected over 15,000 URI samples from English, Arabic, Danish, and Korean languages to find out how much of the pages from each of these languages are archived. They found that 72%, 53%, 36%, and 33% of their sampled URIs were archived in these languages respectively. They also noted that if a URI is present in the DMOZ collection, it has a higher chance of getting archived. Furthermore, they identified that very few of the sampled Arabic URIs had an Arabic country TLD (e.g., "`.sa`" for Saudi Arabia) and were hosted in an Arabic country. However, this was not the case with Danish and Korean languages.

The GDELT Project, a platform that monitors the world's news media, reported in 2015 that around 2% of the news articles disappear in a couple of weeks and up to 14% in a couple of months [238]. Similarly, SalahEldeen and Nelson reported that about 11% of the resources shared on social media during the 2011 Egyptian Revolution were lost after a year [215]. This is an alarming rate with which resources on the web disappear. Leetaru investigated how much of the web is being archived by the Wayback Machine of the Internet Archive [170]. He looked at the holdings of the Wayback Machine from many different angles and reported various statistics. He expressed the need for more documentation and transparency on policies and algorithms that control what URIs will be archived. He recommended that web archives must adopt acquisition and collection decisions based on the community engagement the way libraries do. Moreover, he noted that we have limited understanding of what is inside of massive web archival datasets, which is one of the core motivations of our work towards web archive profiling.

Hallak estimated recently that almost two thirds of the web traffic is not publicly archivable because it goes to sites that are behind session walls or paywalls, to which some social media sites are big contributors of [126]. Kelly et al. developed a framework to archive the private web and integrate it with the public web to fill some of these cavities [155, 159].

These works identify archival voids as they show some biases in web archiving as well as quantify the small portion of the web many archives hold. These works further motivate us for Memento aggregation.

3.8 WEB ARCHIVE SEARCHING

Web archives are often large collections of historical versions of web pages from numerous

domains where each original URI may have one or more temporal copies. While researchers and archive exploration tools may interact with an archive using APIs or datasets directly, regular users interact with web archives using one of the three interfaces: 1) browsing, 2) fulltext searching, and 3) URI lookup. Browsing works well for small curated collections, but it becomes impractical for large and unstructured collections. Fulltext searching requires indexing existing and ever-growing collections, which is a costly and resource-intensive task. Consequently, large web archives (e.g., the Internet Archive) struggled to make their entire collection searchable. Due to limited fulltext search support in web archives we do not have a meta search engine that can perform keyword searching across web archives. URI lookup is the only approach that works on all web archives, which enables utilities like Memento aggregators. However, the downside of URI lookup is that the user needs to know the exact URI of the desired resource in advance.

Gomes et al. described their fulltext search architecture, challenges, and lessons learned in the process of making the Portuguese Web Archive (PWA) searchable [117]. At the time of their publication in 2013, PWA was the largest public web archive with fulltext search support over 1.2 billion resources. In a related study, Costa et al. surveyed various fulltext architectures and efforts on making web archives searchable [89]. They assessed aspects of scalability, reliability, time-awareness, an performance of Wayback Machine, PWA, and Everlast. Time-awareness is an aspect that is unique to indexing web archive or other temporal collections, which poses unique challenges not only in indexing and ranking, but also in providing an intuitive query interface and a meaningful representation of results.

Kanhabua et al. proposed a clever approach of searching IA without indexing it [150]. They leverage the index of an existing general purpose web search engine (in their example they used Bing) to indirectly search IA. When user makes a query, they retrieve a ranked list of relevant URIs from Bing, then use these URIs to perform lookup in IA to see which of these URIs have any mementos. One downside of their approach is the lack of ability to surface historical resources that are not live on the web, hence recency-focused search engines may not retain such references in their index.

3.9 ARCHIVE PROFILING

Memento query routing was earlier explored in the two content-based archive profiling efforts described below in Sections 3.9.1 and 3.9.2, but they explored extreme cases of profiling. In an earlier study [28], we found that an intermediate approach that gives flexibility with regards to balancing accuracy and effort can result in better and more effective

Memento routing.

Memento query routing was also explored in an aggregator's usage-based archive profiling effort. We found that traffic from *MemGator* (our Memento aggregator service) requested less than 0.003% of the archived resources in *Arquivo.pt*. There is a need for content-based archive profiling which can express what is present in archives, irrespective of whether or not it is being looked for.

3.9.1 URI-R PROFILING

Sanderson et al. created comprehensive content-based profiles [217, 216] of various *International Internet Preservation Consortium (IIPC)*[2] member archives by collecting their *CDX* files and extracting *URI-Rs* from them. This approach gave them complete knowledge of the holdings in each participating archive, hence they can route queries precisely to archives that have any mementos for the given *URI-R*. This approach yielded no false positives or false negatives (i.e., 100% *Accuracy*) while the *CDX* files were fresh. However, these collected *CDX* files would go stale very quickly because many web archives keep crawling the web constantly and add more mementos to their collections regularly after indexing batches of crawled data. In addition to that, on-demand web archives such as Save Page Now (a service from the Internet Archive to submit URIs for immediate archiving) [120] add hundreds of mementos to their collections every second [149] and make them available almost immediately. It is a resource and time intensive task to generate such profiles and some archives may be unwilling or unable to provide their *CDX* files. Such profiles are so large in size (typically, a few billion *URI-R* keys) that they require special infrastructure to support fast lookup. Acquiring fresh *CDX* files from various archives and updating these profiles regularly is not easy.

3.9.2 TLD PROFILING

In contrast, AlSum et al. explored a minimal form of archive profiling using only the *TLDs* and *Content-Language* [39, 40]. They created profiles of 15 public archives using access logs of those archives (if available) and fulltext search queries. They found that by sending requests to only the top three archives matching the criteria for the lookup URI based on their profile, they can discover about 96% of *TimeMaps*. When they excluded IA from the list and performed the same experiment on the remaining archives, they were

[2] `https://netpreserve.org/`

able to discover about 65% of *TimeMaps* using the remaining top three archives. Excluding IA was an important aspect of evaluation as its dominance can cause bias in results. This exclusion experiment also showed the importance of smaller archives and the impact of aggregating their holdings. This minimal approach had many false positives, but no false negatives.

3.9.3 RESPONSE CACHE PROFILING

Later, Bornand et al. implemented a different approach for Memento routing by building binary classifiers from LANL's Time Travel aggregator cache data [66]. They analyzed responses from various archives in the aggregator's cache over a period of time to learn about the holdings of different archives. They reported a 77% reduction in the number of requests and a 42% reduction in response time while maintaining 85% *Recall*.

Klein et al. revisited the performance of the above binary classifier-based approach after running the service for over a couple of years [162]. They reported an average *Recall* of about 0.73 (i.e., about 12% reduction from the originally reported results) which means the classifier misses more than one quarter of resources that are present in a given archive. They plotted absolute numbers of false positives in 13 archives and also provided the total number of entries (about 2.6 million) from the classifier log that they evaluated. However, it is not clear if each log entry corresponds to one archive or has a combined entry for all the archives. That is why it is difficult to assess the accuracy of the classifier. They also reported that a more frequent retraining of the models can improve the prediction accuracy.

These approaches can be categorized as usage-based profiling in which access logs or caches are used to observe what people were looking for in archives and which of those lookups had a hit or miss in the past. While usage-based profiling can be useful for Memento lookup routing, it may not give the complete picture of archives' holdings, producing both false negatives and false positives because the results are sensitive to the queries used to train the model[3]. In Section 8.5.1 of Chapter 8 we will discuss that our MemGator logs only reveal a tiny portion of an archive's holdings. In Figure 68 of Chapter 8 on Page 151 we illustrate that most of the frequently archived resources are never accessed (a blind spot for usage-based profiles) and most of the frequently requested resources are never archived (a blind spot for content-based profiles).

[3] `https://groups.google.com/forum/#!topic/memento-dev/YE4rt6L5ICg`

3.9.4 URI-KEY PROFILING

In previous work [28, 29] (described in detail in Chapter 6), we explored the middle ground where archive profiles are neither as minimal as storing just the *TLD* (which results in many false positives) nor as detailed as collecting every URI-R present in every archive (which goes stale very quickly and is difficult to maintain). We first defined various profiling policies, summarized *CDX* files according to those policies, evaluated associated costs and benefits, and prepared gold standard datasets [28, 29]. In our experiments, we correctly identified about 78% of the URIs that were or were not present in the archive with less than 1% relative cost as compared to the complete knowledge profile and identified 94% URIs with less than 10% relative cost without any false negatives. Based on the archive profiling framework we established, we further investigated the possibility of content-based profiling by issuing fulltext search queries (when available) and observing returned results [30] if access to the *CDX* data is not possible. We were able to make routing decisions of 80% of the requests correctly while maintaining about 90% *Recall* by discovering only 10% of the archive holdings and generating a profile that costs less than 1% of the complete knowledge profile.

3.10 CHAPTER SUMMARY

In this chapter we reviewed scholarly literature on various related topics. We first described *Surface Web*, *Deep/Hidden Web*, and *Dark Web* and discussed work done to index them. We then reviewed various research papers on textual database summarization and search form detection. Moreover, we reviewed various scholarly works on focused crawling and on-premise indexing. We also described how these are related to web archives and archive profiling. Furthermore, we discussed some work done in query routing and its relevance with Memento routing. After that we discussed Bloom filters, their relevance to our work, and their shortcomings. Then we reviewed some work that attempt to estimate the size of search engine indexes and the size of the indexable web. Furthermore, we reviewed works that explore the archival coverage of the web, the web that cannot be archived by public web archives, and potential approaches to integrate private web archiving with public archives. We then reviewed some work on making large-scale web archive collections searchable. Then we discussed initial efforts on Memento routing via both content-based and usage-based profiles. Finally, we discussed our own preliminary work on archive profiling to complete the context that will be further explored in the later chapters in a more detailed manner.

CHAPTER 4

MEMENTOMAP FRAMEWORK

In this chapter we describe the *MementoMap* framework for web archive profiling. The intent of this framework is to build a high-level understanding of a web archive's holdings, express it in a format that is easy to disseminate, and utilize a collection of *MementoMaps* from various web archives to perform efficient Memento routing in Memento aggregators. We first describe three research questions and how each question addresses certain aspects of the framework. Then we describe three main components of the framework that correspond to each of the research questions. Finally, we lay out an evaluation plan to assess the effectiveness of the framework.

4.1 RESEARCH QUESTIONS

We divide this work in three primary research questions about ingestion of archival holdings and voids, serialization of the summary, and utilization of the summary for Memento routing. We discuss our research questions one by one below.

4.1.1 RQ1: HOW TO LEARN AN ARCHIVE'S HOLDINGS AND VOIDS?

Under this research question we explore various ways to identify resources that are present in an archive or absent from it. Learning about an archive's holdings in this context means knowing URIs it contains and voids means knowing URIs it does not contain. Moreover, we are also interested in the datetime of mementos and the content language, when available. However, we are not interested in the content of mementos. This research question is explored in more details in Chapters 6 and 7. Another related question can be about "who would generate these profiles?" and the answer is both web archives and third parties, but since their access levels to the archival holdings would be different so will be their generated profiles.

We broadly divide the task of learning archival holdings in two categories, content- and usage-based.

Content-based Profiling

In the content-based profiling web archives tell what they contain. This is usually performed by accessing archival index or certain APIs they provide. We have two primary means of building content-based profiles:

- ***CDX Profiling*** – In CDX profiling we access archival indexes of a web archive as static files or through an API. We then build a higher-level understanding of archival holdings after filtering certain unnecessary index records off. CDX profiling gives the most comprehensive knowledge of archive's holdings. It is the fastest way to build an effective profile, but acquiring CDX data and keeping it up to date is difficult task. CDX files are gold-standard, but there are both organizational barriers and engineering challenges for most archives sharing this data. As such, we cannot assume it will be available. This can more effectively be done by web archives themselves, ideally by integrating *MementoMap* feature in archival replay systems.

- ***Fulltext Search Profiling*** – Some web archives provide fulltext search feature. Although, we are not interested in the content mementos, fulltext search gives a way to discover many URIs that are present in a archive. In this process we make fulltext search queries in web archives that support it and record URIs from returned results. In this mechanism we can learn at a maximum of N URIs per search query where N is the number of results returned per page by the fulltext search, which could be different for each archive. This value generally defaults to 10 or 20, but sometimes it is possible to customize how many results are desired by the client. It is worth noting that not all queries return a full page response. Also, as we continue to perform more queries we start to get some results that we have already learned about, so the overall learning rate slows down. Another down side of this approach is that it generally only surfaces textual URIs (i.e., those that point to HTML or text pages that are fulltext indexable).

Usage-based Profiling

In the usage-based profiling we observe web archives' responses as we attempt to fetch mementos or TimeMaps. In this process we learn about both absence and presence of queried URIs in the archive. In this approach the learning rate is slow as we learn about a single resource in each request. To avoid false negatives, these profiles are kept very high

level (for example only at the level of TLDs), which means they return more false positives. We can perform usage-based profiling in two ways:

- ***Sample URI Profiling*** – In the sample URI profiling we first build a sample URI set from a source (such as a directory or social media) depending of the purpose of the profile. Then we check to see if those URIs are absent or present in a number of web archives. Based on our observation we profile those archives' holdings.

- ***Response Cache Profiling*** – The response cache profiling is a special case of the sample URI profiling in which the sample of URIs is not built in advance, but the profiling is performed gradually as users access an archive. This profiling takes a long time to build and needs a constant feedback loop as the state of archiving changes over time (i.e., new resources are archived and some previously archived resources become inaccessible). This can be done at a Memento aggregator or at an archive itself by analyzing their access logs. LANL's Time Travel service uses this technique to train binary classifiers for Memento routing based on the past observations.

4.1.2 RQ2: HOW TO SUMMARIZE AND SERIALIZE ARCHIVAL HOLDINGS FOR DISSEMINATION?

Once we learn about a web archive's holdings, we need to summarize it and serialize for dissemination. For summarization, we first introduced various profiling policies under *HmPn* and *DLim* categories that are described in detail in Chapter 6. These policies summarized grouping URIs based on common prefixes up to a certain host or path depth and some other similar criteria. However, we later found these profiling policies were not very flexible as they did not allow merging profiles of two different policies. We then explored a more flexible summarization and serialization approach by adding wildcard support in SURT which is discussed and evaluated in Chapter 8.

We introduce a serialization file format called *MementoMap* for which we introduce a *Unified Key Value Store (UKVS)* [14] format that can be useful in many web archiving and other applications. Chapter 8 describes various ways to use this serialization format to express summarized holdings of an archive in many different ways as necessary for certain applications.

Generation and Compaction

We supply a list of URIs that are present in a web archive or a list of URIs that are absent from a web archive learned by various techniques described under RQ1 as an input to generate a *MementoMap*. Additionally, we also provide various configuration options that control the level of details in the *MementoMap*, which in turn decides when a URI prefix needs to be rolled up into a higher level group. This decision also considers the depth of the host or path segment in question and some heuristic values that we learned by analyzing a large archival index. We designed a single-pass, memory-efficient, and parallelization-friendly algorithm for *MementoMap* generation.

To limit or reduce the overall size of a *MementoMap* (either by the bytes count or number of records) we allow compaction of existing *MementoMaps*. A user can supply more aggressive compaction parameters and provide an existing *MementoMap* as input to get a reduced sized *MementoMap*. The compaction procedure iteratively and dynamically identifies sections of the *MementoMap* that can be rolled up into higher level nodes, reducing the size at the cost of a less detailed profiling in certain sub-trees of URIs. Like the generation procedure, we designed the compaction algorithm to be single-pass, memory-efficient, and parallelization-friendly, too. The generation algorithm internally uses the compaction procedure with appropriate parameters. These procedures are described in Chapter 8.

Updates and Merger

Many web archival collections evolve gradually over time while others grow in periodic batches. Hence, it is important to have a means to accommodate new changes in the holdings of an archive to its *MementoMap*. A naive approach would be to regenerate the complete *MementoMap* every once in a while from scratch, but this is wasteful, especially for large web archives. We allow incremental updates to existing *MementoMaps* by generating smaller *MementoMaps* from the freshly added data then merging it into the primary *MementoMap* and running the compaction procedure on it.

Pagination

Our *MementoMap* framework also allows arbitrary split of data which is essential for an effective pagination. Pagination is desired due to many practical reasons for example:

- An entity might want to store and disseminate its large *MementoMap* in chunks of manageable size

- An entity might want to organize smaller *MementoMaps* by time (such as yearly) or by structure (such as TLDs)
- An entity might want to avoid frequent updates and merger by providing smaller *MementoMaps* of freshly archived resources along with the old data on which the dust is settled already
- An entity might want to keep its *MementoMaps* separate for URIs it holds and URIs it does not, for maintainability

Dissemination and Discovery

For dissemination and discovery of *MementoMaps* we propose that web archives make their *MementoMap* available at the *well-known URI* [193] "`/.well-known/mementomap`" under their domain names. Alternatively, a custom *URI* can be advertised using the "`mementomap`" *link relation* (or "`rel`") in an *HTTP Link* header or *HTML <link>* element. Third parties hosting *MementoMaps* of other archives can use the "*anchor*" attribute of the *Link* header to advertise a different context. This primary entry point is for initial discovery, more *MementoMaps* (such as pages of the same profile or profiles made with different levels of details for different purposes) can be discovered from there using link relations or corresponding metadata of the primary *MementoMap*.

4.1.3 RQ3: HOW TO UTILIZE MEMENTOMAPS FOR MEMENTO ROUTING?

A *MementoMap* can support many applications such as coordinated crawling between archives, visualization of the archive's holdings, or routing of requests from a Memento Aggregator to the right archive. It is the latter application that is the focus of this work.

Considering a *MementoMap* as a document that describes holdings of a web archive, if a set of *MementoMaps* of a number of archives is provided, it is possible to predict candidate archives that might contain a given lookup URI. Essentially, by this analogy we convert this routing problem into an information retrieval problem. A few potential approaches of routing could be as following:

- ***Binary Classifier*** – In this approach we can build individual binary classifiers for each archive. For a given lookup URI we ask every classifier to predict the presence or absence of it in the corresponding web archive, then route the look up request to

archives with the positive response from the classifier. This approach was explored by Bornand et al. and used in the LANL Memento aggregator [66]. This approach keeps the prediction model of every archive separate and independent which allows easy update and per-archive tweaks. However, the down side of this approach is scalability. If the number of archives grows to hundreds or thousands, it becomes impractical to run as many classifiers and query every classifier in parallel.

- ***Inverted Index*** – In this approach we create an inverted index of keys from every *MementoMap* and keep record of their corresponding frequencies or other weights. This approach scales well with the growth of number of archives or *MementoMaps* as the index is precomputed and can always return only the relevant references to archives. *MementoMaps* of various archive can be more detailed than we need for routing purposes, in that case they can be further optimized during index creation. For example, *Arquivo.pt* focusses primarily on archiving `*.pt` sites among many other TLDs, so its profile can have more detailed summary of `*.pt` URIs. However, a Memento aggregator may not be getting a significant number of requests for `*.pt` sites, it we may choose to keep only the TLD-level entry for `pt,*` in the *Inverted Index*. This practice can significantly reduce the size of the index.

- ***Routing Score Estimation*** – In this approach we can leverage various machine learning techniques to estimate the likelihood of finding a URI in a web archive represented by one or more *MementoMaps*. Some available signals for model building include the host and path depths (say, token count) of the lookup URI, token count of the matched key in *MementoMap* (which can either be an exact match or a partial key with a wildcard), difference of the two counts, and stored frequency values. After estimating routing scores in each candidate archive above certain threshold we can rank them for routing and let the aggregator choose top-k archives or archives above certain threshold to aggregate from.

4.2 MEMENTOMAP COMPONENTS

We have divided the *MementoMap* framework in three major components that can be implemented and leveraged independently. These components are roughly parallel to the three research questions described above, but have an implementation perspective.

4.2.1 INGESTION

The ingestion component creates a uniform input for summarization and serialization step for a given web archive in the form of a list of URIs it holds and/or a list of URIs it does not. This component can process static CDX files, CDX server API, or access logs. Moreover, this component can also learn about the holdings of an archive via fulltext search or sample URI lookups.

4.2.2 SUMMARIZATION AND SERIALIZATION

The serialization component takes a list of URIs (sample or comprehensive) that are present in a web archive and/or a list of URIs that are absent from an archive as the input, identifies higher level groups of URIs that are present or absent in the archive, and yields a compact representation of the summary of archival holdings. These serialized representations can then be advertized on the web for automated discovery or disseminated using some other means. We use *UKVS* data format for serialization of *MementoMaps*.

While we provide an independent CLI tool and library for *MementoMap* creation and management, integrating it in every archival replay system is out of scope. Furthermore, we have seen interest in this work from the Portuguese Web Archive, the UK Web Archive, and the National Library of Australia, but ensuring adoption of *MementoMap* by web archives and aggregators is out of scope.

4.2.3 ROUTING

For efficient Memento routing we create an *Inverted Index* from *MementoMaps* of various web archives and perform binary search in it for all possible *URI-Keys* of the lookup URI. Then we utilize *Routing Score Estimation* technique as described above on matched index records to create a rank ordered list of archives as potential candidates for Memento routing. Combining these two approaches allows the system to scale well as the *Inverted Index* provides a small subset of an arbitrary number of web archives for further ranking evaluation. We intend to have a reference implementation of this approach and have it be used by our *MemGator* tool in the future.

4.3 EVALUATION PLAN

In this work we evaluate individual *MementoMaps* on cost vs. accuracy and the aggregated index of multiple archives for Memento routing will be evaluated on its routing

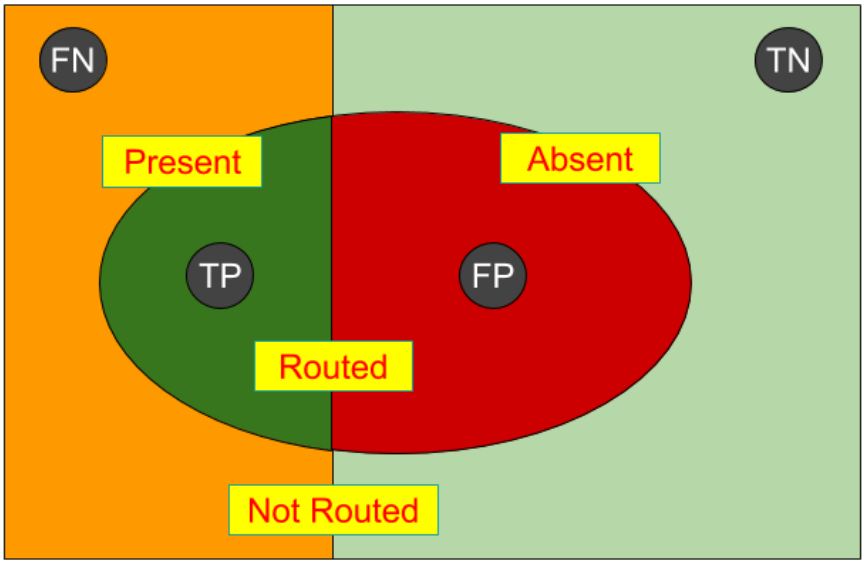


Fig. 35. Memento Routing Matrix

efficiency. Additionally, we also evaluate the freshness of *MementoMaps* as archives evolve over time.

In Figure 35 we illustrate how we map measures of Memento Routing to the traditional classification concept of Information Retrieval in the form of a confusion matrix. The outer large rectangle shown in the figure represents a set of sample URIs for which we want to measure routing statistic against a web archive using its profile. The rectangle is divided by a vertical line, on the left hand side of which are all the URIs from the sample set that are actually present in the archive and on the right hand side are the URIs that are not present. Inside area of the oval represents all the URIs that an archive profile predicts to be present in the archive and their corresponding requests should be routed to the archive while the outside area represents set of URIs from the sample that are not likely to be present in the archive based on the profile and should not be routed. Ideally, we would like to minimize false negatives (FN; when we fail to route a URI lookup to an archive that has the URI) and false positives (FP; when we route a URI lookup to an archive that does not have the URI) and maximize true positives (TP; when we route a URI lookup to an archive that has the URI) and true negatives (TN; when we do not route a URI lookup to an archive that does not have the URI). However, due to the inherent nature of classifiers an attempt to minimize FN would result in increased FP and an attempt to maximize TN would result in decreased TP. It is worth noting that the cost of false positives affects the infrastructure

(i.e., unnecessary work) while the cost of false negatives affects users (i.e., failure to discover resources of interest).

By analyzing many different configurations to generate archive profiles (i.e., *MementoMaps*) we can identify the relationship and right balance among precision, recall, and associated costs that are suitable for different application needs. We describe many relevant statistical measures (such as precision, recall, specificity, accuracy, and routing efficiency) in Chapter 6.

4.3.1 COST

Cost in this context means a lot of things including time of data acquisition and ingestion, time of processing, storage space, as well as the network bandwidth for dissemination. There is also some cost associated with incremental or periodic updates and synchronization of *MementoMaps*.

4.3.2 ACCURACY

Accuracy is the measurement of correctly identifying presence or absence of a set of URIs in an archive represented by a *MementoMap* (i.e., in the Information Retrieval sense, (TP+TN)/All). A more detailed *MementoMap* generally yields a better accuracy, but costs more to generate and maintain. We evaluate this inherent cost vs. accuracy relationship to identify a suitable balance depending on various factors such as available resources and the application of the *MementoMap*.

4.3.3 FRESHNESS

As an archive collects more resources or changes the state of existing mementos its corresponding *MementoMaps* will go stale as they will end up producing more false negatives and/or false positives (resulting in reduced accuracy). Factors that control the freshness include the rate and frequency of archiving and the level of detail of corresponding *MementoMaps*. A web archive that archives a lot of web resources and frequently indexes them for replay will need more frequent updates to its *MementoMaps*. On the other hand a less detailed *MementoMap* would require less frequent updates, but it has the inherent cost of producing more false positives (hence, low accuracy) in the first place. We evaluate these relationships to come up with some guidelines about the frequency of updates for *MementoMaps*.

4.3.4 ROUTING EFFICIENCY

Once we have a number of *MementoMaps* from various archives we create an *Inverted Index* from them and build a routing score estimator. By changing the level of details in the index and the cut-off value of routing scores we evaluate the accuracy of routing across a number of web archives. This will involve measuring the number of archives that we requested, but had no relevant mementos and the number of archives the we missed which had corresponding mementos to return for a given set of URIs. Along with measuring the number of archives, we also measure the number of mementos that we miss with varying configurations.

4.4 CHAPTER SUMMARY

In this chapter we outlined our *MementoMap* framework. We described our three primary research questions about learning an archive's holdings, summarizing and serializing it, and using it for Memento routing. We then described three major components of the framework parallel to these research question. Finally, we set out an evaluation plan for various aspects involving this framework such as cost, accuracy, freshness, and efficiency.

CHAPTER 5

TOOLS IMPLEMENTATION

During our work we implemented various related tools to aid the research process (such as processing and indexing *WARC/CDX/access log* data received from web archives, replaying mementos, aggregating web archives, generating/evaluating archive profiles, and disseminating archive profiles) and released them publicly under the open-source MIT license [199]. These tools include an archival replay system, a Memento aggregator, an archive profiler, an access log parser, and a reference implementation of *MementoMap* generator. Figure 36 summarizes our tool and software contributions in the web archiving ecosystem and annotates the functional placement of each tool. Some of these tools helped other researchers in carrying out their research both at Old Dominion University and outside, including international institutions. In this chapter we briefly describe these tools.

5.1 INTERPLANETARY WAYBACK

InterPlanetary Wayback (IPWB) [152, 20, 23, 22] is a distributed archival replay system that facilitates permanence and collaboration in web archives by disseminating contents of WARC files into the InterPlanetary File System (IPFS) [59] network. IPFS is a peer-to-peer content-addressable file system that inherently allows deduplication and facilitates opt-in replication. IPWB splits the header and payload of WARC response records before disseminating into IPFS to leverage deduplication, builds a CDXJ index with references to the IPFS hashes returns, and combines the header and payload from IPFS at the time of replay.

Figure 37 illustrates the indexing and replay process of IPWB. The indexer extracts records from the WARC store one record at a time, splits each record into HTTP header and payload, stores the two pieces into IPFS, and generates a CDXJ record using the returned references and some other metadata from the WARC record. The replay receives requests from users containing a lookup URI and optionally a datetime, queries for matching record in the CDXJ, fetches the corresponding header and payload from the IPFS Store (using references returned from the index record), combines them, and performs necessary transformation to build the response to the user. The software is made available under MIT license [19]. See Figure 83 (Appendix C) for the reference manual.

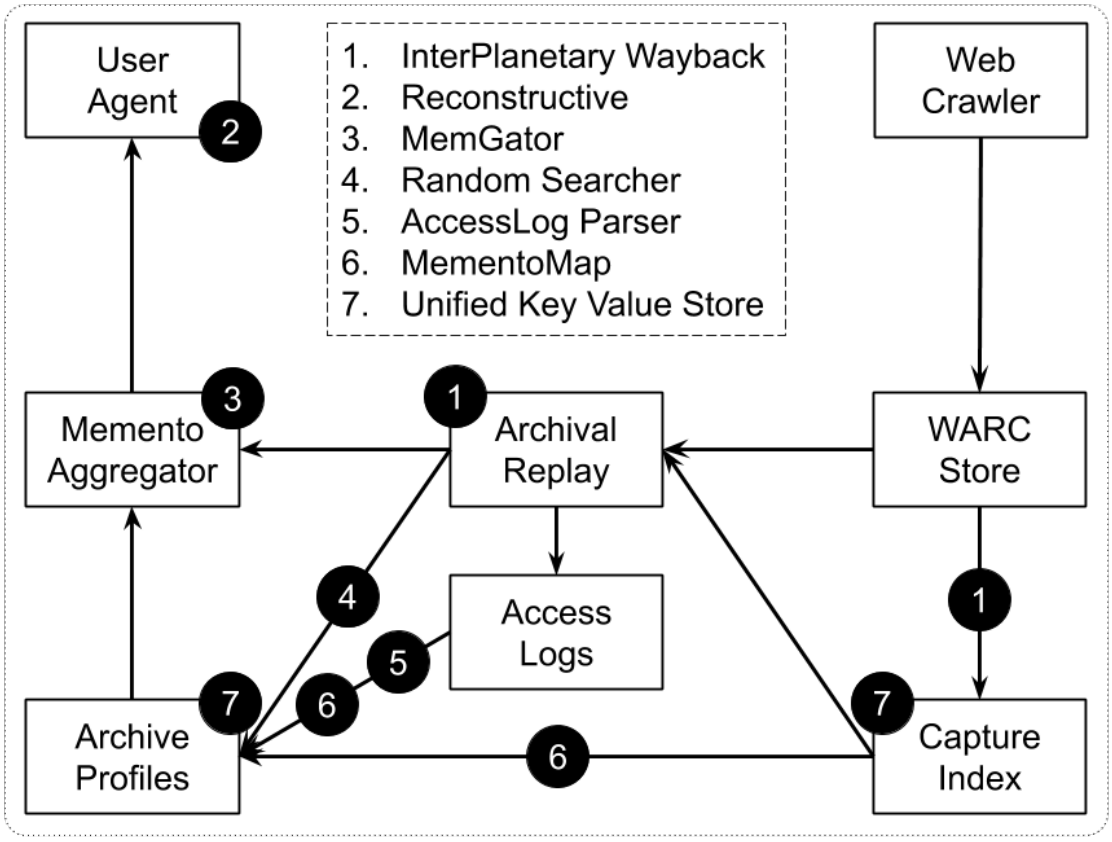


Fig. 36. Tools Contributions in the Web Archiving Ecosystem

IPWB is a Memento-compliant archival replay system with the potential to integrate *MementoMap* framework natively.

5.2 RECONSTRUCTIVE

Reconstructive is a JavaScript library that we built to facilitate client-side URL rerouting [21] and an unobtrusive archival banner inclusion [24] in the IPWB. It utilizes JavaScript's *ServiceWorker* API to intercept requests and reroutes them to their appropriate archived version as illustrated in Figure 38. We made both the rerouting and banner components available under MIT license as an independent library so that they can be utilized in other tools [17]. See Figures 84 and 85 (Appendix C) for the reference manual.

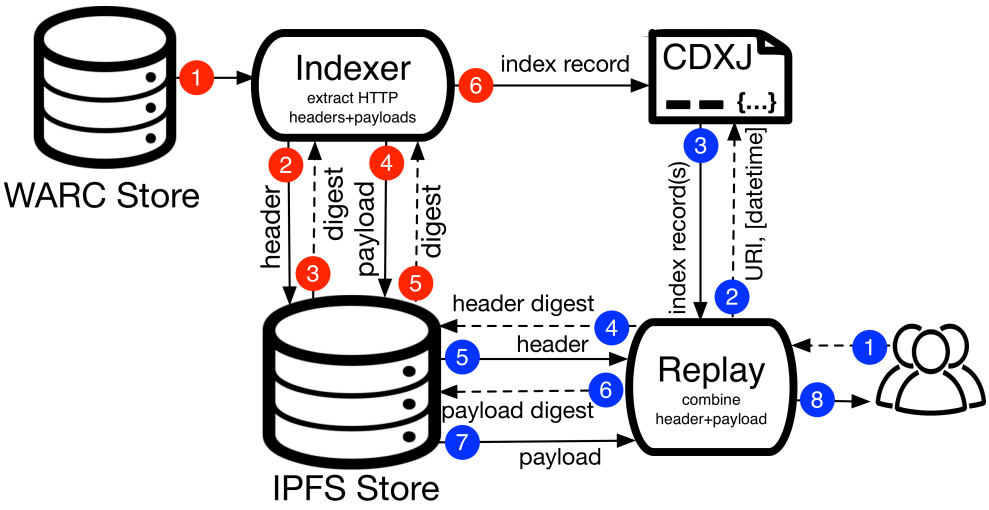


Fig. 37. IPWB Indexing and Replay Workflow

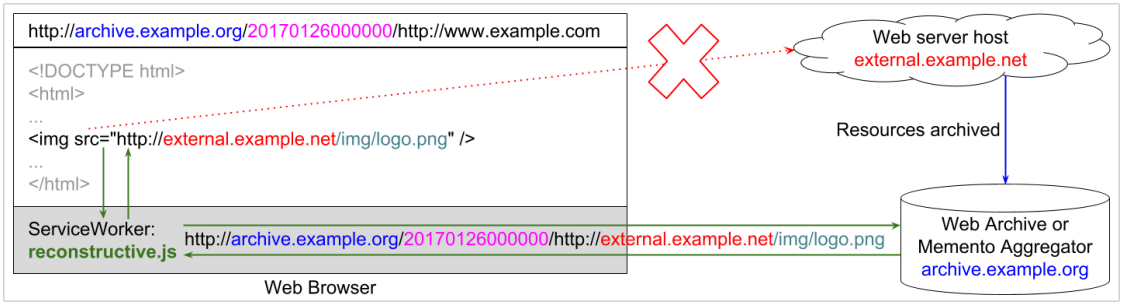


Fig. 38. Reconstructive Intercepts a Zombie Resource and Reroutes to its Archived Copy

5.3 MEMGATOR

MemGator is an open-source, easy to use, portable, concurrent, cross-platform, and self-documented Memento aggregator CLI and server tool written in Go [27]. Currently, there are two well-known Memento aggregator implementations: a closed-source one that powers LANL's Time Travel service and our open-source MemGator. MemGator works as a drop-in replacement of Time Travel API when used as a service, but has added features like the ability to use it as a CLI tool. While there are a few other open-source Memento clients, they are either not full-featured or not maintained [38, 143]. MemGator implements all the basic features of a Memento aggregator (e.g., TimeMap and TimeGate) and gives the ability to customize various options including which archives are aggregated and a selection of response

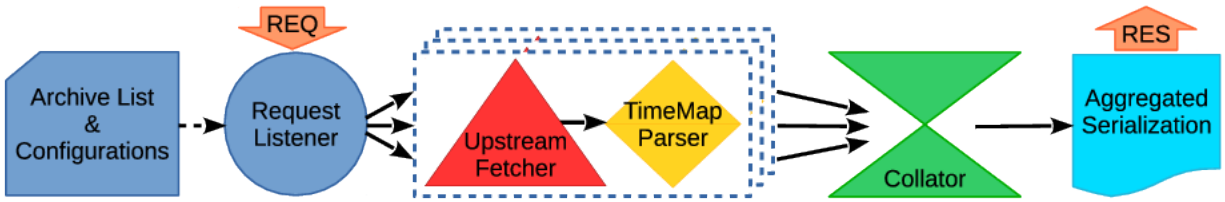


Fig. 39. MemGator Workflow Diagram

formats suitable for different application needs. It is being used heavily by tools and services such as Mink [157], WAIL [61, 156], OldWeb.today, and archiving research projects and has proved to be reliable even in conditions of extreme load [131]. The software is made available under MIT license [10].

Figure 39 illustrates the workflow of the MemGator implementation. The main thread (the request listener) loads the list of archives and other configuration options. When a lookup request is received, MemGator spins off *goroutines* (lightweight threads of Go language) for each individual archive. These individual *goroutines* fetch the TimeMap from individual archives independently. If the response is successful, the goroutine passes the data to a TimeMap parser via a *channel* (message passing mechanism of Go), which makes a linked list of the responses in a chronological order. The parser sends the linked list data to the collator which accumulates responses from each individual goroutine and merges them while maintaining the sorting. Once all *goroutines* are completed or timeout occurs, the accumulator passes the aggregated linked list to the serializer. Depending on the format requested by the client (such as Link or JSON), the data is serialized and returned as the response to the user. See Figure 86 (Appendix C) for the reference manual.

We have been running an instance of MemGator in server mode since early 2016 which is accessible publicly. This has provided useful logs for longitudinal study of Memento aggregators. MemGator is also the prime candidate of utilizing *MementoMap* framework for efficient Memento routing.

5.4 RANDOM SEARCHER

Random Searcher is a script to interact with the fulltext search interface of web archives to perform a random walk to discover a sample of the holdings of an archive. It is an implementation of the *Random Searcher Model* as described in Section 6.4.1 of Chapter 6. The script is made available under the Archive Profiler repository [6]. Since web archives

lack a unified standard fulltext search API, the script currently only supports Archive-It collections. However, it is possible to write modules to extend it to make it work with other archives in the future.

5.5 ACCESSLOG PARSER

AccessLog Parser is a Python package to parse *Common Log Format* and *Combined Log Format* [41] web server access logs. The package comes with a built-in CLI tool. While the package and too works with any general web server log file, it has some added features to recognize web archive replay access logs and extracts some web archive-specific attributes when present. These attributes include extraction of various known endpoints (e.g., Memento, TimeMap, TimeGate, and calendar), original URI (i.e., URI-R), any memento datetime value, optional type modifiers (e.g., "`id_`", "`if_`", "`im_`", etc.). The tool provides means to transform access datetime to formats that are friendly for sorting and arithmetic operations. Moreover, it provides various types of filters and output formatting options to meet users needs. The tool is made available publicly under MIT license [5]. See Figure 87 (Appendix C) for the reference manual.

5.6 MEMENTOMAP

Initially, we created a collection of scripts to process CDX files, build archive profiles with various profiling policies, evaluate effectiveness of generated profiles, and prepare analysis reports. This was our early implementation attempt which helped us create baseline gold standard datasets. We learned a lot from this and identified some shortcomings of our preliminary approaches. These scripts are still available publicly for reference under the Archive Profiler repository [6].

MementoMap tool is the reincarnation of now defunct Archive Profiler. It implements the most recent proposal of the *MementoMap* framework. The tool allows generation of *MementoMaps*, compaction of existing ones, and lookup of an individual URI or a batch of URIs in a given *MementoMap* file. This tool is described in more details in Chapter 8 under Section 8.3.

The tool can be used as a Python module to facilitate native *MementoMap* support in various archival replay systems such as IPWB and PyWB. Additionally, it can be extended to run as a service to provide and HTTP API for many *MementoMaps*, which can then be utilized by aggregators to avoid broadcasting. The software is made available under MIT license [9]. See Figure 88 (Appendix C) for the reference manual.

5.7 UNIFIED KEY VALUE STORE

Unified Key Value Store is a file format that we discuss under Chapter 8. While we do not have an independent generator and parser script of the format, we have its support added in the InterPlanetary Wayback and *MementoMap* tools as described above in Sections 5.1 and 5.6. We plan to extract it into a separate library/tool in the future to be used independently in other places.

5.8 CHAPTER SUMMARY

In this chapter we described various open-source tools, libraries, and scripts that we built and released during our research. These include: InterPlanetary Wayback (an archival replay system), Reconstructive (a client-side URL rerouting and archival banner injection script), MemGator (a Memento aggregator), Archive Profiler (a set of script to profile and analyze holdings of web archives), Random Searcher (a script to sample holding of web archives via fulltext search), AccessLog Parser (a web server access log parser with added features for web archives), *MementoMap* (a tool and module to generate, manage, and utilize archive profiles), and Unified Key Value Store (a file format for flexible and efficient key-value data storage). These tools are contributions of this work some of which can be useful in other research and development works beyond the scope of archive profiling.

CHAPTER 6

LEARNING ARCHIVAL HOLDINGS

In order to route lookup requests to the archives that are likely to return good results, we need to learn about the holdings of various archives. This chapter addresses our first research question, "***RQ1**: How to learn about the holdings and voids of an archive?*". In this chapter we only explore the first part of this question that concerns how to learn about the holdings of an archive. In Chapter 7 we will address the second part of the question that deals with archival voids.

We examine various strategies of learning about the holding of web archives. We have the following four approaches to discover the holdings of an archive:

1. ***CDX Profiling*** – If an archive's CDX files are available, then we can generate profiles with complete knowledge of their holdings [28, 29].

2. ***Fulltext Search Profiling*** – Random query terms are sent to the fulltext search interface of the archive (if present) and from the search response we learn the URIs that it holds. These URIs are then utilized to build archive profiles [30].

3. ***Sample URI Profiling*** – A sample set of URIs are used to query the archive and build the profile from the successful responses. This is quite wasteful, as <5% of the sample URIs are found in any archive. AlSum et al. explored the *Sample URI Profiling*, but only on Top-Level Domains (TLDs) [40].

4. ***Response Cache Profiling*** – This approach depends on the response data collected by an aggregator over a period of time as queries are made to the archive. Cached responses are analyzed to learn about their holdings. As a result the *Response Cache Profiling* is based on what people were looking for as opposed to what is in the archives. This approach is different from the first two in a way that it is a usage-based profiling approach while the first two are content-based. Bornand et al. explored this approach by building binary classifiers from LANL's Time Travel aggregator cache data [66].

An archive profile has an inherent trade-off in its size vs. its ability to accurately describe the holdings of the archive. If a profile records each individual original URI the size of the

profile can grow quite large and difficult to share, query, and update. On the one hand, an aggregator making routing decisions will have perfect knowledge about whether or not an archive holds archived copies of the page, or Mementos. On the other hand, if a profile contains just the summaries of top-level domains (TLDs) of an archive the profile size will be small but can result in many unnecessary queries being sent to the archive. For example, the presence of a single Memento of `bbc.co.uk` will result in the profile advertising `.uk` holdings even though this may not be reflective of the archive's collection policy. In this chapter we examine various policies for generating profiles, from the extremes of using the entire URI-R to just the TLD.

6.1 ARCHIVE PROFILE DATA STRUCTURE

In simple terms an archive profile data structure is a key-value store with some additional metadata. Keys are some form of a URI transformation (optionally, some other fields such as date, language, etc.) and the value is an indicator of the holdings in the given archive corresponding to the key. We discuss this further in Chapter 8.

Figure 40 illustrates a sample archive profile. Lines 1–4 of Figure 40 contain some metadata about the archive and the profile itself. Line 3 describes that there is only one field used as the lookup key which is SURT of the lookup URI. If there were multiple lookup fields (space separated), "`!keys`" would hold an array of those filed names in the order they appear in the data. The "`type`" attribute of the metadata describes that the document is a *URI-Key* profile using the policy *H3P1* (as discussed in Section 6.2). Lines 5–8 illustrate the statistical data about the archive holdings. Each line of the data section holds one record which has one or more key fields followed by a single line JSON as the value. There can be an arbitrary number of attributes in the JSON block, but in this example we use the sum of URI-M counts from all the profiles under each *URI-Key* as "frequency" and keep track of the number of profiles they came from as "spread".

6.2 URI-KEYS AND PROFILING POLICIES

URI-Key is a term we introduce to describe the keys generated by transforming URIs based on various policies. This is used as a lookup key in the profile. Initially, we have created policies that can be classified in two structural categories, *HmPn* and *DLim*. These policies enabled a baseline for evaluation. Later, we introduced wild-card support in *URI-Keys* to get away with rigid profiling policies and make them more flexible.

```
!context ["https://oduwsdl.github.io/contexts/archiveprofile"]
!id      {"uri": "http://www.webarchive.org.uk/ukwa/"}
!keys    ["surt"]
!meta    {"name": "UKWA Collection", "type": "urikey#H3P1", "...": "..."}
com,dilos,)/region {"frequency": 14, "spread": 2}
edu,orst,)/groups  {"frequency": 3, "spread": 1}
uk,ac,rpms,)/      {"frequency": 124, "spread": 1}
uk,co,bbc,)/images {"frequency": 152, "spread": 3}
```

Fig. 40. Sample Archive Profile Data Structure

6.2.1 HMPN POLICY

Policies of the generic form *HmPn* mean that the keys will have a maximum of "m" segments from the hostname and a maximum of "n" segments from the path. A *URI-Key* policy with only one hostname segment and no path segments (*H1P0*) is called *TLD-only* policy (as discussed in Section 3.9.2). *H3P0* policy covers most of the registered domains (that have one or two segments in their suffix [185], such as `.com` or `.co.uk`). If the number of segments are not limited, they are denoted with an "x", for example, *HxP1* policy covers any number of hostname segments with maximum of one path segment and *HxPx* means any number of hostname and path segments. Note that the *HxPx* policy is not the same as the *URIR* policy (as discussed in Section 3.9.1) as *HxPx* strips off the query parameters from the URI, while the *URIR* policy stores complete URIs. For example, `https://youtube.com/watch?v=QNo5ZDvKuHg` becomes `com,youtube,)/watch?v=QNo5ZDvKuHg` under *URIR* policy and `com,youtube,)/watch` under *HxPx* policy.

6.2.2 DLIM POLICY

Policies of the generic form *DLim* are based on the registered domain name (`RD`), the number of segments in subdomain (`#S`), path (`#P`), and query (`#Q`) sections of a URI, and the initial letter of the path (`PI`). A generic template for this category of *URI-Keys* can be given as "`RD[#S[/#P[/#Q[/PI]]]]`". The *DDom* policy includes only the registered domain name in SURT format, while *DSub*, *DPth*, *DQry*, and *DIni* policies also include sections of the template up to `#S`, `#P`, `#Q`, and `PI` respectively. In addition to these, `RD` can be further decomposed to form `DSuf` and `DTld` as domain suffix and top-level domain respectively. `DTld` is equivalent to `H1P0`, but may or may not be the same as `DSuf` (e.g., `.com` can be

```
# A canonicalized SURT
uk,co,bbc,news,)/images/logo.png?height=80&rotate=90%c2%b0&width=200
  ↓ URI FEATURE EXTRACTION ↓
Registered Domain (RD): uk,co,bbc,)/
Path Initial (PI):      i
Subdomain (#S):         1
Path (#P):              2
Query (#Q):             3
  ↓ URI-Key GENERATION ↓
HmPn:
  H1P0: uk,)/
  H3P0: uk,co,bbc,)/
  HxP1: uk,co,bbc,news,)/images
DLim:
  DDom: uk,co,bbc,)/
  DPth: uk,co,bbc,)/1/2
  DIni: uk,co,bbc,)/1/2/3/i
```

Fig. 41. Illustration of URI-Key Generation

both a TLD and domain suffix, but `.co.uk` is only a domain suffix).

6.2.3 URI-KEY GENERATION

Figure 41 illustrates the process of generating *URI-Keys* from a URI. URIs are first canonicalized then go through SURT. For *HmPn* policies, query section and fragment identifier of the URI are removed (if present), then depending on the values of "m" and "n" any excess portions from the SURT URL are chopped off. The hostname segments are given precedence over the path segments in a way that no path segment is added until all the hostname segments are included, hence `uk,co,)/images` is an invalid *URI-Key*, but `uk,co,bbc,news,)/images` would be valid if the hostname is `news.bbc.co.uk` or `www.news.bbc.co.uk`. For *DLim* policies, the registered domain name is extracted with the help of the Public Suffix list (which is updated periodically). Then depending on the individual policies, segments from zero or more sections (such as subdomain and path) of the URI are counted, and if necessary, the initial letter of the first path segment is extracted (replaced with a "-" if not alphanumeric). These values are then placed inside the template to form the *URI-Key*.

6.3 PROFILING THROUGH CDX SUMMARIZATION

We first collected CDX files from three different web archives of different sizes. Then we generated 23 different profiles (with 17 *HmPn* policies, five *DLim* policies, and one *URIR* policy) for each archive's dataset to measure their resource requirement and routing efficiency.

6.3.1 DATASETS

For the evaluation we prepared two types of datasets, archive profiles and query URI-Rs. For profiles, we used three archives (Table 3):

1. ***Archive-It Collections*** – We acquired the complete holdings of Archive-It [137] before 2013 and indexed the collections (in CDX format) to create a dark archive of the service. The archive has 2,952 collections with more than 5.3 billion URI-Ms and about 1.9 billion unique URI-Rs. It has more than 1.9 million ARC/WARC files that take about 230 TB disk space in compressed format. We created *URI-Key* profiles with various policies for the entire archive from the CDX files.

2. ***UK Web Archive*** – We acquired a publicly available CDX index dataset from UKWA [242]. The dataset has separate CDX files for each year (starting from year 1996). We created individual *URI-Key* profiles from each of the early 10 years of CDX files (from year 1996 to 2005) with different profiling policies. We also created a combined profile by incrementally accumulating data for each successive year to analyze the growth. These 10 years of CDX files have about 1.7 billion URI-Ms and about 0.7 billion unique URI-Rs.

3. ***Stanford University Archive*** – We acquired the complete CDX files of the Stanford Web Archive Portal [234]. This dataset is relatively smaller than the above two and has data ranging from the year 2013 to 2015. It has about 25 million URI-Ms and 12 million unique URI-Rs.

We created the second dataset by collecting four million URIs; one million random unique URI-R samples from each of these four sources:

1. ***DMOZ Archive*** – URIs used in a study of HTTP methods [18].

2. ***IA Wayback Access Log*** – URIs extracted from the access log used in a study of links to the Internet Archive (IA) content [37].

Table 3. Archive Dataset Size

Archive	**URI-Rs**	**URI-Ms**	**Size**
Archive-It	1.9B	5.3B	1.8TB
UKWA	0.7B	1.7B	0.5TB
Stanford	12M	25M	8.3GB

Table 4. Presence of the Sample Query URI-Rs in Each Archive

Sample (1M URIs Each)	**Archive-It**	**UKWA**	**Stanford**	**Union of {AIT, UKWA, SUA}**
DMOZ	4.097%	3.594%	0.034%	7.575%
Memento Proxy Logs	4.182%	0.408%	0.046%	4.527%
IA Wayback Logs	3.716%	0.519%	0.039%	4.165%
UKWA Wayback Logs	0.108%	0.034%	0.002%	0.134%

3. ***Memento Aggregator Access Log*** – URIs extracted from the access log used in a previous archive profiling study [40].

4. ***UKWA Wayback Access Log*** – URIs extracted from an anonymized sample of the recent UKWA Wayback access logs.

We then checked to see how much of each sample set is archived in each archive (or prevalence in statistical terms). Table 4 shows that out of 1 million sample query URI-Rs from DMOZ only 40,969 (or 4.097%) URI-R are archived in Archive-It, 35,936 (or 3.594%) in UKWA, and 341 (or 0.034%) in Stanford. The union of the archived URI-Rs is 75,745 (or 7.575%), almost same as the summation of the individual values, which shows there is little overlap among these archives. The table also shows that none of the individual archives have more than 5% of any sample set, which means the prevalence is <5% for any pair of an archive and a sample set.

We also examined how much of the archived URIs from different sample sets overlap in various archives. Figure 7 (Chapter 1) shows that few request URI-Rs are held in any archive and very few are held in all archives. For example, there are only two URI-Rs in 1 million DMOZ sample URI-Rs that are archived in all the three archives and no archive

has more than 4.1% of the URI-Rs, while 924,255 (or 92.426%) URI-Rs are not archive in any of the three archives as illustrated in Figure 7a.

6.3.2 PROFILE GROWTH ANALYSIS

In commonly used CDX files each entry corresponds to a URI-M[1] (as described in Section 2.7.4 of Chapter 2). The length of each line in a CDX file depends on the length of the URI-R in it. Our experiment shows that the average number of bytes per line (α) in our dataset is about 275, which means every one gigabyte of CDX file holds about 3.9 million URI-Ms. Equation 1 approximates the pattern shown in Figure 42a where C_m denotes the number of URI-Ms and S_c denotes the total size of CDX files in bytes.

$$C_m = \frac{S_c}{\alpha} \tag{1}$$

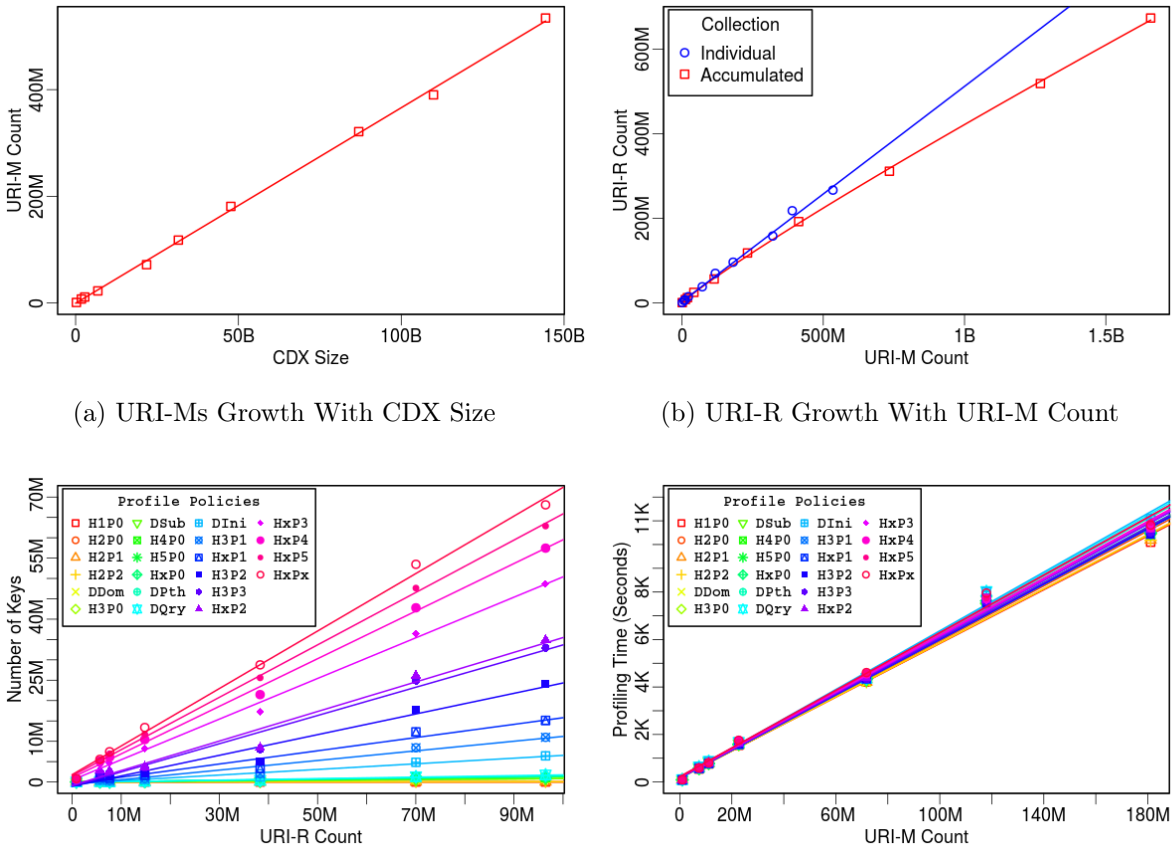
Figure 42b shows the relationship between URI-M Count (C_m) and URI-R Count (C_r) in two ways; 1) for each year of UKWA CDX files individually as if they were separate collections and 2) accumulated ten consecutive years of data one year at a time while each time it recalculates total number of unique URI-Rs visited. The ratio of URI-M Count to URI-R Count (γ) as shown in Equation 2 is indicative of the average number of revisits per URI-R for any given time period. The value of γ varies from one archive to the other because some archives perform shallow archiving while others revisit old URI-Rs regularly to capture as many changes to those resources as possible [178]. In our dataset γ_{UKWA}=2.46 and γ_{AIT}=2.87. The accumulated trend better accounts for how archives actually grow over time. It follows Heaps' Law [97] as shown in Equation 3 where URI-Rs are analogous to unique words in a corpus. K and β are free parameters that are affected by the value of γ, but the actual values are determined empirically. For the UKWA dataset K=2.686 and β=0.911.

$$\gamma = \frac{C_m}{C_r} \tag{2}$$

$$C_r = KC_m^{\beta} \tag{3}$$

URI-Keys are used as lookup keys in a profile. The number of *URI-Keys* is a function of URI-Rs, not URI-Ms, hence increasing URI-Ms without introducing new URI-Rs does

[1]In our dataset Archive-It has 0.71% non-HTTP entries (such as DNS queries or malformed records) in their CDX files while UKWA has no non-HTTP entries.



(a) URI-Ms Growth With CDX Size

(b) URI-R Growth With URI-M Count

(c) Space Cost

(d) Time Cost

Fig. 42. Growth and Costs Analysis for Different Profiling Policies in UKWA

not affect the number of *URI-Keys*, instead, it only changes the statistical values (such as "frequency") stored for the corresponding key. Figure 42c shows the number of unique *URI-Keys* (C_k) generated for different numbers of unique URI-Rs (C_r) on different profiling policies. Every profiling policy follows a straight line with a slope value (ϕ_{policy}) and zero *y-axis* intersect because for zero URI-Rs there will be zero *URI-Keys*. For a given profile policy, we define the ratio of *URI-Key Count* (C_k) to *URI-R Count* (C_r) as *Relative Cost* (ϕ_{policy}) as shown in Equation 4. The *Relative Cost* varies from one archive to the other based on their crawling policy. Archives that crawl only a few URIs from each domain will have relatively higher *Relative Cost* than those who crawl most of the URIs from each domain they visit. Table 5 lists ϕ_{policy} values for UKWA dataset.

$$\phi_{policy} = \frac{C_k}{C_r} \tag{4}$$

Figure 42d illustrates the time required to generate profiles with different policies and different data sizes. Previously we used a memory based profiling approach, but now we use a file based approach instead. The latter approach scales better and allows for distributed processing. Our experiment shows that the profiling time is mostly independent of the policy used, but we found that *DLim* policies take slightly more time than *HmPn* policies because they require more effort to extract the registered domain based on the public suffix list. We found that about 95% of the profiling time is spent on generating keys from URIs and storing them in a temporary file while the remaining time is used for sorting and counting the keys and writing the final profile file. Hence, a memory based key-value store can be used for the temporary data to speed up the process. Also, when an archive has a high value of γ, we can save significant amount of time by generating keys from each URI-R only once then adjust the "frequency" according to the number of occurrences of the corresponding URI-Rs. However, when profiles are generated on small sub-sets of the archive (for example, for incremental updates) there is a lower chance of revisits in the small subset. The mean time to generate a key for one CDX entry τ can be estimated using Equation 5 where T is the time required to generate a profile from a CDX file with C_m URI-Ms. The value of τ depends on the processing power, memory, and I/O speed of the machine. On our test machine it was between 5.7×10^{-5} to 6.2×10^{-5} seconds per URI-M (wall clock time). As a result, we were able to generate a profile from a 45GB CDX file with 181 million URI-Ms and 96 million unique URI-Rs in approximately three hours.

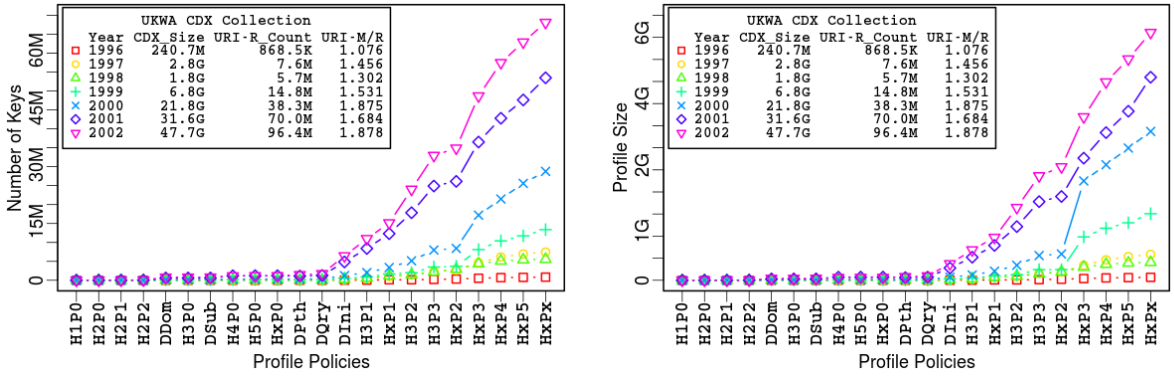
$$\tau = \frac{T}{C_m} \tag{5}$$

Figure 43 illustrates the correlation among the number of *URI-Keys* 43a and profile size on disk 43b for various collection sizes with various profiling policies. The policies are sorted in the increasing order of their resource requirement. If these generated profiles are compressed (using gzip [93, 112] with the default compression level), for bigger profiles they use about 15 times less storage than uncompressed profiles, but result in a similar growth trend. These figures are helpful in identifying the right profiling policy depending on the available resources such as storage, memory, computing power. Here are some common observations in these figures:

- For host segments less than three, path segments do not make a significant difference as they are not included unless all the host segments of the URI are already included.

Table 5. Relative Cost of Various Profiling Policies for UKWA

Policy	**Relative Cost (ϕ_{policy})**
H1P0	<0.00001
H2P0	0.00026
H2P1	0.00038
H2P2	0.00056
DDom	0.00857
H3P0	0.00858
DSub	0.00876
H4P0	0.01340
H5P0	0.01368
HxP0	0.01371
DPth	0.01577
DQry	0.01838
DIni	0.06892
H3P1	0.11812
HxP1	0.16247
H3P2	0.25379
H3P3	0.34668
HxP2	0.36298
HxP3	0.49902
HxP4	0.58442
HxP5	0.64365
HxPx	0.70583
URIR	1.00000



(a) URI-Key Count

(b) Profile Size

Fig. 43. Resource Requirement for Various Profiling Policies and Collection Sizes

Table 6. Confusion Matrix of Memento Routing

		Actual	
		Present	**Absent**
Predicted	**Routed**	True Positives (TP)	False Positives (FP)
	Not Routed	False Negatives (FN)	True Negatives (TN)

- Keeping either the hostname or path segments constant while increasing the other increases in the value, but the growth rate decreases as the segment count increases.
- The last data-point (HxPx) shows a different trend, it is not just one path segment ahead of its predecessor (HxP5), but any path segments more than 5 are included in it.
- Growth due to path segments is significantly faster than the growth due to hostname segments.
- If a single path segment is to be included, it is better to include all host segments, which provides more details without causing any significant cost overhead.

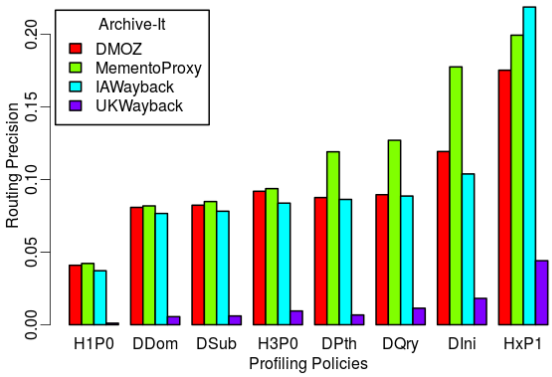
Table 7. Relative Cost, Precision, Specificity, and Accuracy of Profiling Policy Groups

Group	**Policies**	**Cost**	**Precision**	**Specificity and Accuracy**
G_1	H1P0/TLD	Bound by # of TLDs	< 0.05	≈ 0.01
G_2	H3P0, DDom, DSub, DPth, DQry	< 0.01	$\approx 2G_1$	≈ 0.78
G_3	DIni	$\approx 2G_2$	$\approx 3G_1 - 4G_1$	≈ 0.88
G_4	HxP1	$\approx 5G_3$	$\approx 5G_1 - 7G_1$	≈ 0.94
G_5	Higher HmPn	0.4 – 0.7	Not Explored	Not Explored
G_6	URIR	1.0	1.0	1.0

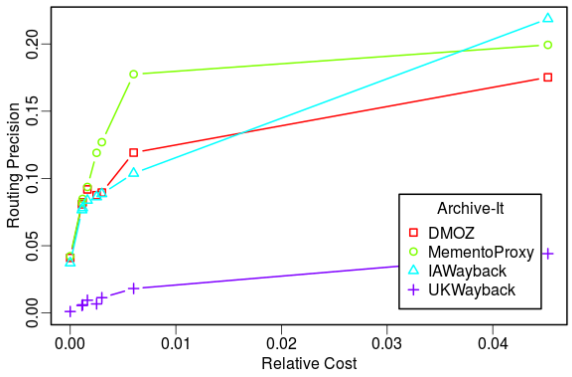
6.3.3 ROUTING EFFICIENCY

To analyze the routing efficiency of profiles, we picked eight policies from the 23 policies we have used to generate profiles for all three archives in our dataset. These policies are *TLD-only* (*H1P0*), *H3P0*, *HxP1*, and all the five variations of the *DLim* policies. We then examined the presence of resources in the archives for our four query URI sample sets, each containing one million unique URI-Rs. Based on the profiling policy, a query URI-R is transformed into a *URI-Key* then it is looked up in the *URI-Key* profile keys to predict its presence.

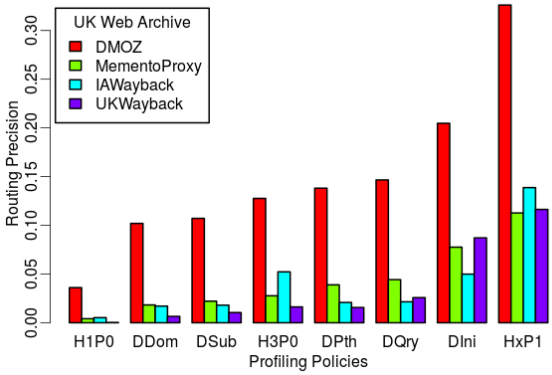
When a sample set of URI-Rs is examined against an archive using a profile to route the matching URIs to the archive, each URI from the sample set may fall in one of the four categories; 1) present in the archive and routed to the archive by the profile (true positive or TP), 2) not present in the archive and not routed to the archive by the profile (true negative or TN), 3) not present in the archive, but routed to the archive by the profile (false positive or FP), and 4) present in the archive, but not routed to the archive by the profile (Table 6). For example, if the query URI (in SURT format) is `uk,co,bbc,)/images/test.png` and it is examined against the profile illustrated in Figure 40 then it will be routed to the archive because the profile has a matching key `uk,co,bbc,)/images` in it. In this case, if the query URI is actually present in the archive it will be a true positive otherwise a false positive. Similarly, if the query URI is `uk,co,bbc,)/videos/test.mp4`, it will not be routed by the profile to the archive because the profile does not have a matching key. In this case, if



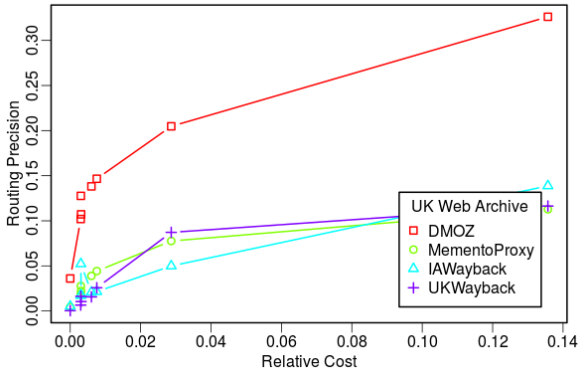
(a) Archive-It: Precision



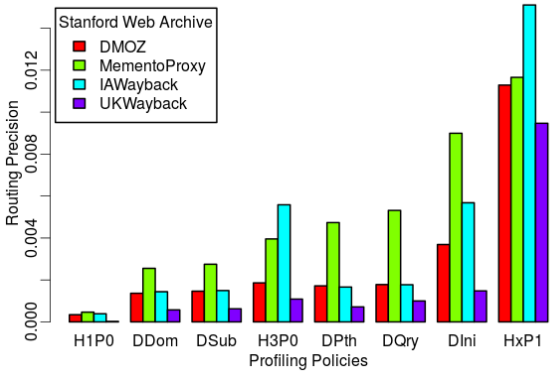
(b) Archive-It: Precision vs. Cost



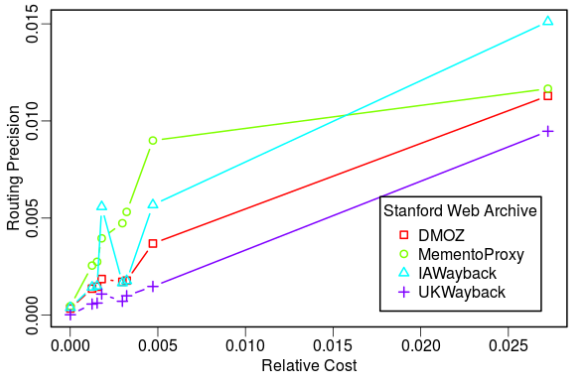
(c) UKWA: Precision



(d) UKWA: Precision vs. Cost

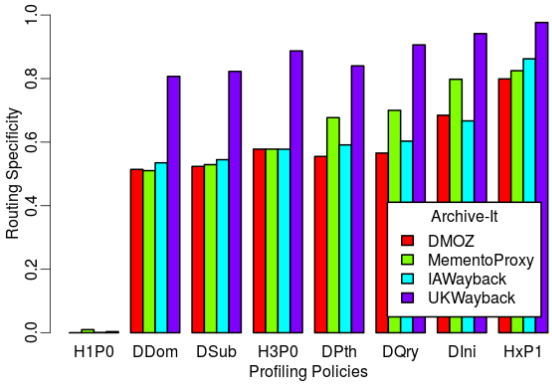


(e) Stanford: Precision

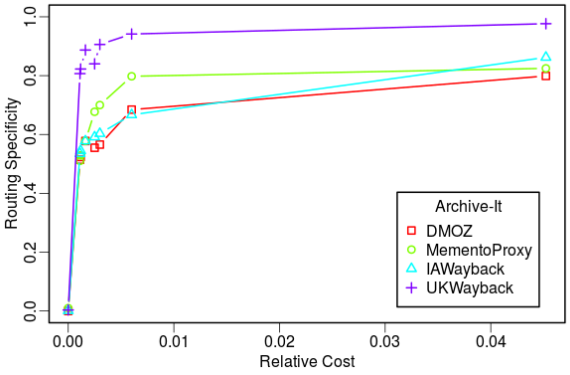


(f) Stanford: Precision vs. Cost

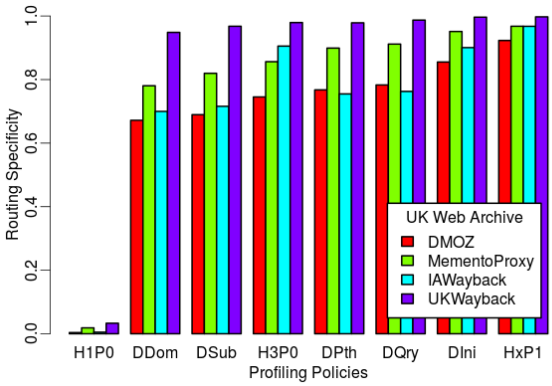
Fig. 44. Routing Precision of Different Profiling Policies in Different Archives



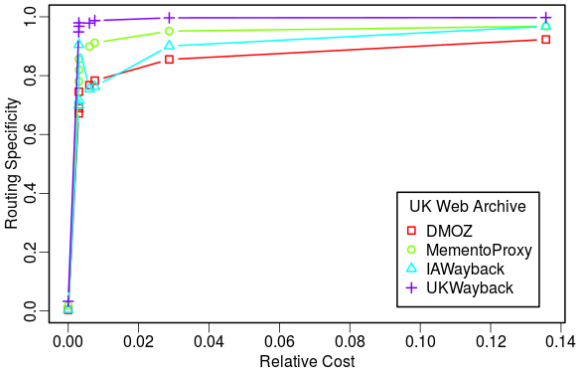
(a) Archive-It: Specificity



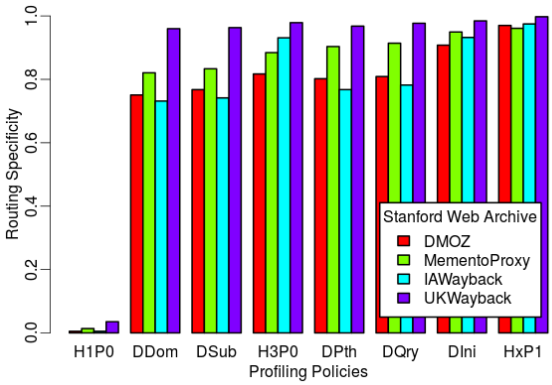
(b) Archive-It: Specificity vs. Cost



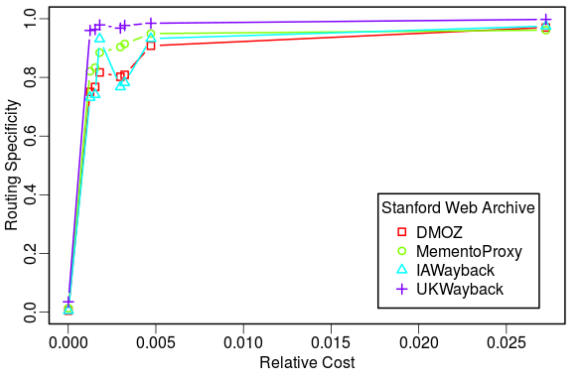
(c) UKWA: Specificity



(d) UKWA: Specificity vs. Cost

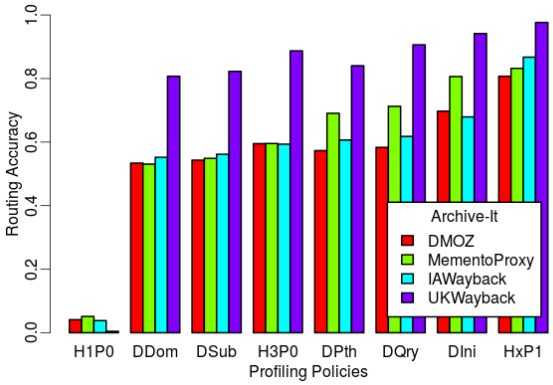


(e) Stanford: Specificity

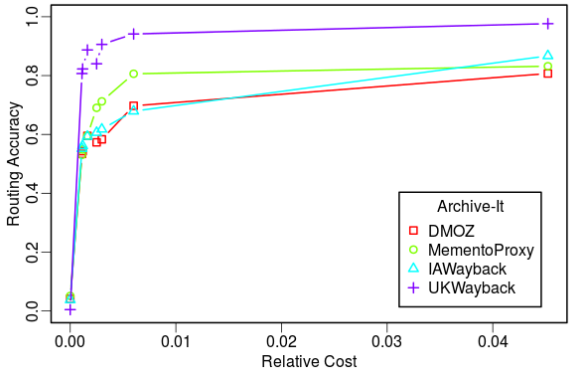


(f) Stanford: Specificity vs. Cost

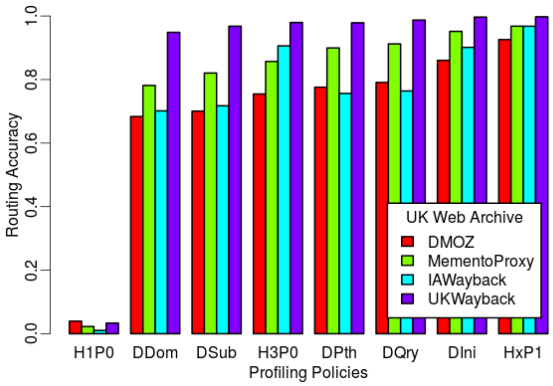
Fig. 45. Routing Specificity of Different Profiling Policies in Different Archives



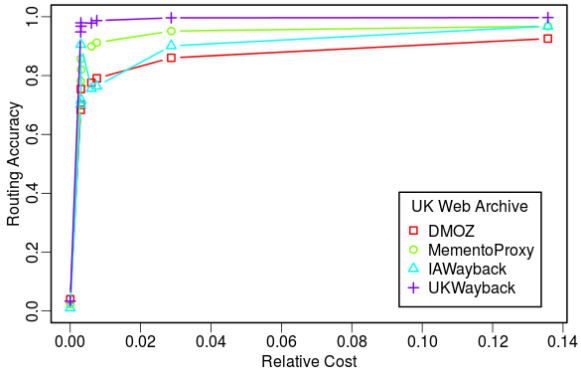
(a) Archive-It: Accuracy



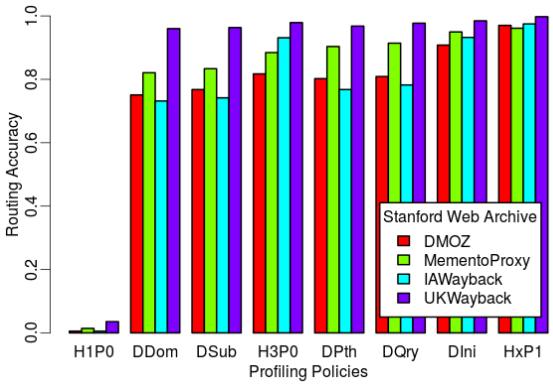
(b) Archive-It: Accuracy vs. Cost



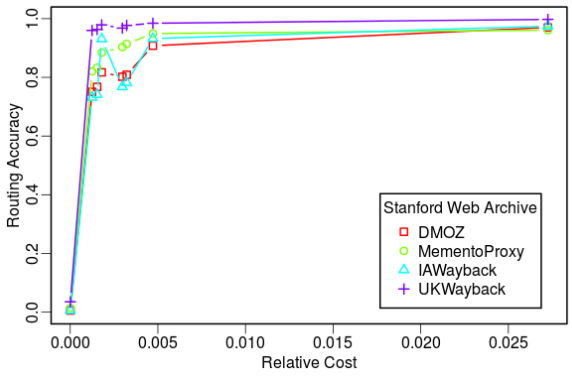
(c) UKWA: Accuracy



(d) UKWA: Accuracy vs. Cost



(e) Stanford: Accuracy



(f) Stanford: Accuracy vs. Cost

Fig. 46. Routing Accuracy of Different Profiling Policies in Different Archives

the query URI is actually present in the archive it will be a false negative otherwise a true negative.

$$\text{Prevalence} = \frac{|\text{Sample URI-Rs Present in an Archive}|}{|\text{Total URI-Rs in a Sample}|} \tag{6}$$

$$\text{Recall}_{policy} = \frac{|\text{URI-Rs Correctly Routed by Profile}_{policy} \text{ to an Archive}|}{|\text{URI-Rs Present in an Archive}|} \tag{7}$$

$$\text{Precision}_{policy} = \frac{|\text{URI-Rs Correctly Routed by Profile}_{policy} \text{ to an Archive}|}{|\text{URI-Rs Routed by Profile}_{policy} \text{ to an Archive}|} \tag{8}$$

$$\text{Specificity}_{policy} = \frac{|\text{URI-Rs Correctly Not Routed by Profile}_{policy} \text{ to an Archive}|}{|\text{URI-Rs Not Present in an Archive}|} \tag{9}$$

$$\text{Accuracy}_{policy} = \frac{|\text{URI-Rs Correctly Routed or Not Routed by Profile}_{policy} \text{ to an Archive}|}{|\text{Total URI-Rs in the Sample}|} \tag{10}$$

High values of true positives and true negatives indicate the ability of the profile to accurately route or not route a lookup to an archive. However, a high value of false positives indicates that in many cases the profile predicts that the lookup URI-Rs are present in an archive, but they are not. This does not affect the client much, as the intent of a client, usually, is to increase the Recall (to get as many Mementos as possible), but it costs the aggregator and archives in the form of unnecessary work. In contrast, a high value of false negatives indicates that in many cases the profile fails to predict the presence of Mementos in an archive for the given lookup URI-Rs. This saves resources of the aggregator and archives, but affects the ability of a client to discover all the Mementos of a query URI-R. Small profiles (such as *H1P0*) remain fresh for long time, but yield a high FP value. However, detailed profiles (such as *HxPx*) yield low FP, but become stale quickly as an archive crawls more resources, hence require regular updates. The FN value does not depend on how detailed a profile is, instead how much a profile knows about an archive's holdings. A profile will yield FNs if it has an incomplete knowledge of an archive (such as when the profile becomes stale) or the profile is utilized in a way that the values of "frequency" and "spread" below certain threshold are ignored.

To establish a baseline, we only check to see if the lookup key is present in the profile and do not use the statistical values (such as their frequency) present in the profile. This brings some false positives in the result, but no false negatives, hence we do not miss any URI-Ms from the archive for the query URI-R. In other words, the Recall value is always 1.0. With this in mind, we can utilize some statistical measures to estimate the routing

efficiency:

- ***Prevalence*** – $[(TP + FN)/(TP + TN + FP + FN)]$
  This is the ratio of the number of URI-Rs in the sample set that are present in the archive to the total size of the sample set (as defined in Equation 6). Due to the sparse nature of web archives, this value is often very low. In our dataset it is less than 0.05 (Table 4).

- ***Recall*** – $[TP/(TP + FN)]$
  This value estimates how many URIs from the sample set that are present in the archive are actually routed to the archive by the profile (as defined in Equation 7). Since our dataset represents the complete knowledge of archives for a given time period and we do not set a threshold on the values of "frequency" and "spread", hence the Recall is always 1.0.

- ***Precision*** – $[TP/(TP + FP)]$
  This value estimates how many URIs from the sample set that are routed to the archive by the profile are actually present in the archive (as defined in Equation 8). Since the Prevalence value is low in our dataset, a small change in the percentage of FP affects the Precision significantly, hence the Precision may not be very effective to estimate the routing efficiency. In our dataset this value is below 0.35 for profiles ranging from *H1P0* to *HxP1*.

- ***Specificity*** – $[TN/(TN + FP)]$
  This value estimates how many URIs from the sample set that are not present in the archive are actually not routed to the archive by the profile (as defined in Equation 9). Since the Prevalence value is low in our dataset, this becomes the significant measure to estimate the routing efficiency as it tells how much unnecessary work a profile can avoid. In our dataset this value is above 0.5 and often close to 1.0 for any profiles other than *H1P0*.

- ***Accuracy*** – $[(TP + TN)/(TP + TN + FP + FN)]$
  This value estimates how many URIs from the sample set are correctly routed or not routed to the archive by the profile (as defined in Equation 10). The accuracy value combines the essence of the Precision and the Specificity. In our experiment, there are no false negatives, hence the Accuracy can be simplified as $(TP + TN)/(TP + TN + FP)$. Additionally, in our dataset, $TP \ll TN$ due to the low Prevalence, the

Accuracy can further be approximated to $\approx (TN)/(TN + FP)$ which is Specificity. As a result, due to the sparse nature of archives, the Accuracy is very close to the Specificity.

Figure 7 (Chapter 1) and Table 4 are good indicators of why archive profiles are useful: since all the three archives have <5% of any of the query sets, there would be many unnecessary queries if an aggregator simply broadcasted all queries to all archives. Essentially, broadcasting would yield a high number of false positives.

As illustrated in Figures 44a, 44c, and 44e, from all three archives it can be seen that the *Precision* grows when a profile with higher *Relative Cost* is chosen. Figure 44 shows that *DDom* profile doubles the precision as compared to the *TLD-only* profile and the *HxP1* profile brings five fold increment in the precision. These figures also show that inclusion of subdomain count does not affect the results much as there are not many domains that are utilizing more than one subdomain. Figures 44b, 44d, and 44f show that there is significant gain in *Precision* with little increment in *Relative Cost*. The difference in the trends of Figure 44 can be understood by means of the following factors:

- Small *Prevalence* values of the sample URIs (as shown in Table 4) affect the growth in *Precision* (on *y-axis*) from one profile to the next.

- Small *Prevalence* also makes the *Precision* a less stable measure of the routing efficiency as it can change significantly with a slight change in the percentage of false positives. This is why DMOZ sample has the highest *Precision* in UKWA (Figure 44d), but IAWayback and MementoProxy perform better in the other two archives (Figures 44b and 44f).

- The UKWA uses a shallow crawling policy which results in higher *Relative Cost* (ϕ). This increases the distance (on *x-axis*) from one profile to the next.

- The UKWayback sample is based on a recent Wayback access log of the UKWA. However, the UKWA dataset is older and may not completely reflect what the archive had at the time when the access log was captured. Also, an access log shows what clients are looking for in the archive rather than what the archive holds. For these reasons, the *Prevalence* of UKWayback is 0.001 for UKWA (i.e., <0.1% of the sample URI-Rs are archived in UKWA). Additionally, the UKWayback sample is biased towards `*.uk` URI-Rs that is not the primary focus of the other two archives, which is why the *Prevalence* is even smaller for the two archives (0.034% and 0.002%).

- The Stanford archive is a small and focused archive and <0.05% of any sample set is archived in it. Hence, this low *Prevalence* causes instability in the *Precision* and shows some unexpected behavior such as decreased *Precision* on increased *Relative Cost* (Figure 44f).

The precision value is useful when the intent is to know what is in the archive. However, from the Memento routing perspective with such a small prevalence value (<0.05) it becomes more important to know which URI-Rs not to route to an archive. For this purpose we measure the routing efficiency in terms of the specificity as illustrated in Figure 45. As illustrated in Figures 45a, 45c, and 45e, from all three archives it can be seen that the *Specificity* of the *H1P0* profile is very poor and it grows significantly when a profile with higher *Relative Cost* is chosen. Figures 45b, 45d, and 45f show the *Relative Cost* it takes to increase the *Specificity*.

To combine the essence of the precision and the specificity, we also measure the routing efficiency in terms of the accuracy as illustrated in Figure 46. As illustrated in Figures 46a, 46c, and 46e, from all three archives it can be seen that the *Accuracy* of the *H1P0* profile is very poor and it grows significantly when a profile with higher *Relative Cost* is chosen. Figures 46b, 46d, and 46f show the *Relative Cost* it takes to increase the *Accuracy*. We have explained earlier that due to the sparse nature of archives, the values of the specificity and the accuracy are almost the same as shown in Figures 45 and 46.

Figures 44, 45, and 46 also show that some profiling policies are quite similar in the *Relative Cost* and their corresponding *Precision*, *Specificity*, and *Accuracy* values. Based on this assessment, we grouped some policies together in Table 7 to provide a simpler cost and efficiency estimate. We took the average of all the values from different archives and samples in each group to estimate the *Specificity* and the *Accuracy* of each group. However, the *Precision* values were less stable (due to small prevalence), hence we provided a relative range instead.

A *URI-Key* profile is more robust than a *URIR* profile because it can predict the presence of resources in an archive that were added in the archive after the profile was generated. For example, if a new image named `Large-Logo.png` is added under `bbc.co.uk/images`, it is very likely that the archives that crawl BBC will capture the new image. The *URI-Key* profile illustrated in Figure 40 will be able to predict the presence of the new resource without any updates, but a *URIR* profile will need an update to include the new URI-R before it can predict the presence of the new image.

6.4 PROFILING THROUGH FULLTEXT SEARCH

If a web archive provides fulltext searching in its collection, it can be leveraged to discover the holdings of the archive. In this approach, we send some random query terms to the archive and collect the resulting URIs returned from the archive. Although with each successful response we learn some new URIs, the learning rate slows down as we go forward because of the fact that some of the URIs returned in a response may have been already seen in earlier queries. This follows Heaps' Law [97].

With carefully chosen values of various advanced search attributes and selection of a suitable list of keywords, we can affect the parameters of Heaps' Law and maximize the learning rate. In this section we analyze different fulltext approaches to discover holdings of an archive. We also explore different ways to perform the search when the language of the archive is unknown or a list of suitable keywords is not available for that language.

For our experiments we chose the Archive-It hosted North Carolina State Government Web Site Archive collection[2] which is the largest collection in Archive-It and provides fulltext search. Since we have Archive-It data (up to 2013) hosted in our dark archive at Old Dominion University (ODU), it was easier for us to perform the coverage analysis on this dataset.

We started our experiment by searching for stop-words. For example when we searched for the term "a" it returned 26M+ results, which is very close to the number of the HTML resources in the collection up till March 2013. However, each page only contains 20 results, hence, in order to learn all the 26M+ URIs we will have to make 1.3M+ HTTP GET requests. Our goal was to make as few requests as possible to learn a diverse set of URIs in the collection. Additionally, this approach cannot be generalized, as not all archives will behave the same on stop-words.

In the next step we built static and dynamic word lists and searched for those words as query terms. For each term we only record the URIs in the first resulting page and move on to the next word in the list. Having a configurable pagination would have benefited us by choosing a larger number of results per page, but Archive-It has it fixed to 20 results and does not allow any changes.

1. ***Top Words*** – We collected top the 2,000 English nouns[3] and used them as the query terms. We accumulated the first page results to plot the learning curve, but

[2] `https://www.archive-it.org/collections/194`
[3] `http://worddetail.org/most_common/nouns`

also extracted other information such as the result count for each search term. As expected, these top terms yielded high values for the result count for each terms. Additionally, each term yielded more than one page of results, hence no effort was unsuccessful. We also observed that the response time is correlated with the result count, so the same number of top terms would take longer to fetch than a random word list.

2. ***Random Linux Dictionary*** – We ran the same procedure on a randomly chosen set of 2,000 unique words from the built-in Linux dictionary. This time the average number of results per terms was lower, but there were many terms for which the collection returned no results or fewer than 20 (page size) results.

3. ***Dynamically Discovered Word List*** – The above two experiments were based on the static lists of words. This static word list approach requires the knowledge of the language (and field) of the collection to choose a static list of words for that language, which may not be easily available. Additionally, the list would be finite and may not be enough to discover a sufficient number of the collection holdings. To overcome this issue, we build a model (discussed in Section 6.4.1) that can dynamically discover new words from the searched pages and utilize those words to perform further searching. We introduced different policies in the dynamic discovery model based on how the new words are learned and how the next word for searching is chosen. We then analyzed the learning rate and the number of HTTP requests for each policy.

6.4.1 RANDOM SEARCHER MODEL

To perform searches on a static or dynamic word list, we developed a general *Random Searcher Model (RSM)* that can be configured to operate in one of the four modes with the help of configurable *Vocabulary Seeding* and *Word Selection* policies. To understand the model we discuss different data structures, policies, operation modes, and procedures separately then put them together to describe the overall working.

Data Structure

The RSM has the following data structures to hold various intermediate statistics and states:

1. ***Vocabulary*** – A data structure to hold the list of words to be searched. Depending on some policies it may or may not allow duplicates. This data structure should provide

the number of total or unique words in the list. The data structure should support functions to overwrite the list, add more words to the list, pop a random word from the list, and remove all occurrences of the popped word in the list.

2. ***SearchLog*** – A data structure to hold the record of each search attempt. It contains the searched word, attempt result (success or failure), and any additional meta information such as the response result count and newly discovered URIs. An implementation may choose to offload some results to a separate data structure or include more attributes for analysis. This data structure should provide the number of total or successful searches. The data structure should support functions to add new records, querying if a given word is present, and randomly selecting a successful searched word.

3. ***ResultBank*** – A dictionary like data structure to hold all the discovered URI-Rs and their respective memento counts. This data structure should provide the number of total URI-Rs (or dictionary keys). The data structure should support functions to add new records and iterate over all the records.

Policies

Policies control the behavior of the RSM. We have defined two different policies with various valid configuration values as described below:

1. ***Vocabulary Seeding Policy*** – This policy controls how the *Vocabulary* is seeded and how often. Valid values are:

    - *Static* – The *Vocabulary* is manually seeded in the beginning with a static list of search keywords. It allows seeding only once and the procedure terminates when all the seeded keywords are consumed.
    - *Progressive* – The *Vocabulary* is initialized with a few words, but it is aggressively overwritten after each successful search with the newly discovered words, except when there are no new words discovered.
    - *Conservative* – The *Vocabulary* is initialized with a few words and new words discovery is only performed when all the words from the *Vocabulary* are consumed. The *Vocabulary* is only reseeded when it is empty.

2. ***Word Selection Policy*** – This policy controls how search words are selected from the *Vocabulary*. Valid values are:

- *Popularity Biased* – Randomly select one word from the *Vocabulary* where the probability of a word being selected is proportional to the term frequency in the *Vocabulary* normalized by the total number of terms in the *Vocabulary*. A simple implementation of this policy would consider the *Vocabulary* as a bag of random words (with duplicates allowed) to select a random word from it. There are other memory efficient approaches possible, but this simple approach illustrates the concept naturally.
- *Equal Opportunity* – Randomly select one word from the list of unique words in the *Vocabulary*.

Operation Modes

An operation mode is a valid combination of *Vocabulary Seeding* and *Word Selection* policies (as mapped in Table 8). Not all combinations of the two policies are valid. We recognize the following four as valid modes of the *RSM*:

1. ***Static*** – This mode operates on a static word list that is known in advance so it does not have to make additional fetches to discover more words. However, finding a suitable word list for a given collection could be difficult.

2. ***PopularityBiased*** – In this mode the searcher randomly picks a URI from the returned result and fetches that page to discover new words. Once the words are fetched, it randomly picks a word from that and repeats the search operation. In this way the searcher has to fetch a page after every search attempt to search for the next word. Selection of the words is random, but the duplicates are not removed so the words with higher frequency in the page have higher chance of being selected.

3. ***EqualOpportunity*** – This mode works the same way as the *PopularityBiased* mode does, but it picks the next word from the unique list of words so a rare word on the page has the same probability of being selected as a high frequency one.

4. ***Conservative*** – The *PopularityBiased* and the *EqualOpportunity* modes have the drawback of being twice as costly in terms of number of HTTP requests as compared to a static word list of the same size. The reason for this is that after every successful search, there is a page fetch to learn new words. However, this *Conservative* mode does not throw away previously learned words and consumes each of them. It makes a page

Table 8. RSM Operation Mode Mapping With Policies

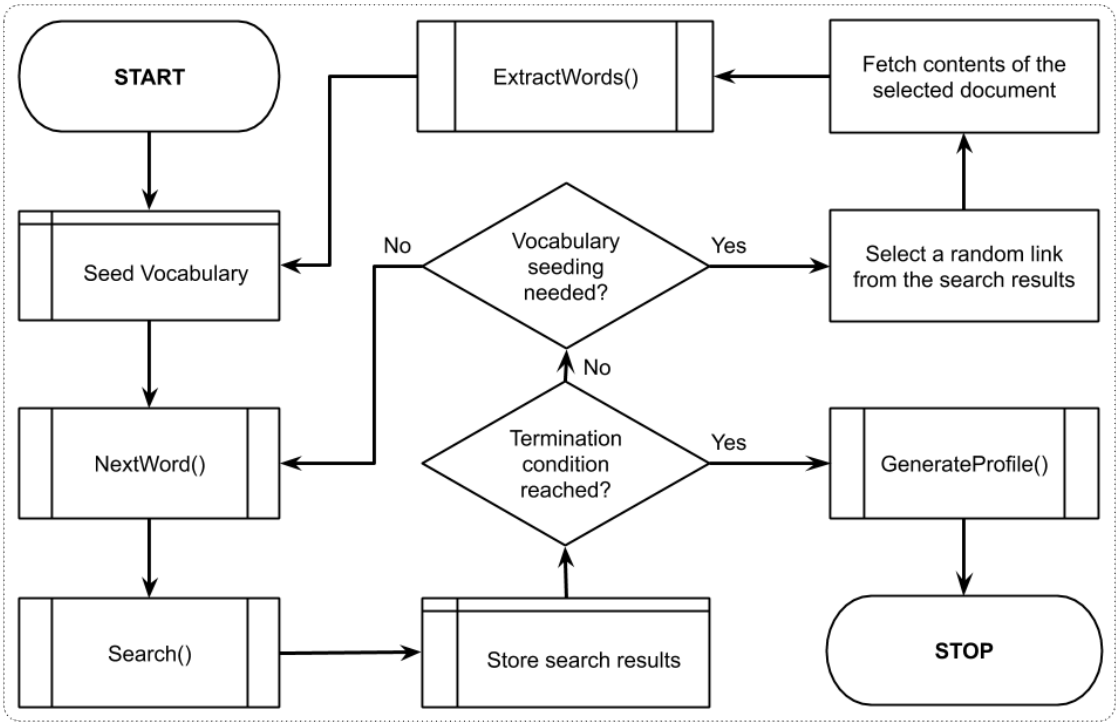
Operation Mode	**Vocabulary Seeding**	**Word Selection**
Static	Static	Equal Opportunity
PopularityBiased	Progressive	Popularity Biased
EqualOpportunity	Progressive	Equal Opportunity
Conservative	Conservative	Equal Opportunity

fetch to learn new words only when all the previously learned words are consumed. This reduces the number of HTTP requests significantly.

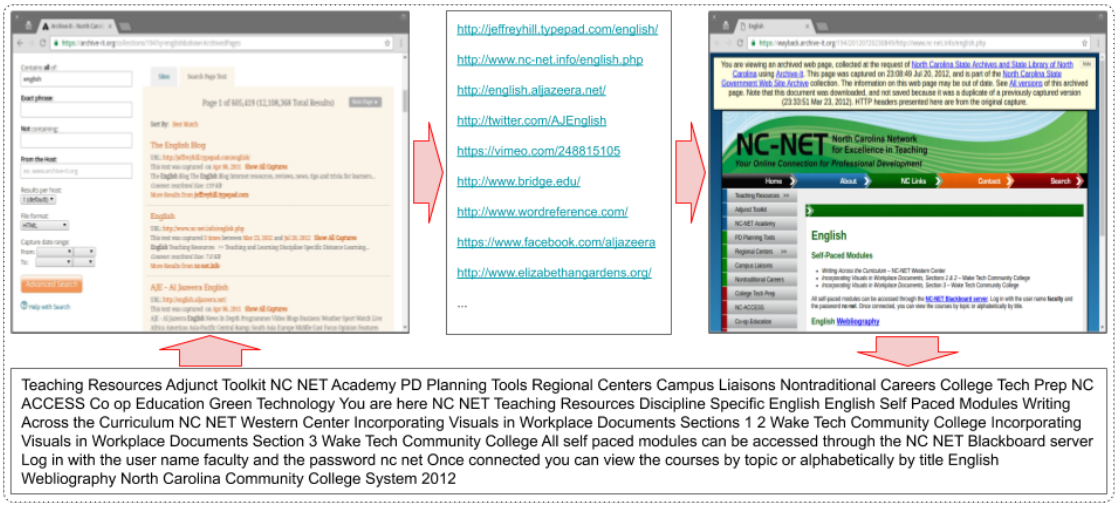
Procedures

The RSM has the following public procedures:

- ***Initialize()*** – This method initializes a Random Searcher instance with supplied configuration options and *Vocabulary* seed.
- ***TerminationCondition()*** – This method returns True if the conditions are met to terminate the *NextWord* iterator. It returns False otherwise.
- ***NextWord()*** – This method pops the next word from the *Vocabulary* based on the Word Selection Policy (and removes all the occurrences of the word in the *Vocabulary*, if any). If the *Vocabulary* is empty and the terminating condition is not met then it randomly picks a successful searched word from the *SearchLog*. This is a generator function that can act as an iterator until the configured termination condition is met.
- ***Search()*** – This method searches the collection for a given search keyword and populates the *SearchLog* and the *ResultBank* with appropriate values from the search result. Depending on the configured policies it may also select a URI from the result to extract words from it to seed the *Vocabulary*.
- ***ExtractWords()*** – This method fetches the page at the given URI, sanitizes it (by stripping off the markup, scripts, and styles), tokenizes the text to split in non-empty words, and returns the bag of words (duplicates included, if any) in the order they appeared in the document.



(a) Flowchart of the Random Searcher Model



(b) An Example Illustration of the Random Searcher Model

Fig. 47. Random Searcher Model Overview

- ***GenerateProfile()*** – This method iterates over all the items in the *ResultBank* and generates an *Archive Profile* based on the supplied profiling policy.

To run the *Random Searcher* an instance of the RSM is initialized with one of the four possible modes, some initial seed words, termination condition configuration, tokenizer pattern, and other configurations. Then the *NextWord* generator is iterated over to discover next word for searching until it hits the termination condition. For every word, the *Search* method is called which internally performs the collection lookup and updates various data structures of the RSM instance. Once the iterator terminates, *GenerateProfile* method is called to serialize collected statistics in an *Archive Profile*. An overview of the RSM is shown in Figure 47a as a flowchart diagram and in Figure 47b as a working example illustration.

6.4.2 IMPLEMENTATION

We implemented the RSM in Python and made the code available publicly [6]. However, due to the lack of a uniform search API across archives, the implementation is not generic enough to run on any archive. The page scraping part of the code that extracts useful pieces of information from the search result page and assembles them in a data structure needs custom implementation for each archive. The Archive-It search interface has an undocumented JSON API that we used to avoid HTML parsing. Our implementation has some additional intermediate data structures and logging in place for the sake of analysis which is not needed for production purposes.

6.4.3 RSM EVALUATION

To evaluate the RSM we estimated the search cost of different operation modes for discovering certain portion of the archive holdings in terms of a given profiling policy. To evaluate generated archive profiles we measured how much of the archive holdings we must discover in order to gain a satisfactory Recall and corresponding routing efficiency for a given profiling policy. In this analysis we have used four different profiling policies. Examples are derived from the URI `https://www.news.BBC.co.uk/Images/Logo.png?width=80&height=40`.

- ***H1P0*** – Only TLDs are used as keys (212 unique keys in the collection). E.g., `uk)/`.
- ***DDom*** – Only registered domain name is used as keys (91,629 unique keys in the collection). E.g., `uk,co,bbc)/`.
- ***HxP1*** – All host segments and one path segment are used as keys (1,724,284 unique keys in the collection). E.g., `uk,co,bbc,news)/Images`.

Table 9. Cost Comparison of RSM Operating Modes

Operation Mode	**Query Cost**	**HTTP Cost**
Static	C	C
PopularityBiased	C	$2C$
EqualOpportunity	C	$2C$
Conservative	C	$C + \delta$ (where $\delta \ll C$)

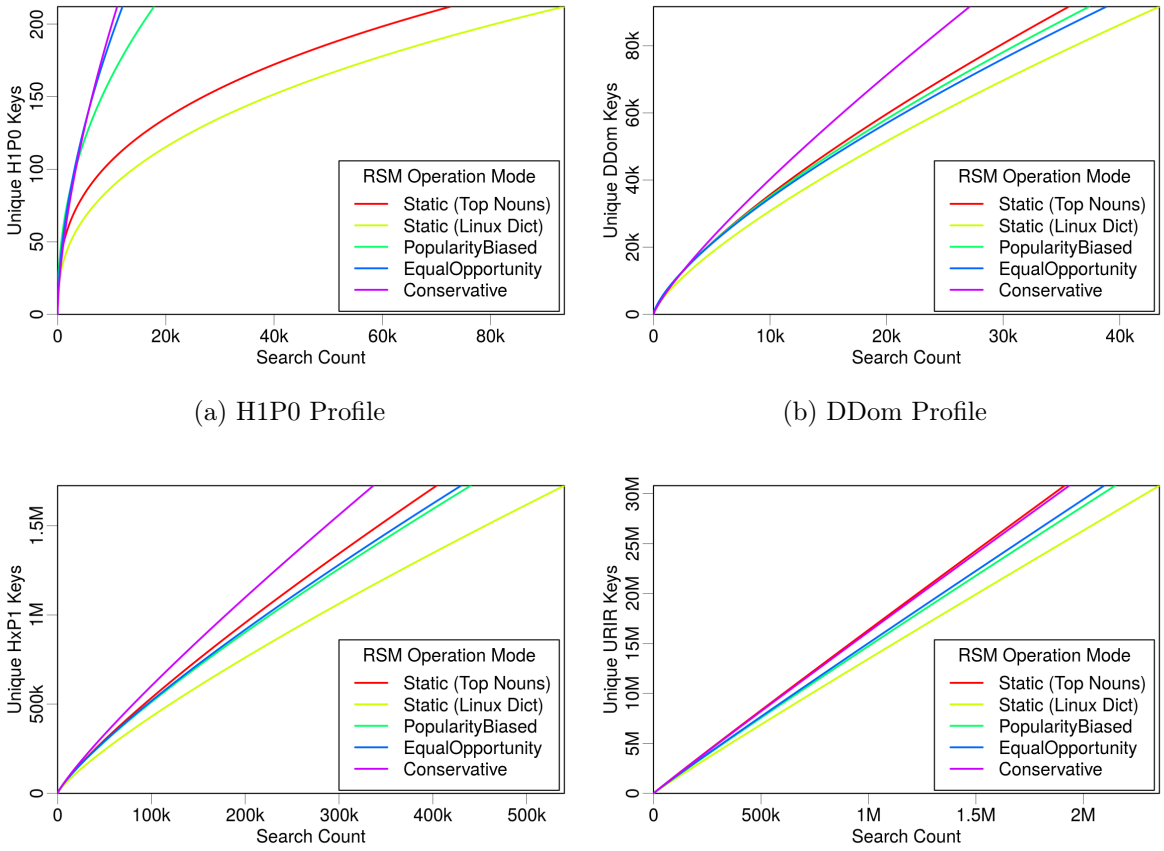
- ***URIR*** – SURTed URI-Rs are used as keys (30,800,406 unique keys in the collection). E.g., `uk,co,bbc,news)/Images/Logo.png?height=80&width=200`.

Estimating Searches Needed

Figure 48 illustrates the projected estimate of the search cost for each of the four profiling policies based on the initial 2,000 searches. Each graph is extended up to the 100% limit of each profiling policy on the y-axis. For example, there are total 212 unique TLDs in the collection, hence the upper y-axis limit in Figure 48a is set to 212. The RSM operation modes in which an additional HTTP request is made to fetch a Memento to extract words causes additional HTTP GET overhead. Table 9 shows the HTTP cost of each RSM operation mode with respect to the corresponding query cost. The value of δ is quite small as compared to C. In our experiments we found that for $C = 2,000$ the value of δ was 8. Which means for making 2,000 queries we only needed eight additional HTTP GETs to extract new words in *Conservative* mode. The *Conservative* mode is an overall winner. It shows a higher learning rate and does not require a static list of words to be supplied. It is also good because it does not have much overhead HTTP request cost for the term discovery.

Profile Routing Efficiency

Unlike CDX analysis, a fulltext search based archive profile will often be created based on partial knowledge of the archive holdings. This may cause incomplete Recall in Memento routing, which means that an archive might have some Mementos unknown to the profile and as a result it may not have a matching key in it to route the query there. Hence, it is important to analyze the incremental changes in the confusion matrix as we know more and more of the archive's holdings. Table 6 illustrates the confusion matrix in the Memento routing context. For this analysis we selected the first ten years of UKWA dataset as the gold



(a) H1P0 Profile

(b) DDom Profile

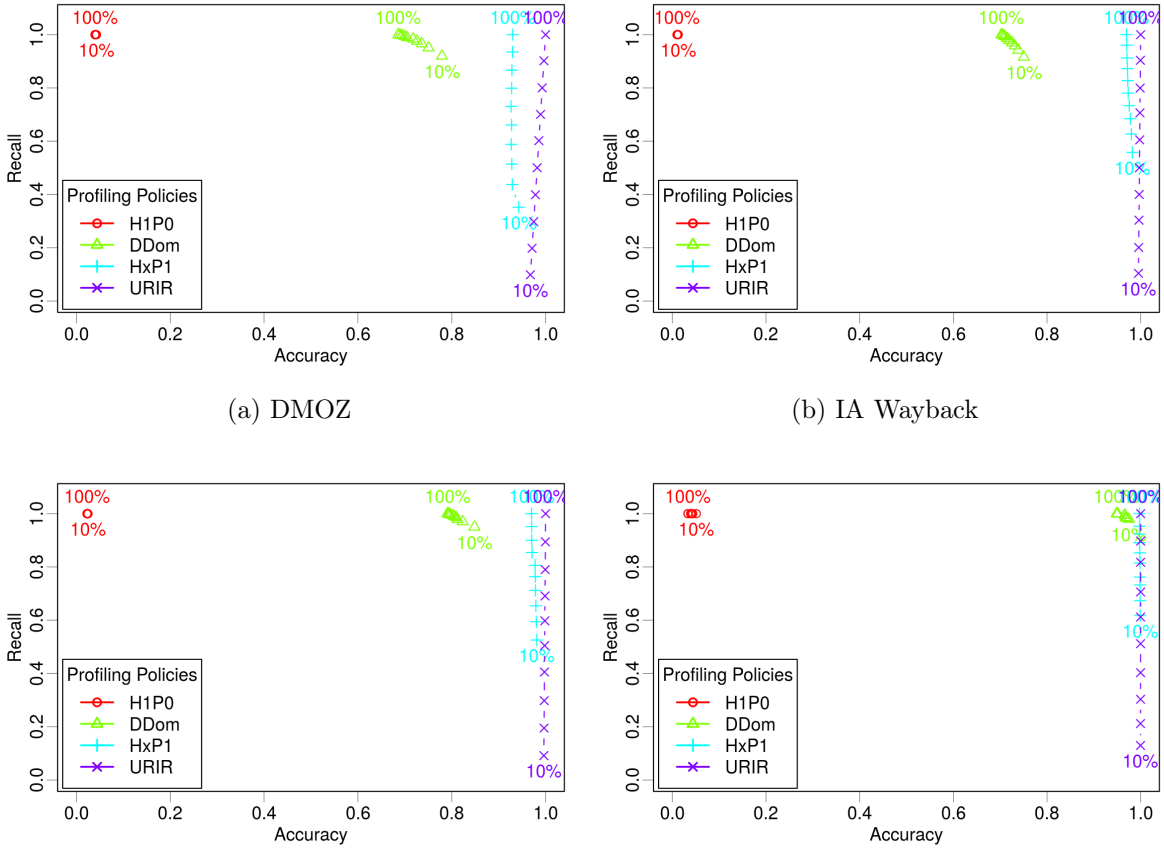
(c) HxP1 Profile

(d) URIR Profile

Fig. 48. Searches Needed vs. Required Coverage

dataset, then we extracted the unique URI-Rs from it and randomized it. The randomized URI-R list was then split into ten equal chunks. We then profiled these chunks incrementally using different policies to see how these policies perform against a sample set of query URIs when we know only 10%, 20%, 30%, ... to 100% of the archive. We repeated this process for four different query URI sample sets of one million each. In each case we calculated the confusion matrix and plotted Accuracy vs. Recall graph to estimate the routing efficiency. Accuracy here means how often an archive profile is correct in routing or not routing a query to the archive.

Figure 49 shows that if the complete archive is known, we should choose higher order profiles such as HxP1. The HxP1 profile has Accuracy almost as good as the URIR profile. The cost of the HxP1 profile is less than one sixth of the cost of the URIR profile.



(a) DMOZ

(b) IA Wayback

(c) Memento Proxy

(d) UKWA Wayback

Fig. 49. Incremental Accuracy vs. Recall as a Function of Archive Knowledge

Additionally, the HxP1 profile has higher Recall than the URIR profile when the complete archive is not known. However, since we cannot afford to lose much of the Recall, we favor smaller profiles such as DDom when we have partial access to the archive (such as when using fulltext search). The DDom profile (that has less than 1% cost as compared to the URIR profile) can have about 0.9 Recall while correctly routing (or not routing) more than 80% of the URIs by only knowing 10% of the archive. Cases where only a tiny fraction of the archive is known by sampling (such as when neither CDX is available nor fulltext search), we favor the smallest profile H1P0/TLD-only to maintain an acceptable Recall value. Even though the URIR profile always yields 100% Accuracy, it suffers from poor Recall. With the complete CDX accessible, the URIR policy costs a lot compared to other policies. Additionally, the URIR policy does not have any predictive powers for unseen URIs, hence

maintaining the freshness of the profile is challenging as the archives acquire more URI-Rs.

6.5 CHAPTER SUMMARY

In this chapter we outlined four approaches of learning an archive's holdings: 1) CDX Profiling, 2) Fulltext Search Profiling, 3) Sample URI Profiling, and 4) Response Cache Profiling. We performed exhaustive analysis of the first two approaches, but left the other two as those were explored by other researchers already. We briefly described the data structure of archive profiles, defined the term *URI-Key*, and outlined numerous profiling policies with varying levels of details. We described and illustrated the *URI-Key* generation method for each profiling policy.

For CDX summarization we described various datasets that we collected. We described various relevant statistical measures for evaluation. Then we evaluated the overlap among archives, growth of profiles with different policies, and their cost vs. routing efficiency. We found that the growth of the profile with respect to the growth of the archive follows Heaps' law, but the values of free parameters are archive-dependent. With accuracy defined as correctly predicting that the requested URI-R is present or not present in the archive, we gained about 78% routing accuracy with less than 1% relative cost and 94% routing accuracy with less than 10% relative cost without any false negatives. The registered domain profile doubles the routing precision with respect to the *TLD-only* profile, while a profile with complete hostname and one path segment gives ten fold routing precision.

For fulltext search profiling we described another set of datasets. We then introduced our contribution called *Random Searcher Model*. We described its data structure, policies, operation modes, procedures, and reference implementation. For evaluation we selected four distinct and diverse profiling policies. Then we evaluated number of search operations needed to discover certain fraction of holdings of an archive and the cost of these searches. Finally, we evaluated the routing efficiency of profiles created using *Random Searcher Model*. We concluded that the DDom profile (that has less than 1% cost as compared to the URIR profile) can have about 0.9 Recall while correctly routing (or not routing) more than 80% of the URIs by only knowing 10% of the archive.

CHAPTER 7

LEARNING ARCHIVAL VOIDS

In order to not route lookup requests to the archives that are not likely to return good results, it can be useful to learn about the voids of various archives. This chapter addresses our first research question, "***RQ1**: How to learn about the holdings and voids of an archive?*". In this chapter we only explore the second part of this question that concerns how to learn about the voids of an archive.

In Chapter 6 we explored various means to identify and summarize resources that an archive holds. Knowing about the summary of holdings of an archive allowed us to improve Memento routing accuracy without any false negatives. In this chapter we explore various means to identify resources that are not present in an archival collection or a web archive is not willing to serve. This will further improve the routing accuracy by reducing false positives.

7.1 INTRODUCTION AND MOTIVATION

We introduce the term *Archival Voids* to describe what is missing from a web archive as opposed to *Archival Holdings* that describe what a web archive holds. This can be defined as a function that takes two arguments, a *URI Key* and a web archive, and returns the measure of the archival void under the given URI scope (e.g., a TLD, a domain, or a domain name with a path prefix) in the archive. However, a reliable estimate of a void required knowing the set of URIs under the scope that ever existed and the set of URIs under that scope that are present in the archive. Knowing the cardinalities and overlap or difference of these sets is often not practical as it might require crawling a whole domain or TLD. However, in some cases it is possible to estimate the archival voids. For example, a webmaster who knows all the URIs of a website with finite resources can query a web archives to know how many of those resources are present in or absent from the archive. In some cases, it might be possible to estimate the number of URIs in scope by querying a public search engine and extracting the number of hits, but this number may not be reliable as it may not contain historical pages or non-textual resources. That said, in this work we are generating an archival voids profile based on what a web archive knows, so in this case we will only report portions of the web that have zero resources archived/accessible and are requested frequently.

"Why do we care about archival voids?" This is an obvious question to ask, especially after knowing what is present in an archive. One might argue that if we already know what is present in an archive then everything else can be considered to be missing from the archive. This statement will be true if we had a complete knowledge profile (which we already discussed in Chapter 6 that it is not practical and has its own issues when it comes to freshness). On the contrary, if we had an archival holdings profile based on URI sampling then we may not have an accurate knowledge of what is present in the archive, hence we cannot deduce what is not present in it. Similarly, when we have a summarized profile (using one of the profiling policies discussed in Chapter 6), we may conclude many URIs to be present in an archive, but they might be absent from it (i.e., false positives).

To understand this, lets assume that an archive holds resources at paths "`/a/1`", "`/a/2`", "`/a/3`", "`/b/1`", and "`/b/2`" under "`example.com`" domain. This needs five different keys in the profile to describe these holdings, but we can summarize it as "`com,example)/a/*`" and "`com,example)/b/*`" (here we are using wildcard character to illustrate that we have all variations at the path depth 2). While this summary ensures that we do not assume "`/c/1`" is present, it does suggest that "`/a/1/z`", "`/a/4`" and "`/b/3`" (and many others) are present. If we could list or summarize resources that applications might be interested in, but are not present in the archive, we can further improve the accuracy by reducing false positives.

An aggressively summarized archival holdings profile improves the freshness, but inherently introduces many false positives. An archival voids profile can compensate for that by identifying those false positives and explicitly denying their presence in the archive. This means an archival holdings profile and an archival voids profile can work together as opposing forces to find the sweet spot for an increased routing accuracy while minimizing the profile size and maximizing freshness.

An archival voids profile has some use cases beyond Memento routing. For example, an archive can identify voids in its collections to crawl those resources and fill the cavities while those resources are still live on the web. Another use case could be public disclosure of resources that an archive does not want to collect/serve due to their collection policies. Moreover, it can be helpful in coordinating with other archives, like IIPC members do. For example, if an archive has a void in a specific URI space, but another archive has holdings for the same, then they have complementary holdings.

7.2 SOURCES OF TRUTH

The URI space is infinite and the web is vast. Many people have attempted to estimate

the size of the web at different times and have come up with different numbers from a few billions to a few trillions (as discussed in Section 3.1 of Chapter 3). However, knowing the size of the web and web archive holdings can only lead us to estimating how much of the web is not archived. If we want to know what sections of the web are not archived, we need to know all the existing URIs, not just their number. Knowing URIs of all the existing resources on the web or creating a representative sample of the web is hard. However, we can sample URIs from certain sources (e.g., DMOZ, social media, or access logs) that are of interest for a specific application, while knowing that these samples will have their own purposes and biases. We can create archive profiles of archival voids in the following ways:

- Perform lookups of sample URIs in an archive and record all the URIs that are not archived.
- Use access logs of a Memento aggregator or the archive itself to identify resources that are absent from an archive.
- Use URLs from the access control lists (ACL), approved take down requests, resources blocked by `robots.txt`, and domains/TLDs blocked by an archive's policy.

URIs collected by the means listed above can be summarized to form archival voids profile. In the previous chapter we described Random Searcher Model (RSM) (Section 6.4.1) to learn about the holdings of an archive using fulltext search. However, fulltext searching is not a suitable technique for archival void detection because it only returns resources that are present in an archive. In this chapter, we only investigate archival voids profiling using access logs of an archive. Other approaches are either inefficient or beyond our abilities (e.g., we do not have access to archiving policies or ACLs of any web archive).

7.3 EVALUATION

To evaluate our process for estimating archival voids, we use access logs of a web archive. We extract URI-Rs from the access logs and identify URI-Rs that have always returned "`404 Not Found`" responses (ignoring any "`3xx`" or "`5xx`" responses). Then we filter URI-Rs out that are not accessed frequently, so that we profile only the popular resources. Figure 68 (in Chapter 8) shows that there are many URIs that are accessed frequently, but are not archived. Being able to summarize them can improve routing efficiency.

Table 10. Arquivo.pt Access Logs Summary

Feature	**Value**
Number of files (1 file per day)	2,220
Total size	461G
Total size (GZipped)	37G
Total lines (requests)	1,647,573,303
Logs start date	2013-12-02
Logs end date	2019-12-31
Missing date (filled with an empty file)	2016-09-08
Memento support start date	2016-06-03
Log configuration changed (a field added)	2019-09-17
Major replay system upgrade (fixed many issues)	2019-11-18
TimeMap endpoint changed (`/timemap/*/` → `/timemap/link/`)	2019-11-18

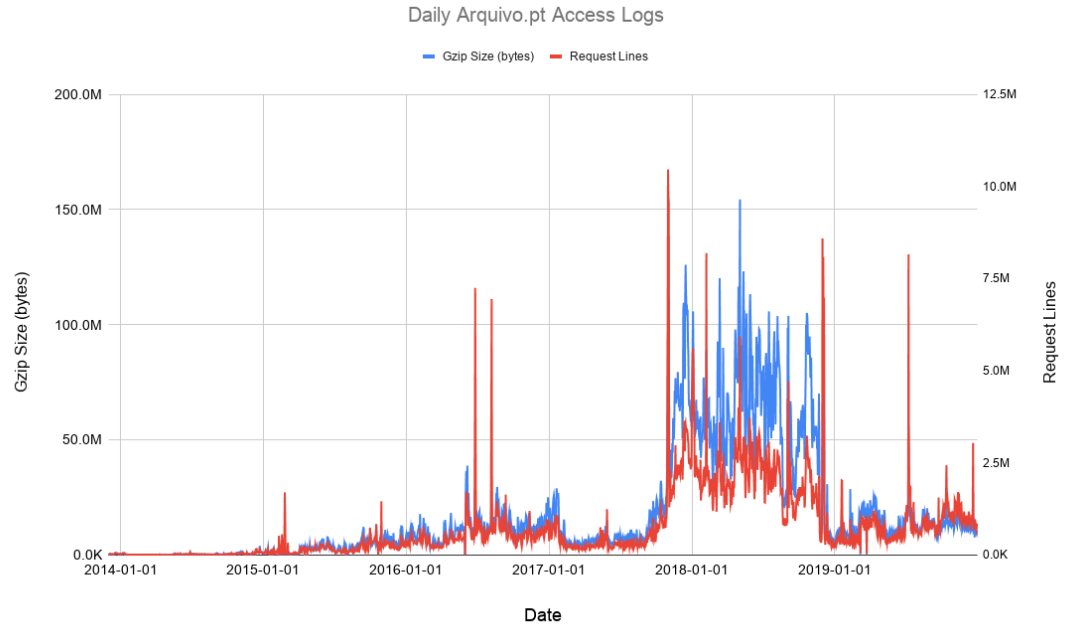
7.3.1 ACCESS LOGS DATASET

With the generous support from `Arquivo.pt`, we have access to over six years of their web archives' access logs. Table 10 summarizes the access logs data we acquired. These log files contain about 1.6 billion records, but not all of these records are Memento related. Memento support was added to Arquivo.pt in June 2016. To analyze these access logs we created an access log parser with unique features for web archive access logs (as described in Section 5.5 of Chapter 5).

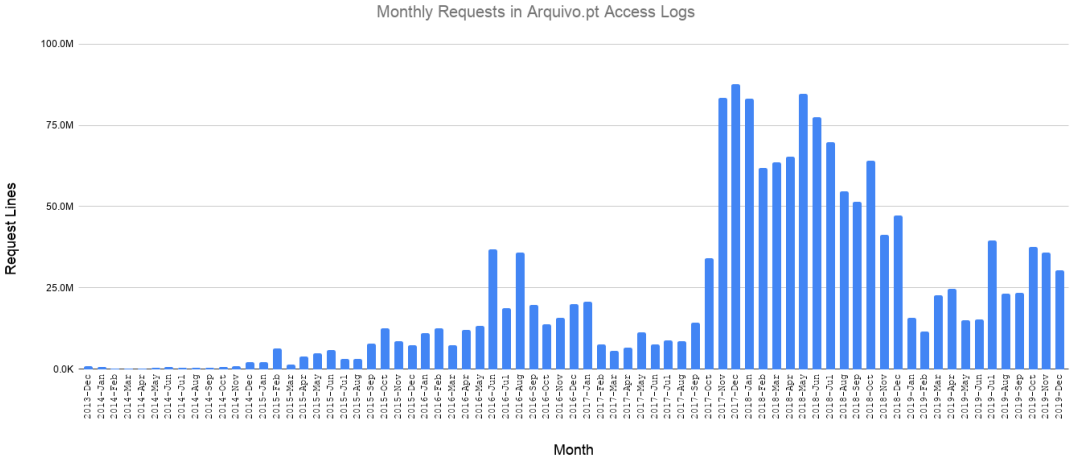
7.3.2 ACCESS PATTERNS

Figure 50 illustrates daily and monthly access patterns of Arquivo.pt. There is a significant increase in traffic for last few months of a 2017 and for the most part of 2018. On further investigation on user-agents and status code distribution, we found that this increase in traffic was primarily from Googlebot and a small portion of it was coming from YandexBot. Together, these two bots were responsible for over 80% of the traffic.

On November 23, 2018, Arquivo.pt updated their `robots.txt` file to exclude all the bots from accessing their resources under "`/wayback`" path under which their archival replay operates. In Figure 51 we illustrate two versions of their `robots.txt` from the same day, on which the latter shows corresponding change made to the file. The timing of this change



(a) Daily Arquivo.pt Access Log Records



(b) Monthly Arquivo.pt Access Log Records

Fig. 50. Access Patterns in Six Years of Daily Log Files of Arquivo.pt

corresponds to the drop in traffic coming from search engine bots.

Furthermore, we noticed an increased bot activity in 2019 that attempt to access Arquivo.pt's `robots.txt` file many times every second. These requests are coming from many

```
$ curl https://web.archive.org/web/20181123104043id_/https://arquivo.pt/robots.txt
User-agent: Arquivo-web-crawler
Disallow: /wayback

User-agent: *
Disallow: /nutchwax/search
Disallow: /search

$ curl https://web.archive.org/web/20181123125853id_/https://arquivo.pt/robots.txt
User-agent: Arquivo-web-crawler
Disallow: /wayback
Disallow: /noFrame/replay

User-agent: *
Disallow: /wayback
Disallow: /noFrame/replay
Disallow: /nutchwax/search
Disallow: /search
```

Fig. 51. Arquivo.pt Excluded Bots from Accessing Its Archival Replay on November 23, 2018

different locations, and some of those hosts belong to Google. They all have the same request signature (i.e., the same request URI, user-agent, and referrer). While we do not fully know the purpose and origin of these requests yet, it does not concern us much because the requests are not about mementos or their replay system, so they are out of our scope for this work.

Table 11 shows most frequent URIs that were accessed at least 10,000 times from Arquivo.pt over the period of six years. While their own domain `fccn.pt` and some other globally popular websites are present in this list, the high frequency of some less obvious resources suggest that they are perhaps coming from some browser add-ons or some pages that some people/tools open often where these resources are embedded.

Table 12 describes how often resources from various TLDs were accessed from the archive each year. The top five TLDs include "`.pt`", "`.com`", "`.org`", "`.net`", and "`.eu`". When preparing these statistics, we removed any TLDs that did not appear in all years as they were insignificant, and often malformed entries. This table only shows statistics on requests that are related to a memento (i.e., they have a URI-R in their path). Such requests can be URI-Ms, URI-Gs, or URI-Ts. The grand total is a little over one billion requests, which is two thirds of the total number of requests in their logs. Numbers under the 2018 column are larger than other years due to increased activity from search engine bots in the year 2018. The "`.au`" TLD shows an interesting trend as it was not as popular as some of the

Table 11. Most Frequently Accessed Resources from Arquivo.pt

URI	Count
fccn.pt/	102,953
google.com/	44,673
youtube.com/	29,418
facebook.com/	16,778
connect.facebook.net/en_us/sdk.js	16,462
discovery.dundee.ac.uk/admin/.../editor/contributiontojournaleditor.xhtml	14.608
tripadvisor.com.tr/cookiepingback?early=true	13,556
publico.pt/	13,022
lamonitor.com/	11,901
static.tacdn.com/js3/src/modules/component/bounceusertracking-v21915390943b.js	11,781
static.tacdn.com/js3/src/modules/component/bounceusertracking-v21915390943a.js	11,041
youtube.com/watch	10,563

other TLDs below it, but search engine bots seemed more interested in it in the year 2018, which made it go significantly up in the table. This suggests the need for periodic updates of archival voids profiles as the demand of certain sections of the web changes over time.

Considering TimeMaps, being one the most accessed resources by Memento aggregators, we decided to see what other user-agents are interested in them. There were about 42 million TimeMap requests in their logs. We found that LANL's TimeTravel Service is the largest source of traffic to Arquivo.pt's TimeMap endpoint. The first few months after Arquivo.pt added Memento support LANL's aggregator was making a significant number of requests, but it slowed down after a few months. For the first few months LANL was using an old Memento aggregator written in Java, which was later replaced by a new code that utilizes a classifier and an improved caching stack. The increased traffic during the first few months was likely caused by the cache front-loading and data collection for training the classifier. The second source of traffic was our own MemGator instance, running at Old Dominion University. There is a spike in July 2018, from our service, because someone used our service to access TimeMaps of a long list of URIs, which changed our regular usage pattern significantly. OldWeb.today uses our MemGator tool on its own servers to reconstruct Mementos in old browsers, which was another significant source of traffic to TimeMaps. Other sources include a variety of user-agents, often pointing to cURL, HTTP libraries in different languages, or research projects. Two notable user-agents among them which caused increased traffic on certain months pointed to an in-house script of Arquivo.pt

Table 12. Yearly Access Frequency of Top TLDs in Arquivo.pt

TLD	2014	2015	2016	2017	2018	2019	Total
.pt	1,769,211	5,638,317	42,105,411	139,685,874	455,617,209	75,585,184	720,401,206
.com	188,937	1,041,797	12,113,651	84,960,808	122,007,623	26,912,173	247,224,989
.org	14,594	76,424	1,106,144	5,383,964	35,025,928	2,411,604	44,018,658
.net	17,035	74,222	167,770	7,421,125	18,903,068	2,938,864	29,522,084
.eu	1,089	19,539	388,764	2,727,154	20,129,910	1,346,343	24,612,799
.au	10	189	815	30,275	4,147,213	92,882	4,271,384
.gov	172	3,258	16,502	485,699	1,930,423	479,466	2,915,520
.uk	5,061	3,920	14,400	391,839	1,364,946	343,796	2,123,962
.edu	627	5,945	16,230	192,558	1,385,982	291,992	1,893,334
.br	5,901	39,667	329,656	132,944	1,030,141	266,208	1,804,517
.ru	442	637	2,666	113,716	1,179,413	95,298	1,392,172
.de	564	2,613	22,047	143,661	737,228	444,187	1,350,300
.io	9	1,501	24,598	46,006	1,150,983	79,497	1,302,594
.pl	2	743	5,787	61,107	1,071,524	116,270	1,255,433
.int	160	894	2,617	97,603	731,551	204,840	1,037,665
.fr	149	3,009	19,283	142,771	644,848	195,730	1,005,790
OTHERS	34,717	125,950	171,235	721,084	3,335,357	1,441,533	5,829,876
ALL	2,038,680	7,038,625	56,507,576	242,738,188	670,393,347	113,245,867	1,091,962,283

(which we believe they use for periodic service quality/health check) and a MediaWiki bot, called WaybackMedic [79], that fixes broken links.

7.3.3 SOFT-404 TIMEMAPS

To build an archival voids profile from access logs, it is important to isolate records that have never been "`200 OK`". When we tried to see the distribution of TimeMap responses over different status codes, we found an insignificant number of "`404`s", except in the last two months. This was counter intuitive because our MemGator logs suggest that in more than 96% requests Arquivo.pt returns no mementos in its TimeMap as shown in Table 20 (Chapter 8). After further investigation, we found that their old replay system had bugs, causing it to return Soft-404 TimeMaps (as described in Section 2.1.4 of Chapter 2), until it was upgraded on November 18, 2019.

Now we had two choices, either profile only the last six weeks of data or somehow identify Soft-404 responses. Logs do not contain the response body, so we could not do much about

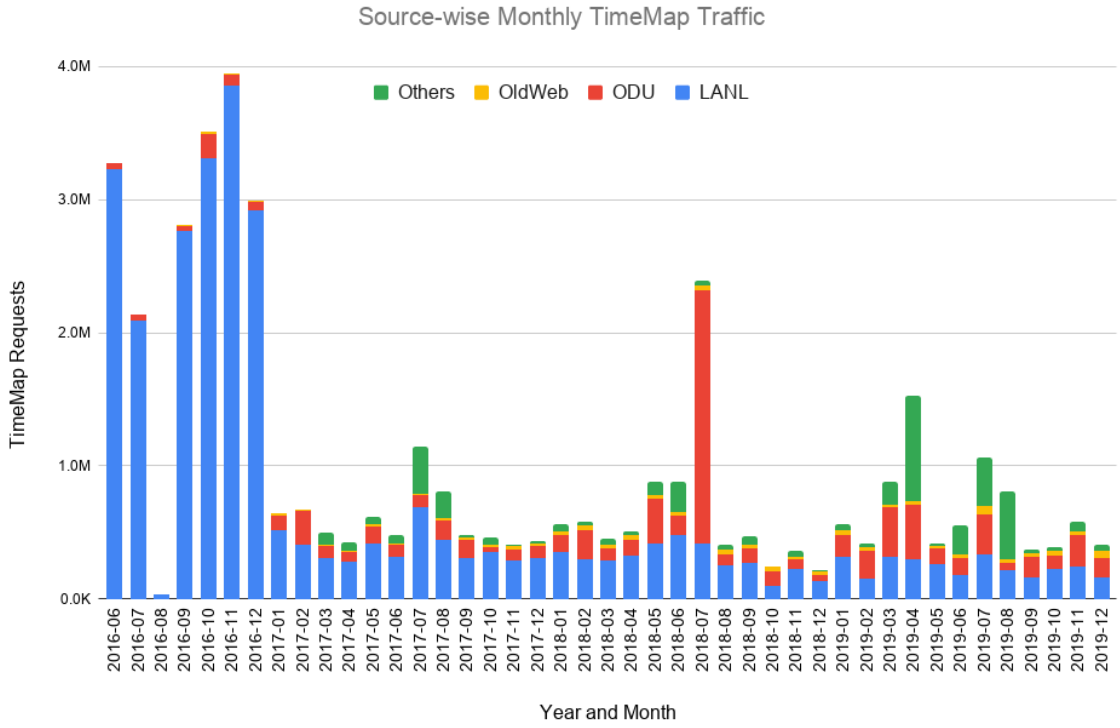


Fig. 52. Monthly TimeMap Access of Arquivo.pt from Various Sources

classifying responses. However, these access logs contain number bytes they returned in each response. We could think of two possibilities of what their TimeMap response might have been for resources they do not have any memento of: 1) there could be a plain message saying something like "the resource is not found", or 2) the response included the URI-R one or more times along with some other template body. In the first case, number of bytes will be exactly the same for all the failed TimeMap requests, but we did not see a single byte size over-represented. In the second case, number of bytes in response will be a linear function of the size of the request URI as shown in Equation 11. In this equation K represents the number of times URI-R appeared in the response and C is the constant size of the template body.

$$\text{Response Bytes} = K * \text{URI-R Size} + C \tag{11}$$

To investigate our hypothesis we checked for TimeMap requests in December 2019 access logs (when the Soft-404 issue was fixed) to find resources that are consistently returning

Table 13. Soft-404 TimeMap Response Bytes

Timestamp	Request	Bytes
1546885931	/wayback/timemap/*/http://matkelly.com/wail	222
1546885968	/wayback/timemap/*/http://matkelly.com/wail/	225
1547238957	/wayback/timemap/*/http://matkelly.com/wail	222
1547239466	/wayback/timemap/*/http://matkelly.com/wail	222
1547239877	/wayback/timemap/*/http://matkelly.com/wail	222

```
1 <https://arquivo.pt/wayback/timemap/*/http://matkelly.com/wail>;
  ↪  rel="self"; type="application/link-format",
2 <https://arquivo.pt/wayback/http://matkelly.com/wail>; rel="timegate",
3 <http://matkelly.com/wail>; rel="original"
```

Fig. 53. A Potential Soft-404 TimeMap

404s and checked responses corresponding to them in the past logs. Table 13 shows Soft-404 records of one such resource. In this table all the rows have a consistent number of bytes (222) except the second one (225). However, the second row also has a trailing forward slash in its URI-R, which is missing from the other rows. This was a clear indication that the URI-R was repeated three times in the Soft-404 response body, which increased the byte size of the response by 3 when only one extra character was added to the URI-R. Now, we knew the value of $K = 3$ and the size of URI-R; using Equation 11 we can compute $C = 150$. With this insight, we tried it on some other requests and found it working, which gave us more confidence.

In Figure 53 we tried to reconstruct what the Soft-404 response might have been, which matches our calculated numbers and looks like a reasonable representation. From this response we think we know the nature of the bug in their code. They, were perhaps not checking for the existence of any mementos for a given URI-R before generating the response. Instead they were creating the obvious initial lines of the response and then looping over all the mementos, which would loop zero times if there were no mementos and only the initial lines will be returned.

With this ability to identify Soft-404s reliably, we went through all the TimeMap requests

Table 14. Status Code Distribution of TimeMaps in Arquivo.pt Access Logs

Status	**Requests**
200	2,614,615
301	2,455
302	224,535
400	98,267
404	38,615,290
429	42,720
500	134,858
503	1,015
TOTAL	41,733,755

and fixed the status code ("`404`" for "`200`") in a copy of logs. We used these amended logs for further analysis.

7.3.4 STATUS CODE CHANGES OVER TIME

To ensure that we only profile URIs that never returned a successful response (after amending Soft-404s), we decided to investigate how often URIs change from one status code to the other. In the case of a URI-M there are many status codes possible, both due to observed status codes from the origin and the state of the replay server. However, in the case of a TimeMap we anticipated a limited number of different status codes. Actual status code distribution of TimeMaps is shown in Table 14. If there are no mementos for the given URI-R, the status should be `404`, otherwise `200`. In rare cases we expect `5xx` status codes, in case the server is facing any issues. However, the last two months of data had many `302` responses as well. On further investigation we found that when Arquivo.pt upgraded their replay system, they also changed some of their service endpoints (in this case, their TimeMap changed from `/timemap/*/` to `/timemap/link/`), for which they put redirects in place. In addition to this, they also had a few `301` responses for certain TimeMaps where the URI-R contained Facebook's tracking token in the query parameter, which they redirected to a URI-R without the tracking token. After knowing this, we removed all the redirect responses because they were not adding anything to our assessment of the popularity or unavailability of resources.

After this cleanup we sorted entries primarily on their canonical representation and a secondary sort on their time, so that we can know how each URI changed from one status to the other. We were expecting that a few resources that were 404 before would become 200 when they are eventually crawled and are made available and a few resources might go the other way if they are taken down for some reason. Other status code transformations were expected to be less likely (e.g., the server returning 5xx response occasionally).

However, when we analyzed our data, we found that there were many fluctuations between 200 and 404, where some resources changed their status codes back and forth hundreds of times. It turned out that it was caused by lack of proper URI normalization/canonicalization (see Section 2.6.1 of Chapter 2). For example, when a TimeMap was requested for apple.com they returned 200, but for Apple.com or APPLE.COM they returned 404 instead. We thought about a few approaches to amend this effect as well, but that could change our result in ways that can be harmful, so we decided to exclude all the requests that include any upper case letter in the hostname portion of their URI-R and started over. After excluding entries with any upper case letter in their hostname the number of fluctuations went down, but there were still many entries with hundreds of fluctuations back and forth between 200 and 404. We concluded that lack of URI canonicalization was not limited to just hostnames in Arquivo.pt, but perhaps they had little to no canonicalization in place.

After that we decided to work with the dataset without any canonicalization or filtering, considering each URI-R in the logs as an independent resource. This means we will have many non-canonical URIs that will always report 404 while their corresponding canonical version may or may not behave the same way. This may increase the size of our void profiles, but we expect the prevalence of many unique non-canonical URI-Rs to be small, which may fall below the threshold to be included.

Table 15 summarizes status code fluctuations of URI-Rs in the access logs of Arquivo.pt. There are 15,502,081 unique non-canonical URIs that have always returned the 404 status code and 680,328 URIs have always returned the 200 status code. We believe that the number of URIs returning 200 status code would have been a little larger and 404 status code a little smaller if Arquivo.pt were to exercise URI canonicalization from the beginning. There were 36,447 URIs that returned 404 status code in the past, but later started to return 200 while there were only 685 URIs that gone from 200 to 404. These numbers confirm our intuition about more URIs becoming available over time while a few of the existing resources disappearing (for example, blocked or taken down after reports or policy reviews). This table does not reflect how many times and for how long certain status codes

Table 15. Status Code Fluctuations of URI-Rs in Arquivo.pt Access Logs

Status Codes Over Time	**URI-Rs**
404	15,502,081
200	680,328
404,200	36,447
200,404	685
200,404,200	648
404,200,404	48
404,200,404,200	43
200,404,200,404,200++	40

remained associated with a given URI.

While analyzing data for status code fluctuations without any URI canonicalization we found that one specific URI was still exhibiting about 150 fluctuations back and forth between `200` and `404` status codes. On further investigation we found that it was `http://www.fccn.pt/` (this domain belongs to Arquivo.pt) which appeared a total of 102,799 times in the access log and 88,807 times with status codes `200` or `404`. This URI returned `200` status code only 105 times while `404` status code 88,702 times. We further investigated the status code fluctuation pattern for this URI and found that it would return `404` status code hundreds of times in a row with occasional `200` status code every once in a while. It turned out that the server always returned the `404` status code for requests coming from a specific IP address which has the "`Mozilla/5.0+(compatible; UptimeRobot/2.0; http://www.uptimerobot.com/)`" user-agent, but the `200` status code to everyone else (such as MemGator or the TimeTravel Service). From the user-agent string we can tell it is a server health check service which periodically polls specific resources, but we do not know why the server behaves differently for this user-agent.

7.3.5 ROUTING ACCURACY

After we identified most frequently accessed resources that have never returned a successful response for any of their canonical or non-canonical URI-Rs, we created archival void profiles with these. Table 16 shows the repetition breakdown of the number of URIs that have always returned the `404` status code. There are over 13 million canonicalized URIs that

Table 16. 404-Only URI-R Repetitions in Arquivo.pt Access Logs and False Positive Reduction Due to the Archival Void Profile

Repetitions	**URI-Rs**	**MemGator Requests Saving %**
1s	13,673,599	64.67
10s	698,959	17.00
100s	2,319	8.42
1,000s	99	2.85
10,000s	2	0.00

have always returned the `404` status code, but each of them appeared only 1–9 times while about 0.7 million `404` URIs appeared 10–99 times. The long-tail of low frequency URIs are not suitable for profiling voids as they will increase the size of the profile disproportionately. For example, going from the request saving of 8.42% to 64.67% it would require an increase of about four orders of magnitude in the number of URIs in a voids profile. An attempt to use less detailed profiling policies to reduce the size of the profile would introduce false negatives. However, the last few rows of the Table 16 represent only a few URIs that have been requested thousands or tens of thousands of times and have always returned the `404` status code. Creating a void profile with these would cut the false positives down significantly.

There are over seven million entries in the Arquivo.pt access logs that originated from the MemGator server running at ODU. We analyzed the percentage of requests that could have been avoided if archival voids profiles of various frequencies were made available based on the access log alone. Table 16 shows that about 2.85% false positive requests could have been avoided by only profiling URIs that have appeared thousands of times and have always returned the `404` status code. This saving could have been around 8.42% if we included URIs that were repeated more than hundreds of times. We have reported lower bounds to avoid any false negatives while we believe that the numbers would have been even better if Arquivo.pt had a proper URI canonicalization in place from the beginning.

7.4 WHO SHOULD PROFILE ARCHIVAL VOIDS?

It is very important to keep the profile of archival voids fresh, otherwise false negatives will increase very quickly. Unlike archival holdings profiles, aggressively reducing the URI key size can be harmful in archival voids profiles as users will fail to discover many resources

that are present in a web archive.

An archival voids profile is expected to complement an archival holdings profile, so the entries about what is missing can be very specific. However, it is possible to use an archival voids profile independently, and is ideal for large web archives such as the Internet Archive. If an archive is going to return good results for most of the requests, then it will be wasteful to profile its holdings for the sake of routing. Knowing what it does not contain or is not willing to serve is a more compact way to improve routing accuracy for such web archives.

In 2015, a Twitter bot called *ICanHazMemento* was launched which polls Twitter periodically to fetch new Tweets that contain the hashtag "`#ICanHazMemento`" and a URI in their conversation chain and replies to them with a URI-M, pointing to a memento of the URI in a web archive [197]. To find a suitable URI-M, it would perform a lookup in LANL's TimeTravel service and link back to it. If it does not find any mementos for the URI-R, it attempts to save the resource in one or more archives and then tweet about them. However, the TimeTravel service had caching in place, so it would fail to recognize that a new memento was created for the resource it returned "`404 Not Found`" response and cached it a moment ago. Consequently, users following the link posted as a reply to their tweets will fail to access a memento that was created. The issue was noticed and was fixed soon after by configuring the TimeTravel service to not cache "`404`" responses. An archival voids profile generated by a third party can have a similar issue.

Because of the potential danger of false negatives due to stale archival voids profiles, it is recommended that an archival voids profile is generated by an entity that is close to the source of truth (e.g., a web archive itself). When third parties (e.g., a Memento aggregator) generate such profiles, they should add very specific entries and should update the profile frequently. Also, they should only add resources in such profiles when they have gained enough confidence that the resource is indeed missing and has very little chance to be available anytime soon (e.g., due to successive failure responses of frequently queried resources).

7.5 RECOMMENDATIONS

Based on our assessment, we have some recommendations for those generating such profiles (most likely, web archives themselves or Memento aggregators):

- Keep archival voids profile separate as a paginated resource so that it can be updated and consumed independently and more frequently (which is also a more logical

approach because the data source for the holdings profile would primarily be CDX indexes while voids profiles will be generated using access logs and collection policies).

- Be more specific in including URIs in the voids profile and include shorter URI keys only when the confidence is very high or a domain or TLD is blocked by the collection policy (e.g., pornographic TLD "`.xxx`").
- Update frequently from the list of take down requests from domain owners or governments.
- Include only resources that are high in demand, but missing or prohibited, because listing items that no one is requesting for is not going to benefit in saving unnecessary traffic while exposing more information in public and making the profile big.

On the utilization side proper order of evaluation will be important when there are many competing URI keys for the lookup URI in both the archival holdings profile as well as archival voids profile with different host/path depths.

7.6 CHAPTER SUMMARY

In this chapter we defined and discussed *Archival Voids* and established a means to represent portions of URI spaces that are not present in web archives. With the help of examples we explained the purpose of creating archival void profiles and illustrated how it works in conjunction with the archival holdings profile in a hierarchical manner to describe holdings and voids in more specific portions of the URI spaces. We discussed various sources of truth that can be used to create archival voids profiles. For evaluation we used access logs from Arquivo.pt to create an archival voids profile and analyzed it against our MemGator access logs. In the process we described access patterns in Arquivo.pt and surfaced various corner cases and issues that were present in it. We discussed prevalent Soft-404 TimeMaps in the access logs for many years and techniques we used to remedy that in order to make a more meaningful analysis of the dataset. We discussed the distribution of HTTP response status codes in the access logs and reported how these status codes changed over time for various URIs. We evaluated the routing accuracy against various archival voids profiles created from these access logs and found that we could have avoided more than 8% of the false positives (on top of the accuracy we got from archival holdings profile as discussed in Chapter 6) if Arquivo.pt were to provide an archival voids profile based on URIs that were requested

hundreds of times and never returned a success response. Finally, we discussed who should create archival voids profile and provided some guidelines based on our understanding.

CHAPTER 8

SERIALIZATION AND DISSEMINATION

In this chapter we describe our extended form of *SURT* that we use as *URI-Keys* in our flexible archive profiles. We also describe a *Unified Key Value Store* file format that we evolved from *CDXJ*. We utilize these for archive profile serialization and call it a *MementoMap* [32]. Finally, we evaluate various aspects of *MementoMaps* generated with different levels of details. This chapter addresses our second research question, "***RQ2**: How to build an archive profile that will best summarize an archive's holdings/voids and allow for dissemination and exchange?*"

8.1 UNIFIED KEY VALUE STORE (UKVS)

Unified Key Value Store (UKVS) [14] is an evolving file format proposal that is a contribution of this *MementoMap* work. It is an evolution of the *CDXJ* format that we earlier proposed to be used by *Archive Profiles* [29]. This format utilizes extended *SURT* with wildcard support and improves various other aspects to simplify it and eliminate some limitations of our prior proposal (such as not being able to express URIs that are absent from an archive or lack of support to merge two profiles generated with different profiling policies). We generalized the format to be more inclusive and flexible after we realized its utility in many web archiving related use cases (such as indexing, replay access control list (ACL), fixity blocks [47], and extended *TimeMaps*) and many other places such as extended server logging.

Suppose we want to store records of people in which their last and first names will make the combined lookup key and their degree and profession along with an arbitrary number

Table 17. An Example of Sparse Tabular Data

LName	**FName**	**Degree**	**Profession**	**Twitter**	**DoB**	**Award1**	**Award2**
Doe	Jane	PhD	Professor	`@jdoe`	-	-	-
Doe	John	MS	Consultant	-	-	-	-
Roe	Richard	PhD	Scientist	-	May 7, 1920	Fiction 42	Mad Scientist
Shmoe	Joe	-	Lab Asst.	-	-	-	-

```
1 !fields {keys: ["LName", "FName"], values: ["Degree", "Profession"]}
2 Doe   Jane    PhD     Professor   {Twitter: "@jdoe"}
3 Doe   John    Masters Consultant
4 Roe   Richard PhD     Scientist   {DoB: "May 7, 1920", Awards: ["Fiction 42", "Mad Scientist"]}
5 Shmoe Joe     -       "Lab Asst."
```

Fig. 54. An Example of Structured Data With Mandatory and Optional Fields

of other optional attributes will be the value stored in each record. We can use a fixed column tabular data format for this (as shown in Table 17), but it will have two major disadvantages: 1) fields with sparse values are wasteful and 2) adding a new field (e.g., "Award3") will affect all existing records. Additionally, the tabular formats like CSV/TSV have no information about what fields can be used as lookup keys. Other structured data formats like JSON, XML, or YAML can be more expressive, but they are not friendly to Unix text processing tools.

Our proposed UKVS data format solves these issues as illustrated in Figure 54. UKVS is a textual file format for storing data that has one or more key fields for lookup and corresponding value fields. UKVS also allows an optional object to annotate, enrich, or subdivide each record. It uses Object Resource Stream (ORS) [25] syntax with some well-defined fields and structure. UKVS format has the simplicity of popular file formats such as Comma Separated Values (CSV) while being flexible like JavaScript Object Notation (JSON). This makes it suitable for streaming and simple to process using traditional text processing tools, irrespective of the number of records in the file. If the file is sorted, this formats enables quick lookup using binary search, even in large files, which makes it suitable for indexing records.

UKVS consists of some metadata records in the header followed by the body containing data records with one line per record. The format of UKVS data record entries is illustrated in Figure 55. The names of the fields of the data records that belong to keys and values are advertised in the metadata headers in their respective order as shown in Figure 54 Line 1. The "!fields" metadata here describes the structure of the data records where the first two fields "LName" and "FName" are primary and secondary lookup keys respectively. The third and fourth fields represent corresponding values as "Degree" and "Profession" respectively. These four fields are mandatory, so their missing or unknown values must be filled with a placeholder (as illustrated in Figure 54 Line 5 with a "-" sign). Fields other than Optional Single Line JSON (OSLJ) are separated by whitespace and multi-token values are quoted in

```
<key fields...> <value fields...> {<optional single line JSON block>}
```

Fig. 55. UKVS Generic Record Format

double quotes. While multiple consecutive whitespace characters are allowed for readability, they should be avoided to save bytes as per the application needs. This format is similar to CSV/TSV files (or other tabular data formats), but the addition of the OSLJ block allows each record to have arbitrary data, unique to each record, in a structured way.

8.2 MEMENTOMAP FILE FORMAT

MementoMap is our proposed format to serialize archive profiles in a flexible and concise manner. It uses *UKVS* data format and adds a semantic layer on top of it by defining various keywords and fields. It is inspired by the simplicity of `sitemap` and `robots.txt`, but different from them in some aspects:

- Unlike sitemap and robots.txt, *MementoMap* of an archive can be published by third parties, not just the archives
- Unlike sitemap, *MementoMap* may use wildcards in URIs (similar to robots.txt) to reduce the number of records significantly
- Unlike sitemap, *MementoMap* can have flat pagination (generally more suitable for sorted resources) where page can link to next, previous, first, and last pages instead of a nested index of *MementoMap* pages
- Unlike robots.txt, *MementoMap* is safe to sort, split, and merge
- *MementoMap* provides means of variations in organizing various fields to optimize for space and application-specif usability while keeping the essence of records the same
- *MementoMap* enables means to limit the size of the file or the number of records and dynamically adapt to it

The example shown in Figure 56 has two parts, first five lines are headers and the last five lines are data records. The `!context` entry in the header section describes where to look for definitions and descriptions of terms used in the document. The `!id` entry points to the

```
!context ["http://oduwsdl.github.io/contexts/ukvs"]
!id {uri: "http://archive.example.org/"}
!fields {keys: ["surt"], values: ["frequency"]}
!meta {type: "MementoMap", name: "A Test Web Archive", year: 1996}
!meta {updated_at: "2018-09-03T13:27:52Z"}
*                    54321
com,*                10000
com,twitter)/        100
com,twitter)/*       250
uk,co,bbc)/images/* 300
```

Fig. 56. A Basic *MementoMap* Example File

web archive the *MementoMap* is about. The `!fields` entry suggests that in the data records there are only two mandatory fields of which the first one is a SURT (a transformation of URIs that we describe below) of the lookup URI that is used as the key and the second field holds the frequency of archiving for a given SURT key. The first data row `* 54321` suggests that there are a total of 54321 mementos of all the URIs in the archive while `com,* 10000` suggests that there are 10000 mementos that have `.com` TLD in their URIs. The next two entries suggest that the root page of `twitter.com` has 100 mementos, but all the URIs from `twitter.com` collectively have a total of 250 mementos. Finally, there are 300 mementos of resources from `bbc.co.uk` who's path begins with `/images/`.

8.2.1 THE SURT FIELD

We described basic *SURT* in Chapter 2 (Section 2.6.2). Figure 57a illustrates a sample of sorted *SURTs* and highlights different segments. We have extended *SURTs* to support wildcard to allow grouping of *URI Keys* with the same prefix and roll them up into a single key. A visual representation of these *SURTs* is illustrated in Figure 57b in the form of a tree that segregates layers of *Scheme*, *Host*, *Path*, and *Query*. It further annotates various depths of *Host* and *Path* segments as *H0, H1, H2...* and *P0, P1, P2...* that will be useful in understanding some terminologies used later. *SURTs* also allow credentials and port numbers, but we omitted them from the illustration for brevity. It is worth noting that the scheme portion is common in all *HTTP/HTTPS* URIs and has no informational value, hence the "`https://(`" prefix is often omitted. SURT are generally reversible, but in the *MementoMap* context (and anywhere where they are used as index keys) we can add

support for partial SURTs with the help of wildcards to represent a collection of URIs.

8.2.2 THE FREQUENCY FIELD

The `frequency` field is a concise way to represent the archival activity of a URI or a set of URIs (represented by a SURT with wildcard). Figure 58 illustrates the grammar for parsing the frequency field.

In its simple form it represents the number of mementos (or URI-M counts) of a URI or all the URIs in a set. However, it also allows to express number of unique URIs (URI-R counts) along with their URI-M counts for a given set of URIs using a wildcard as illustrated in Figure 59.

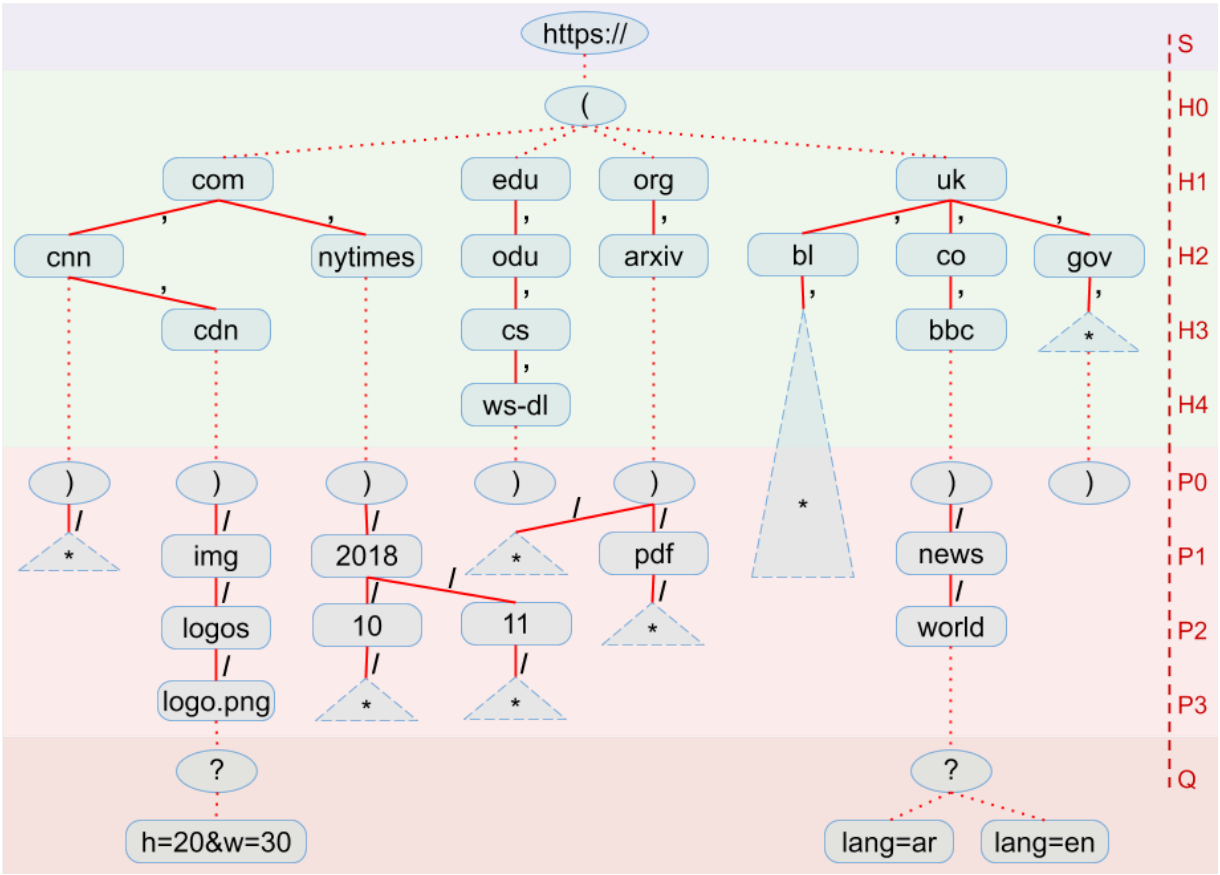
The format of the frequency field here is `[<urim-count>]/[<urir-count>]`. The first record in this example suggests that there are 300 mementos of all the URIs from `apple.com`. The second record suggests that there are a total of 100 unique URI-Rs (original URIs) from `cnn.com` that are archived a total of 400 times, that means on an average each URI is archived four times. In the next entry, `200/` suggests that there are 200 URI-Ms from `example.com`, but the number of unique URI-Rs is unknown. This entry is equivalent to saying `200` as the separator sign (i.e., forward slash `/`) is optional when only the URI-M count is known. The decision to make the separator optional here is to save bytes in the case that is goint to be potentially more common as counting mementos (URI-Ms) and updating URI-M counts are generally easier than counting unique URI-Rs and updating the URI-R count later. In the next entry, `/50` suggests that the number of unique URI-Rs from `facebook.com` that are archived is 50, but how many times are they archived (URI-M count) in unknown. The separator in this case is mandatory to avoid ambiguity. When neither URI-M nor URI-R counts are known, an explicit `/` is mandatory (as illustrated for `google.com`) to be used as the placeholder and to signify that none of the counts are known.

An explicit value of zero URI-Ms (as illustrated for `twitter.com`) gives a means to express absence of resources matching the key, which can be useful for big archives like the Internet Archive, as their positive profile (i.e., the profile that lists resources that are present) might be huge, so they can instead advertise sets of URIs that they do not have (or can not serve) any mementos for, but get a lot of replay requests. These entries with zeros are how we serialize archival voids as discussed in Chapter 7.

An explicit zero memento count in a *MementoMap* is also useful to express holdings of a bigger set, but lack of holdings in some of its subsets as illustrated in Figure 60. This example

```
http://(com,cnn)/*
http://(com,cnn,cdn)/img/logos/logo.png?h=20&w=30
http://(com,nytimes)/2018/10/*
http://(com,nytimes)/2018/11/*
http://(edu,odu,cs,ws-dl)/
http://(org,arxiv)/*
http://(org,arxiv)/pdf/*
http://(uk,bl,*
http://(uk,co,bbc)/news/world?lang=ar
http://(uk,co,bbc)/news/world?lang=en
http://(uk,gov,*)/
```

(a) A sample list of sorted *SURTs*. Different colors signify *Scheme*, *Host*, *Path*, and *Query* segments. The "`https://(`" prefix is common in all *SURTs*, hence removed in practice.



(b) A visual representation of *SURTs* as a tree. Different colored regions signify *Scheme*, *Host*, *Path*, and *Query* segments. Each node of the tree contains a token and each edge denotes the separator of the corresponding segment. Dotted lines indicate transition from one segment to the next. Dotted triangles with a wildcard character "*" denote a sub-tree. Trailing slashes are removed from this representation. Labels on the right hand side (i.e., `S`, `H0`–`Hn`, `P0`–`Pn`, and `Q`) denote corresponding depth in each segment.

Fig. 57. Illustration of SURTs With Wildcards

```
zero      = "0" ;
nonzero   = "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" ;
digit     = zero | nonzero ;
number    = nonzero, { digit } ;
approx    = "~" | "+" | "-" ;
count     = zero | number, [ approx ] ;
frequency = count | [ count ], "/", [ count ] ;
```

Fig. 58. Extended Backus–Naur Form (EBNF) Grammar for the frequency Field.

```
!fields {keys: ["surt"], values: ["frequency"]}
com,apple)/*    300
com,cnn)/*      400/100
com,example)/*  200/
com,facebook)/* /50
com,google)/*   /
com,twitter)/*  0
```

Fig. 59. Variations in frequency Field Value of a *MementoMap*

```
!fields {keys: ["surt"], values: ["frequency"]}
com,cnn)/*          400
com,cnn)/profiles/* 0
com,cnn)/world      0
```

Fig. 60. Zero frequency for a More Specific URI Subtree in a *MementoMap*

```
!fields {keys: ["surt"], values: ["frequency"]}
com,yahoo)/* 0/20
```

Fig. 61. Zero Mementos of Non-Zero URI-Rs in a *MementoMap*

```
!fields {keys: ["surt"], values: ["frequency"]}
com,apple)/*    300
com,cnn)/*      400+/100
com,example)/*  200-/150~
com,facebook)/* /50+
```

Fig. 62. Non-Precise Frequencies in a *MementoMap*

suggests that there are a total of 400 URI-Ms for `example.com`, but zero mementos for URI-Rs that start with `cnn.com/profiles/` and zero mementos for the URI-R `cnn.com/world`. The last two entries here are more specific subsets that override a more general record set by the first entry.

There could be another case where a zero URI-M count is reported, but has a non-zero URI-R count as shown in Figure 61. This can be useful to report the number of URI-Rs that are blocked for legal reasons, in an embargo period, or added in the seed list or frontier queue of the crawler, but not yet ready to be replayed.

Counting mementos and unique original URIs and keeping track of these numbers as the archive grows is a difficult task. For third-party observers it is often impractical to estimate exact values of these counts. Additionally, when counting is performed on subsets of the archival collection and merged, it is difficult to maintain the accuracy of these counts. Luckily, in many use cases a rough estimate might be good enough while avoiding the cost of maintaining exact values. However, it is important to acknowledge whether a frequency value is exact or a rough estimate. This can be expressed using following symbol suffixes to either of the URI-M or URI-R counts (or both):

- No suffix for an exact value
- `+` suffix for a lower boundary
- `-` suffix for an upper boundary
- `~` suffix for an approximate value

These looseness symbols can optionally be used with any numeric values of URI-M and URI-R counts independently to form different combinations. The example in Figure 62 illustrates that there are:

```
!fields {keys: ["surt"], values: ["frequency"]}
com,twitter)/* 100/35 {datetime: {"2014": "20~/10", "2015": "60+/30+"}}
```

Fig. 63. Optional Memento Distribution Over Time in a *MementoMap*

```
!fields {keys: ["surt", "datetime"], values: ["frequency"]}
com,twitter)/* :    100/35
com,twitter)/* 2014 20~/10
com,twitter)/* 2015 60+/30+
```

Fig. 64. Mandatory Memento Distribution Over Time in a *MementoMap*

- exactly 300 mementos for `apple.com` URI-Rs,
- at least 400 mementos for exactly 100 unique `cnn.com` URI-Rs,
- at most 200 mementos for approximately 150 unique `example.com` URI-Rs, and
- an unknown number of mementos for at least 50 unique `facebook.com` URI-Rs

8.2.3 THE DATETIME FIELD

So far, we have been using `surt` as the only lookup key with no information about how the archiving activity of a URI-R or a set of URI-Rs is distributed over time. Knowing about the temporal activity could be very helpful for Memento routing (and many other applications) for resources that were actively archived during a short period of time and hardly had any archival activity for the rest of the time. To express the distribution of archiving activity over time the Optional Single Line JSON (OSLJ) block can be utilized as illustrated in Figure 63.

This example suggests that there are a total of exactly 100 mementos of exactly 35 unique URI-Rs from `twitter.com`. The OSLJ block further decomposes this `frequency` value and suggests that in the year 2014 exactly 10 unique URI-Rs were archived about 20 times and more than 30 unique URI-Rs were archived more than 60 times in 2015.

Alternatively, this information can be expressed with a different organization as illustrated in Figure 64. We have moved the information buried inside of the OSLJ into a

```
!fields {keys: ["surt", "datetime"], values: ["frequency"]}
com,apple)/  20160213052637 1
com,apple)/* 20160213       20
com,apple)/* 2010:2012      100
com,apple)/* :2009          50
com,apple)/* 201603:        120
com,apple)/* :              300
```

Fig. 65. Various Datetime Range Examples in a *MementoMap*

secondary key field as the first two columns `["surt", "datetime"]` are now lookup keys, making it easier to process using traditional text processing tools. We have also avoided unnecessary bytes that were only present to format a valid JSON block and moved the name of the field in the UKVS headers instead of repeating it with each record. However, we have introduced more lines that repeat the value of the `surt` field. Also, if not all entries have frequency values decomposed over time, then the corresponding column will have unnecessary placeholder (a ":" symbol in this case). So, there is clearly a trade-off here that can be evaluated based on the nature of an archive and the application to optimize accordingly.

Potential values of the `datetime` field include valid substrings of the 14-digit date and time format (`YYYY[MM[DD[hh[mm[ss]]]]]`) or a range composed of such substrings separated by a colon ":" symbol. If the start or the end of the range is not given it is considered the beginning of archiving and the current time respectively.

The example in Figure 65 (intentionally unsorted for gradual explanation) illustrates different scenarios `datetime` field can be specified. The first entry suggests that the root page of the `apple.com` is archive exactly once at the exact time `20160213052637` (i.e., `2016-02-13T05:26:37Z`). The second line suggests that URI-Rs from `apple.com` were archived a total of 20 times on February 13, 2016 (i.e., their `Memento-Datetime` in 14-digit format has a prefix of `20160213`). The next entry of `2010:2012` suggests that there are 100 mementos of `apple.com` URIs from year 2010 to 2012. The next line with `:2009` suggests that there are 50 mementos with `datetime` in 2009 and before. The entry with `201603:` suggests that there are a total of 120 mementos starting from March 2016 till now. Finally, the last line suggests that there are an overall 300 mementos of `apple.com` URI-Rs.

8.2.4 OTHER FIELDS

A *MementoMap* may also contain `frequency` information decomposed over other useful fields such as `status`, `language`, `mime`, etc. One or more of these fields can be included in the OSLJ or as key fields (or a combination of both) as described the usage of the `datetime` field above. Knowing the distribution of certain mementos over their `status` codes could help reduce unnecessary traffic for resources that were captured numerous times, but resulted largely in `5xx` codes. Similarly, knowing about the number of redirects (i.e., recorded `3xx` responses) can give a better estimate of useful holdings of resources in an archive. The OSLJ block is also open for application specific data or custom fields for specific human-friendly notes.

8.3 MEMENTOMAP IMPLEMENTATION

We have a reference implementation described in Chapter 5 Section 5.6. It can generate *MementoMaps* from CDX files or a list of URIs, compact them based on certain set of parameters, and enable fast lookup in them.

8.3.1 MEMENTOMAP GENERATION

Generating a *MementoMap* begins by scanning *CDX/CDXJ* files, performing fulltext search, filtering access logs, or any other means to identify what *URIs* an archive holds (or does not hold). These *URIs* are then converted to *SURTs* (if not already) and their query section is stripped off. We call these partial *SURTs* as *HxPx URI Keys* (which means a *URI Key* that has all the host and path parts, but no query parameters). Previously, we found that removing query parameters from these *SURTs* reduces the file size and the number of unique *URI Keys* significantly without any significant loss in the lookup *Accuracy* [29]. We then create a text file with its first column containing *HxPx Keys* and the second column as their respective *frequencies*. The *frequency* column in its simplest form can be the count of each *HxPx Key*. Finally, necessary metadata is added and the file is sorted as the baseline *MementoMap*.

8.3.2 MEMENTOMAP COMPACTION

In order to make a less detailed *MementoMap* (which is desired for efficient dissemination and long-lasting freshness at the cost of increased false positives), we pass a detailed *MementoMap* through a compaction procedure which yields a summarized output that contains

```
func host_keys(surt)
  s = surt.split(")")[0].split(",", MAXHOSTDEPTH)
  return [s[:i].join(",") for i in 1..len(s)]

func path_keys(surt)
  s = surt.split("?")[0].split("/", MAXPATHDEPTH)
  return [s[:i].join("/") for i in 1..len(s)]

func compact(imap, omap, opts)
  htrail = [None] * MAXHOSTDEPTH
  ptrail = [None] * MAXPATHDEPTH
  for line in imap
    key, freq, *_ = line.split()
    k = host_keys(key)
    for i in range(len(k))
      if htrail[i] == k[i] # Existing branch
        htrail[i][1] += freq
      else # New branch
        for j in range(i, MAXHOSTDEPTH)
          if rollup_threshold_reached
            omap.seek(htrail[j][3]) # Move back
            omap.write(htrail[j][:1].join(",* "))
            reset_remaining_trail(ptrail, 0)
        reset_remaining_trail(htrail, i)
      if !htrail[i] # New tree node
        htrail[i] = [k[i], freq, 0, omap.tell()]
        htrail[i-1][2]++ # Incr parent's children
    # Repeat similar logic for path segment
    omap.write(line)
    omap.truncate() # Clear any rollup residue
```

Fig. 66. *MementoMap* Compaction (and Generation) Procedure

fewer lookup keys by rolling sub-trees with many children nodes up and replacing them with corresponding wildcard keys. Our compaction algorithm is illustrated with pseudo-code in Figure 66. As opposed to an in-memory tree building (which will not scale), it is a single-pass procedure with minimal memory requirements and does not need any special hardware to process a *MementoMap* of any size. We leverage the fact that the input *MementoMap* is sorted, hence, we can easily detect at what depth of host or path segments a branch differed from the previous line. We keep track of the most recent state of host and path keys at each depth (up to `MAXHOSTDEPTH` and `MAXPATHDEPTH`), their corresponding cumulative frequencies, how many children nodes each of them have seen so far, and the byte position of the output file when these keys were seen the first time. Each time we

encounter a new branch at any depth, we check to see if a roll up action is applicable at that depth or further down in the existing tree based on the most recent states and the compaction parameters supplied. If so, we move the write pointer in the output file back to the position where the corresponding key was observed first, then we reset the state of all the deeper depths and update them with the current state. As a consequence of this progressive processing, the trailing part of the output file is overwritten many times. The input file does not have to be the baseline *MementoMap*, any *MementoMap* can be supplied as input with fresh compaction parameters to attempt to further compact it. Our algorithm is parallel processing-friendly if the input data is partitioned strategically (e.g., processing each *TLD*'s records on separate machines and combining all compacted output files). It is worth noting that sub-trees of the path section are neither independent trees nor have a single root node (as shown in Figure 57b), as a result, certain implementation details can be more complex than a simple tree pruning algorithm.

8.3.3 LOOKUP IN A MEMENTOMAP

The algorithm for lookup in a *MementoMap* is illustrated with pseudo-code in Figure 67. Given a *URI*, we first generate all possible lookup keys, in which all keys but the longest one have a wildcard suffix (e.g., "`Www.Example.COM/a/b?x=y&c=d`" yields "`com,example)/a/b`", "`com,example)/a/b/*`", "`com,example)/a/*`", "`com,example)/*`", and "`com,*`" as lookup keys). We then perform a binary search in the *MementoMap* with lookup keys in decreasing specificity until we find a match or all the keys are exhausted. In the case of a match, we return the matched lookup key and corresponding frequency results.

8.4 MEMENTOMAP DISSEMINATION

For dissemination and discovery of *MementoMaps* we propose that web archives make their *MementoMap* available at the *well-known URI* "`/.well-known/mementomap`" under their domain names. Alternatively, a custom *URI* can be advertised using the "`mementomap`" *link relation* (or "`rel`") in an *HTTP Link* header or *HTML* <*link*> element (after this link relation is registered in the IANA registry). Third parties hosting *MementoMaps* of other archives can use the "*anchor*" attribute of the *Link* header to advertise a different context. Moreover, *MementoMaps* are self-descriptive as they contain sufficient metadata in their headers to establish a relationship with their corresponding archives. *MementoMaps* support pagination that can be discovered after retrieving the primary *MementoMap* from a *well-known URI* or by any other means.

```
func lookp_keys(uri)
  key = surtify(uri).split("?")[0].strip("/")
  keys = [key]
  while "," in key
    keys.append(sub("(.+[,/]).+$", "\1*", key))
    key = sub("(.+)[,/].+$", "\1", key)
  return keys

func bin_search(mmap, key)
  surtk, freq, *_ = mmap.readline().split()
  if key == surtk # First line matched
    return [surtk, freq]
  left = 0
  mmap.seek(0, 2) # Go to the EOF
  right = mmap.tell()
  while (right - left > 1)
    mid = (right + left) / 2
    mmap.seek(mid)
    mmap.readline() # Skip partial line
    surtk, freq, *_ = mmap.readline().split()
    if key == surtk
      return [surtk, freq]
    elif key > surtk
      left = mid
    else:
      right = mid

func lookup(mmap, uri)
  for key in lookp_keys(uri)
    result = bin_search(mmap, key)
    if result
      return [key, result]
```

Fig. 67. *MementoMap* Lookup Procedure

8.5 EVALUATION

For evaluation we used the complete index of *Arquivo.pt*, complete logs of our *MemGator* service, and generated *MementoMaps*. We first examine logs, then describe holdings of *Arquivo.pt* in detail, and finally measure the effectiveness of various *MementoMaps*. *Arquivo.pt* [117] was founded in 2008 with the aim to preserve web content of interest to the Portuguese community, but not limited to just the `.pt` *TLD* (as shown in Table 18). It has since archived about 5B mementos of which some data was donated to it by other archives,

Table 18. Top *Arquivo.pt TLDs*

TLD	URI-R%	URI-M%
.pt	61.422	68.266
.com	19.610	19.643
.eu	8.665	4.262
.net	1.973	1.829
.org	1.790	1.263
.de	0.635	0.343
.br	0.617	0.470
.uk	0.449	0.260
.fr	0.347	0.173
.nl	0.274	0.131
.mz	0.236	0.414
.pl	0.226	0.104
.io	0.223	0.208
.edu	0.201	0.096
.es	0.200	0.126
.it	0.198	0.109
.cv	0.198	0.335
.ru	0.196	0.203
.ao	0.156	0.295
.us	0.142	0.102
.cz	0.117	0.057
.info	0.113	0.160
IP Addresses	0.070	0.050
Other TLDs	1.941	1.149

including IA, explaining why its temporal spread extends back before the *Arquivo.pt*'s founding date. We analyzed 1.8T of *Arquivo.pt*'s complete *CDXJ* index in production. A brief summary of the dataset is shown in Table 19. We used it along with ODU's *MemGator* server logs to evaluate this work.

Table 19. *Arquivo.pt* Index Statistics

Attributes	**Values**
CDXJ files	70
Total file size	1.8T
Compressed file size	262G
Temporal coverage	1992–2018
CDXJ lines	5.0B
Mementos (URI-Ms)	4.9B
Unique URI-Rs	2.0B
Unique HxPx keys	1.1B
Unique hosts	5.8M
Unique IP addresses	15K

8.5.1 ARCHIVED VS. ACCESSED RESOURCES

We analyzed over three years of our *MemGator* logs containing records about 14 different web archives. In its lifetime it has served a total of 5,241,771 requests for 3,282,155 unique *URIs*. Table 20 shows the summary of our log analysis in which IA has over 35% hit rate, and every other archive is below 10% (down to zero) in decreasing order of hit rate. *Arquivo.pt* is showing a 3.35% hit rate, so we cross checked it with the full index and found that there are only 1.64% unique *URI-Rs* from the *MemGator* logs that are present in *Arquivo.pt* (note that the *CDX* data even includes recent mementos that would have generated a miss prior to them being archived). The difference in these numbers is perhaps a result of some archived *URIs* being looked for more frequently. We also found that these URI-Rs represent less than 0.003% of the holdings of *Arquivo.pt*, which shows the poor utilization of the archive by users of our MemGator server. This low percentage of overlap in access logs and archive indexes conforms to our earlier findings [29]. The table shows an overall 93% miss rate, which is all wasted traffic and delayed response time. Identifying sources of such a large miss rate can save resources and time significantly, which is the primary motivation of this work.

There are some other notable entries in Table 20 such as low number of requests to PastPages which was excluded from being polled in the early days due to its zero hit rate and high error rate and eventual shutdown in 2018. NRS (National Records of Scotland) is

Table 20. *MemGator* log responses from various archives. Data ranges from 2015-10-25 to 2019-01-16.

Archive	**Request**	**Hit%**	**Miss%**	**Err%**	**Sleep**
Internet Archive	4,723,880	35.76	63.68	0.56	1,594
Archive-It	5,011,385	9.14	90.38	0.48	1,556
Archive Today	5,151,720	8.44	88.96	2.60	1,920
Library of Congress	4,862,458	4.77	94.31	0.92	2,705
Arquivo.pt	4,300,221	3.35	96.29	0.36	1,153
Icelandic	5,126,706	2.22	97.14	0.64	3,143
Stanford	5,178,835	1.54	98.02	0.43	1,482
UK Web Archive	5,113,984	1.49	86.30	12.20	2,779
Perma	4,116,099	1.32	98.67	0.01	46
PRONI	5,165,805	0.75	98.72	0.54	1,608
UK Parliament	5,181,991	0.63	98.85	0.52	1,542
NRS	2,683,311	0.21	99.77	0.01	46
UK National	5,178,184	0.10	99.45	0.45	1,457
PastPages	22,058	0.00	62.90	37.10	0
All	61,816,637	5.44	92.92	1.64	21,031

a new addition to the list, hence it shows a low number of requests. The high error rate of the UK Web Archive was primarily caused by a bug in the Go language (used to develop *MemGator*) that was not cleaning idle TCP connections that were already closed by the application. As a result, UKWA's firewall was seeing an ever increasing number of open, but idle connections, hence dropping packets after a hard limit of 20 concurrent connections per host. This has since been fixed after the release of the Go language version 1.7. We have later introduced an automatic dormant feature that puts an upstream archive to sleep for a configurable amount of time after a set number of successive errors. This new feature also addresses situations when archives go out of service and the maintainers of aggregators do not notice it in a timely manner [43, 44, 45, 46].

Figure 68 shows a breakdown of what people are looking for in archives and what web archives hold. The 1.1K entry in the "Ones" row and "Tens" column shows that there are over a thousand *URI-Rs* that were requested 10–99 times in *MemGator* and each has 1–9

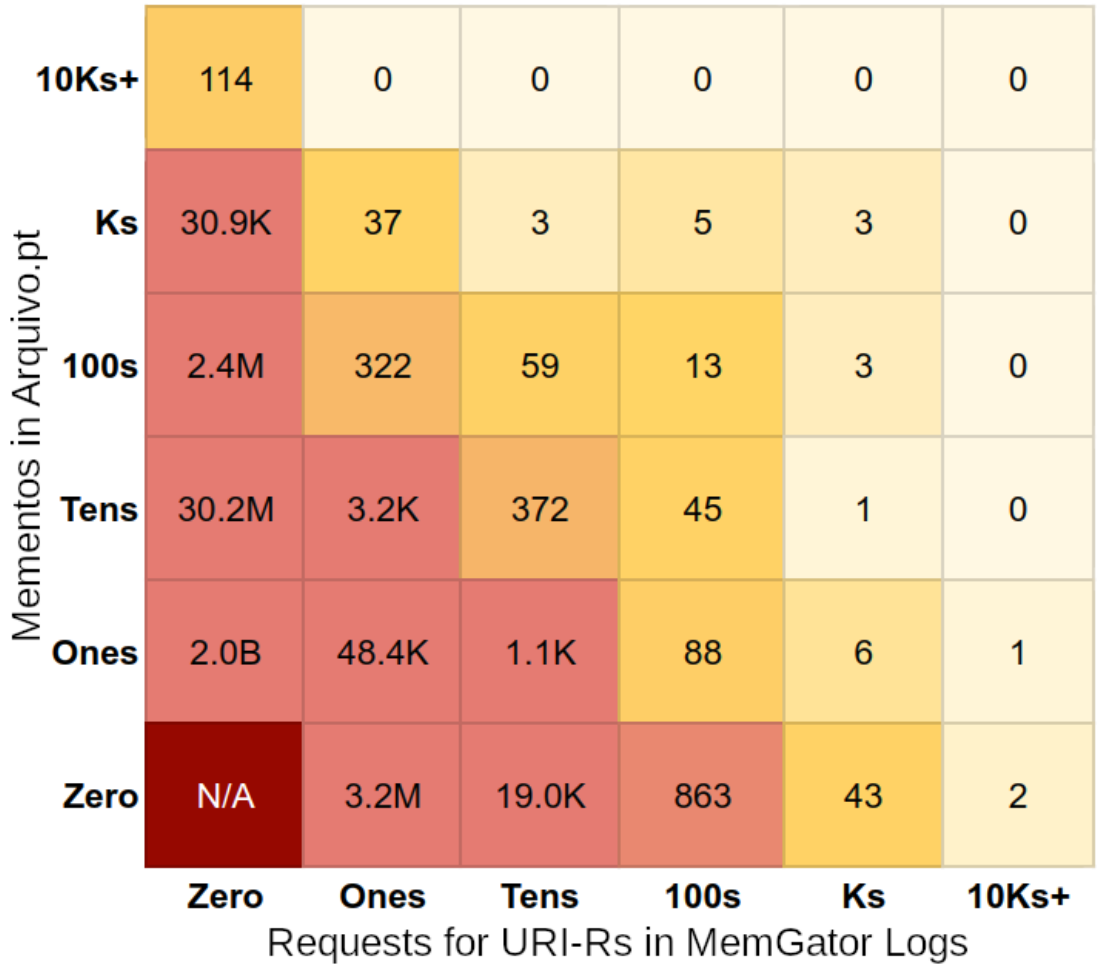


Fig. 68. Overlap between archived and accessed resources in *Arquivo.pt*. *Ones* denote single digit non-zero numbers (i.e., 1–9), *Tens* denote two digit numbers (i.e., 10–99), and so on. The *Zero* column shows the number of mementos of various *URI-Rs* that are never accessed using *MemGator*. The *Zero* row shows the number of access requests for various *URI-Rs* using *MemGator* that are not archived. The *(Zero, Zero)* cell denotes *N/A* because the number of resources that are neither archived nor accessed is unknown.

mementos in *Arquivo.pt*. Large numbers in the "Zero" column show there are a lot of mementos that are never requested from *MemGator*. Similarly, the "Zero" row shows there are a lot of requests that have zero mementos in *Arquivo.pt*. Another way to look at it is that a content-based archive profile will not know about the "Zero" row and a usage-based profile will miss out the content in the "Zero" column. Active archives may want to profile their access logs periodically to identify potential seed URIs of frequently requested missing

Table 21. *URI-M* vs. *URI-R* Summary of *Arquivo.pt*

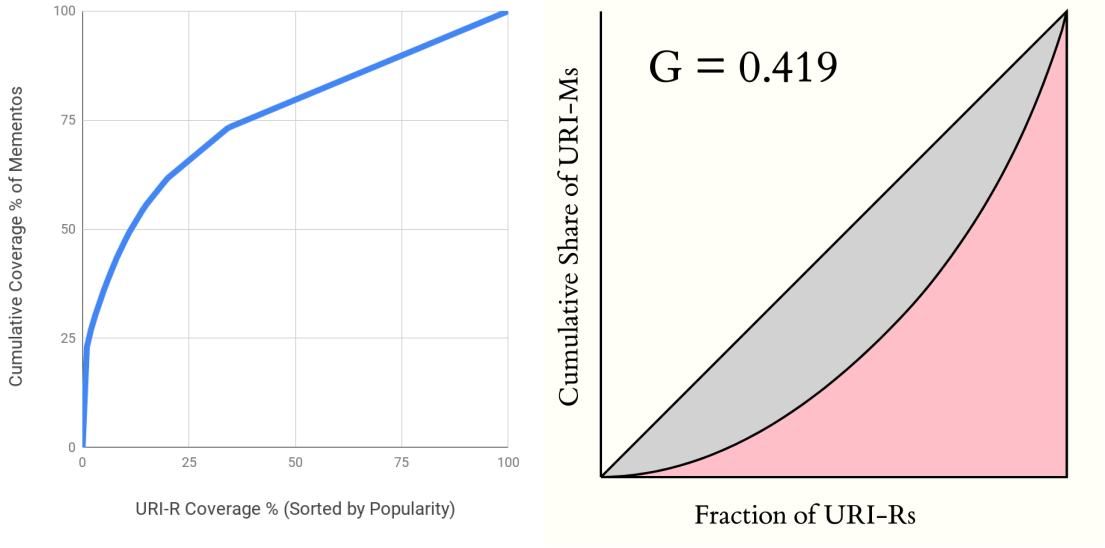
Attributes	**Values**
Unique URI-Rs	1,999,790,376
Total number of mementos	4,923,080,506
Maximum mementos for any URI-R	2,308,634
Median (and Minimum)	1
Mean mementos per URI-R (γ)	2.46
Standard Deviation	57.20
Gini Coefficient	0.42
Pareto Break Point	70/30

resources that are within the scope of the archive. Ideally, we would like more activity along the diagonal that passes from the (Zero, Zero) corner, except the corner itself, which suggests there are undetermined number of *URI-Rs* that were never archived or accessed.

8.5.2 HOLDINGS OF ARQUIVO.PT

Table 21 and Figure 69 summarize the distribution of *URI-Ms* over *URI-Rs* in *Arquivo.pt*. Almost 2M unique *URI-Rs* in *Arquivo.pt* have an average of 2.46 mementos per *URI-R* (γ value [29]), but this distribution is not uniform. The top 30% *URI-Rs* account for 70% of the mementos, for a *Gini Coefficient* of 0.42 [252]. Additionally, the *Median* is one, which means at least half of the *URI-Rs* have only one memento. Furthermore, the most frequently archived *URI-R* has 2.3M mementos (i.e., 0.05% of total), so we decided to investigate it further. Table 22 lists the six most archived *URI-Rs*, and they are mostly one pixel clear images and corner graphics primarily used in web designing in the pre-CSS3 era. The only HTML page that shows up in the top list is a login page. We further investigated all the mementos from all the subdomains of the top *URI-R*'s domain and found that the `blank.gif` image was archived out of proportion. This shows another use for archive profiling – identifying such unintentional biases due to misconfigured crawling policies or bugs in crawlers' frontier queue management.

Furthermore, we partitioned *Arquivo.pt*'s index into yearly buckets for analysis as shown in Table 23. Data prior to year 2008 is mostly donated from other sources in the form of many small files, as *Arquivo.pt* was not yet established. However, when everything is put



(a) Percentage of *URI-Rs* by Popularity vs. Cumulative Percentage of Mementos

(b) Gini coefficient of memento over *URI-R* population

Fig. 69. Distribution of Mementos Over *URI-Rs* in *Arquivo.pt*

Table 22. Most archived *URI-Rs* in *Arquivo.pt*. Most of these resources are either single pixel blank images or corner graphics used for styling in the pre-CSS3 era.

URIs	**URI-Ms**
com,wunderground,icons)/graphics/blank.gif	2,308,634
com,wunderground,icons)/graphics/wuicorner.gif	768,250
pt,ipleiria,inscricoes)/logon.aspx	238,292
com,wunderground,icons)/graphics/wuicorner2.gif	207,448
com,lygo)/ly/i/inv/dot_clear.gif	115,221
com,listbot)/subscribe_button.gif	108,530
com,wunderground,icons)/* (including top URI-R)	3,336,086
com,wunderground,* (41 sub-domains)	3,392,676

Table 23. Yearly distribution of *URI-Rs*, *URI-Ms*, and status codes in *Arquivo.pt*. The symbol γ denotes the ratio of *URI-Ms* vs. *URI-Rs*. Column names with a "+" superscript denote cumulative values as yearly data is processed incrementally. While *URI-M*$^+$ represents a running total, *URI-R*$^+$ does not, because some *URI-Rs* are already seen in previous years. Status codes for the last two years (still in embargo period) do not add up to 100% because a significant portion of their entries are either *revisit* records or screenshots.

Year	URI-R	URI-R$^+$	URI-M	URI-M$^+$	Dup. URI-R%	γ	γ^+	2xx%	3xx%	4xx%	5xx%
1992	1	1	1	1	0.00	1.00	1.00	100.00	0.00	0.00	0.00
1993	1	2	1	2	0.00	1.00	1.00	100.00	0.00	0.00	0.00
1994	128	130	225	227	0.00	1.76	1.75	100.00	0.00	0.00	0.00
1995	642	772	742	969	0.00	1.16	1.26	100.00	0.00	0.00	0.00
1996	110,531	111,303	126,600	127,569	0.00	1.15	1.15	99.96	0.01	0.00	0.00
1997	466,515	563,734	847,783	975,352	3.02	1.82	1.73	100.00	0.00	0.00	0.00
1998	447,042	928,112	747,114	1,722,466	18.49	1.67	1.86	99.23	0.77	0.00	0.00
1999	732,866	1,513,381	1,233,994	2,956,460	20.14	1.68	1.95	76.52	10.61	12.84	0.00
2000	1,710,099	2,874,152	13,413,518	16,369,978	20.43	7.84	5.70	86.99	7.24	5.73	0.00
2001	4,837,012	7,286,174	7,873,642	24,243,620	8.79	1.63	3.33	93.87	4.87	1.25	0.01
2002	7,675,876	13,364,488	13,048,749	37,292,369	20.81	1.70	2.79	90.96	5.11	3.92	0.01
2003	11,043,675	21,565,730	19,989,725	57,282,094	25.74	1.81	2.66	92.12	4.45	3.41	0.03
2004	11,550,512	29,460,627	22,810,763	80,092,857	31.65	1.97	2.72	92.00	5.11	2.88	0.01
2005	9,057,866	35,249,604	19,839,405	99,932,262	36.09	2.19	2.83	93.99	3.94	2.07	0.01
2006	5,979,310	39,609,628	15,388,836	115,321,098	27.08	2.57	2.91	92.33	6.29	1.37	0.01
2007	26,841,427	63,396,199	43,021,527	158,342,625	11.38	1.60	2.50	83.03	14.88	2.08	0.01
2008	113,915,969	166,926,098	174,996,303	333,338,928	9.12	1.54	2.00	85.87	8.95	6.18	0.37
2009	249,069,391	383,960,128	355,833,394	689,172,322	12.86	1.43	1.79	87.37	6.55	6.49	0.36
2010	174,786,328	487,044,797	352,019,433	1,041,191,755	41.02	2.01	2.14	87.39	6.83	6.49	0.42
2011	206,966,813	634,061,322	465,274,765	1,506,466,520	28.97	2.25	2.38	89.13	6.21	6.99	0.58
2012	118,916,669	703,235,309	200,042,923	1,706,509,443	41.83	1.68	2.43	87.79	6.66	7.96	0.46
2013	174,913,693	827,924,633	236,583,969	1,943,093,412	28.71	1.35	2.35	84.03	7.28	10.90	0.57
2014	430,555,712	1,166,054,663	536,560,181	2,479,653,593	21.47	1.25	2.13	80.50	7.10	13.47	0.52
2015	558,504,002	1,563,688,006	1,087,680,516	3,567,334,109	28.80	1.95	2.28	78.32	5.12	17.75	0.32
2016	719,889,903	1,999,522,571	1,353,786,928	4,921,121,037	39.46	1.88	2.46	73.20	6.46	20.78	1.30
2017	685,097	1,999,687,103	1,111,999	4,922,233,036	75.98	1.62	2.46	57.82	5.44	7.89	0.22
2018	106,186	1,999,790,376	847,470	4,923,080,506	2.74	7.98	2.46	22.07	5.63	1.38	0.00
All	1,999,790,376	1,999,790,376	4,923,080,506	4,923,080,506	0.00	2.46	2.46	80.74	6.42	13.86	0.66

together it looks like the archiving activity took off significantly in 2007. Low numbers in years 2017 and 2018 are due to *Arquivo.pt*'s embargo policy. It shows that *Arquivo.pt*'s collection is growing by a healthy pace by mostly collecting new *URI-Rs* as well as revisiting on an average 26% of older ones on a yearly basis. We expected γ would change gradually over time, but years 2000 and 2018 had significantly high values with respect to other years. So, we looked for the possibility of increased `3xx` status codes in those years as a potential

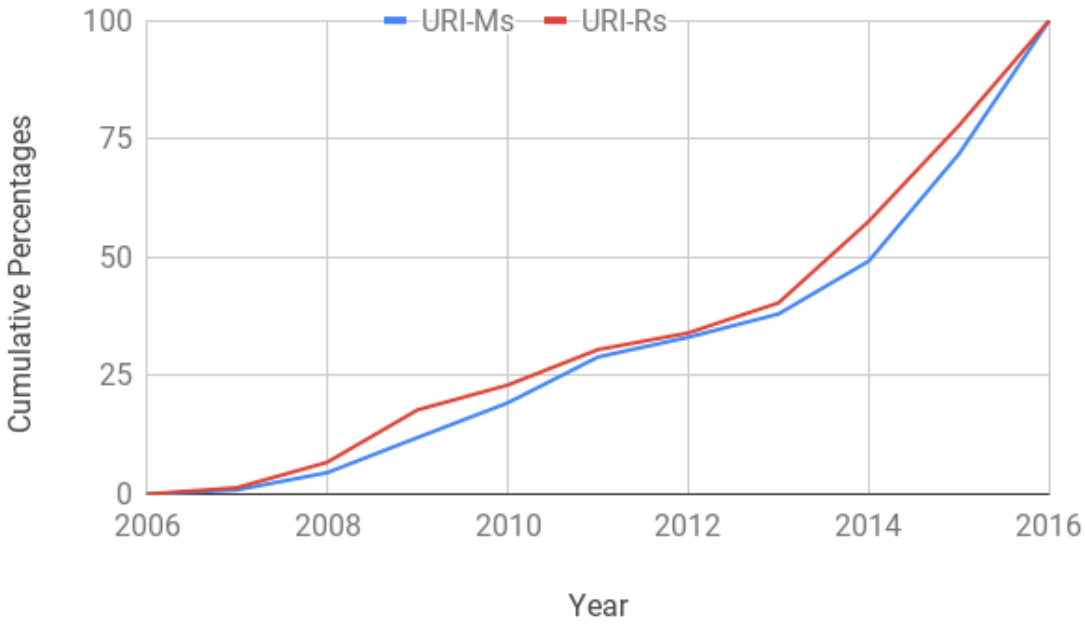


Fig. 70. Cumulative growth of *URI-Rs* and *URI-Ms* in *Arquivo.pt*. Almost half of the mementos are captured in the last two active years alone.

source of increase in γ (e.g., `http` *URIs* redirecting to corresponding `https` version [153, 154]), but we did not see any correlation there. However, the data for these years seems to have come from another source and overall they are insignificant, hence, the cumulative γ^+ is fairly stable between 2 and 3. We noted a significant and steady growth in `4xx` status codes which has crossed the 20% mark in year 2016. Status codes for the last two years (still in embargo period) do not sum up to 100% because a significant portion of their entries are either *revisit* records or screenshots that do not report status codes. In Figure 70 we plotted a cumulative growth graph of both *URI-Ms* and *URI-Rs* to see the shape [147] of *Arquivo.pt* during the active region. Their archiving rate is increasing over time as almost half of the total mementos were archived in the last two active years alone.

8.5.3 THE SHAPE OF ARCHIVED URI TREE

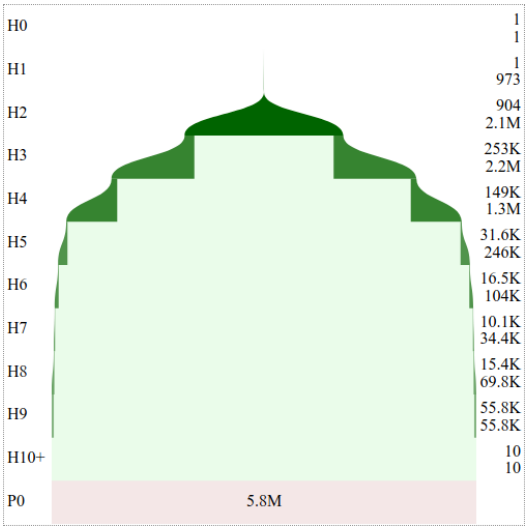
To understand the shape of the *URI Keys* tree in *MementoMap* we first investigated the number of unique *Domains* and *HxPx Keys* that have certain host or path depths as shown in Table 24. These numbers are relative to the size of the *Arquivo.pt* index, but we believe

Table 24. Unique Items With Exact *Host* and *Path* Depths.

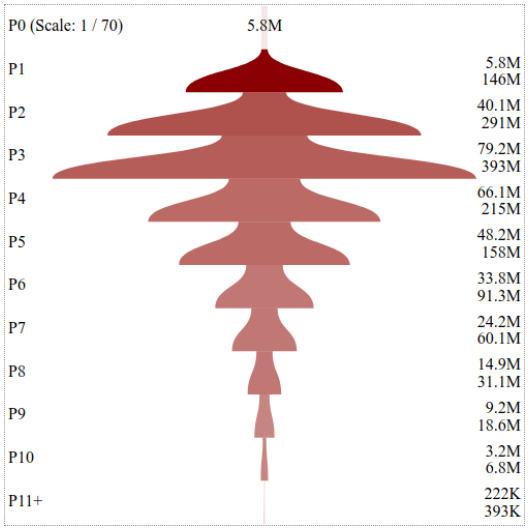
Depth	**Host (Domains)**	**Host (HxPx)**	**Path (HxPx)**
0	1	1	4,456,831
1	119	6,479	113,022,403
2	1,949,845	508,607,506	225,489,773
3	2,097,254	429,000,297	334,455,187
4	1,316,005	161,912,251	174,429,887
5	234,110	21,825,084	127,484,179
6	95,492	7,935,125	68,578,693
7	28,121	3,252,943	45,819,300
8	64,716	3,722,893	22,178,800
9	55,801	2,660,529	15,553,102
10	5	50	6,596,158
11+	5	12	858,856
Total	5,841,473	1,138,923,169	1,138,923,169

a similar trend should be seen in other archives, unless their collection is manually curated and crawled using a more or less capable tool than what is currently being used by many large web archives [184]. There were some outliers in the data that showed a host depth of up to 15 and path depths up to 130, but those were very few in number. These numbers gave us a good starting point to decide how deep we need to analyze hosts and paths for profiling.

Figure 71 shows the shape of the total 1,138,923,169 unique *HxPx Keys* of *Arquivo.pt*'s current index put together in the form of a tree as the *URI Key* space changes on each host and path depth. The tree is broken down in host and path segments (i.e., Figure 71a and 71b) instead of one continuous tree and the latter is scaled down 70 times as compared with the host segment to ensure that the shape of path segment is distinguishable from one depth to the next. In the host segment, at each host level (after *H1*) a significant portion leads to *P0* (i.e., root path), but the remainder has further children host segments (i.e., sub-domains). Figure 71a shows that hostnames with depth more than four (i.e., *H5* and beyond) are significantly small in number. In the path segment, at each level a significant portion terminates, but the remainder branches out into deeper path segments. The shape



(a) *Parents* and *Children* at each *Host* depth. All the terminating host nodes at each level lead to the root path (i.e., *P0*) shown at the bottom.



(b) *Parents* and *Children* at each *Path* depth. The root path (i.e., *P0*) shown at the top is scaled 70 times down as compared with the bottom row of the *Host* segment tree.

Fig. 71. The shape of *HxPx Key* tree of *Arquivo.pt*. Labels on the left denote *Host* and *Path* depths. Corresponding pair of labels on the right denote number of *Parents* and *Children* respectively. Darker nodes have higher number of *Mean Children*. *Host* and *Path* segments are plotted separately with different scales while the bottom row of the *Host* segment corresponds to the top row of the *Path* segment.

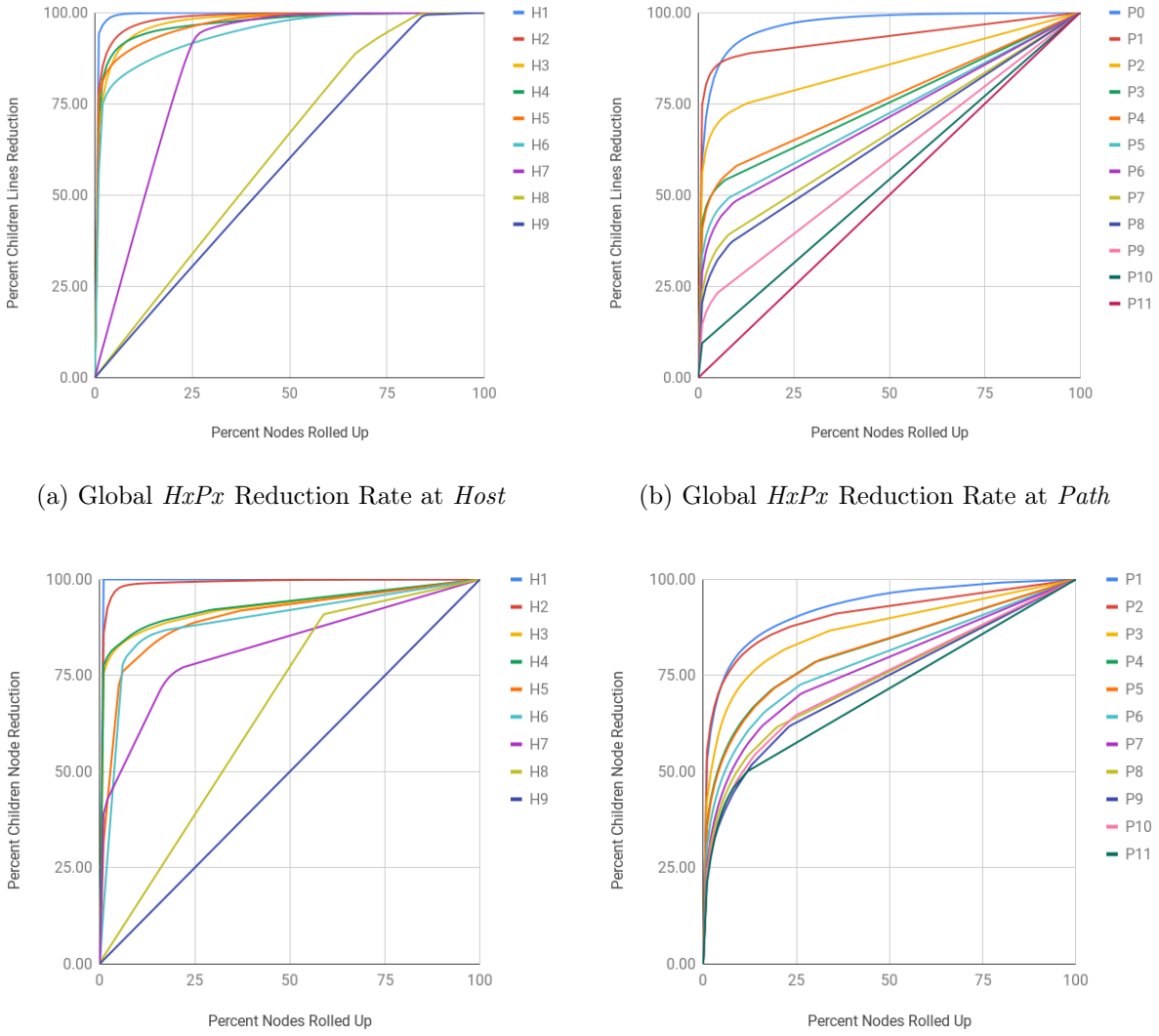
of the path segment in Figure 71b shows that the tree starts to shrink from *P4* and the bulk tree is around *P3*. Any effort to reduce the *URI Key* space near this level can significantly reduce the *Relative Cost*.

Table 25 is based on the total 1,138,923,169 unique *HxPx Keys* of *Arquivo.pt*'s current index. For example, the *H3* (see Figure 57b for naming convention) row means there are a total of 2,158,880 unique *H3* prefixes that cover a sum of 630,309,184 *HxPx Keys* of which the most popular prefix covers 51,849,377 keys alone. The *Mean* number of keys per prefix at *H3* is 291.96 with a *Median* of 7 and *Standard Deviation* of 37,641.59. The *RedQ (Reduction Coefficient)* column represents a derived quantity that we defined as the amount of reduction in keys it would cause if *HxPx Keys* longer than a given depth are stripped off at that depth and only counted reduced unique prefixes. This can be calculated using Equation 12 at depth d where $|\text{HxPx Keys}_{\geq d}|$ is the number of *HxPx Keys* with depth $\geq d$

Table 25. *Host* and *Path* depth statistics of unique *HxPx Keys* in *Arquivo.pt*. Sorted *HxPx Keys* no shorter than a given depth are chopped at that depth, number of occurrences of these keys is *Count*, their total is *Sum*, and various other statistical measures are reported based on these numbers. The *RedQ* value is calculated using Equation 12, *Parents* is the number of non-terminal nodes of the previous depth, *Children* is the number of unique nodes at a given depth, and *MeanChld* is the average number of *Children* per *Parent*.

Depth	Count	Sum	Max	Mean	Med.	StdDev	RedQ	Parents	Children	Mean Child
H1	973	1,138,923,169	616,372,626	1,170,527.41	930	21,620,107.00	1.00000	1	973	973.00
H2	2,068,333	1,138,916,690	109,176,956	550.64	5	91,308.66	0.99818	904	2,068,333	2,287.98
H3	2,158,880	630,309,184	51,849,377	291.96	7	37,641.59	0.55153	253,091	2,158,880	8.53
H4	1,329,137	201,308,887	3,765,122	151.46	10	4,797.10	0.17559	148,589	1,329,137	8.95
H5	245,881	39,396,636	376,969	160.23	5	3,420.96	0.03438	31,635	245,881	7.77
H6	103,579	17,571,552	105,591	169.64	27	1,106.03	0.01534	16,496	103,579	6.28
H7	34,380	9,636,427	19,572	280.29	20	450.16	0.00843	10,061	34,380	3.42
H8	69,829	6,383,484	535	91.42	120	45.75	0.00554	15,359	69,829	4.55
H9	55,811	2,660,591	80	47.67	56	19.6	0.00229	55,811	55,811	1.00
H10+	10	62	19	6.20	2	6.51	0.00000	10	10	1.00
P0	5,841,503	1,138,923,169	2,264,623	194.97	7	3,059.43	0.99487	5,841,503	5,841,503	1.00
P1	145,687,459	1,134,466,338	2,242,344	7.79	1	376.64	0.86817	5,828,059	145,687,459	25.00
P2	290,761,965	1,021,443,935	603,840	3.51	1	130.76	0.64156	40,130,355	290,761,965	7.25
P3	392,635,328	795,954,162	565,043	2.03	1	78.14	0.35412	79,234,027	392,635,328	4.96
P4	215,251,988	461,498,975	512,098	2.14	1	80.01	0.21621	66,059,544	215,251,988	3.26
P5	158,256,277	287,069,088	512,098	1.81	1	65.72	0.11310	48,163,114	158,256,277	3.29
P6	91,334,214	159,584,909	50,384	1.75	1	22.3	0.05993	33,776,599	91,334,214	2.70
P7	60,099,825	91,006,216	44,114	1.51	1	17.24	0.02714	24,201,781	60,099,825	2.48
P8	31,101,768	45,186,916	24,631	1.45	1	15.54	0.01237	14,890,308	31,101,768	2.09
P9	18,601,197	23,008,116	10,247	1.24	1	9.74	0.00387	9,233,634	18,601,197	2.01
P10	6,817,122	7,455,014	5,858	1.09	1	9.36	0.00056	3,206,260	6,817,122	2.13
P11+	858,772	858,856	2	1.00	1	0.01	0.00000	222,432	392,565	1.76

and $|\text{URI Keys}_d|$ is the number of unique partial *URI Keys* stripped at depth d (reported under the *Sum* and *Count* columns of Table 25 respectively). Figures 72a and 72b show the cumulative reduction as the top most frequent keys are rolled up at a host and path depth respectively. Furthermore, there are 253,091 nodes in the tree one depth above (i.e., *H2*) that lead to 2,158,880 nodes at the current depth. While the *Mean Child* count at *H3* is 8.53, the distribution is not uniform. Figures 72c and 72d show the cumulative reduction in immediate children count as the most popular parents leading to the current depth are rolled up incrementally from bottom up. The purpose of the *Reduction Coefficient* is to understand the impact and importance of various host and path depths globally while the *Mean Child* count gives an estimate of a more localized impact at a given depth. For this



(a) Global *HxPx* Reduction Rate at *Host*

(b) Global *HxPx* Reduction Rate at *Path*

(c) Incremental *Host* Children Reduction

(d) Incremental *Path* Children Reduction

Fig. 72. Global and Incremental *Host* and *Path* Segment Reduction. Global reductions describe the change in the total number of *HxPx Keys* (or the size of sub-trees) when keys are rolled up at a given *Host* or *Path* depth. Incremental children reductions describe the change caused by roll ups of immediate children nodes into their corresponding parent nodes at a given *Host* or *Path* depth. Nodes with larger sub-trees and children counts in the two cases respectively are rolled up first.

work we have used the latter as a factor to decide when to roll a sub-tree up while compacting a *MementoMap*. Rolling the sub-tree up at *H1*, *H2*, and *P0* are not applicable for evaluation

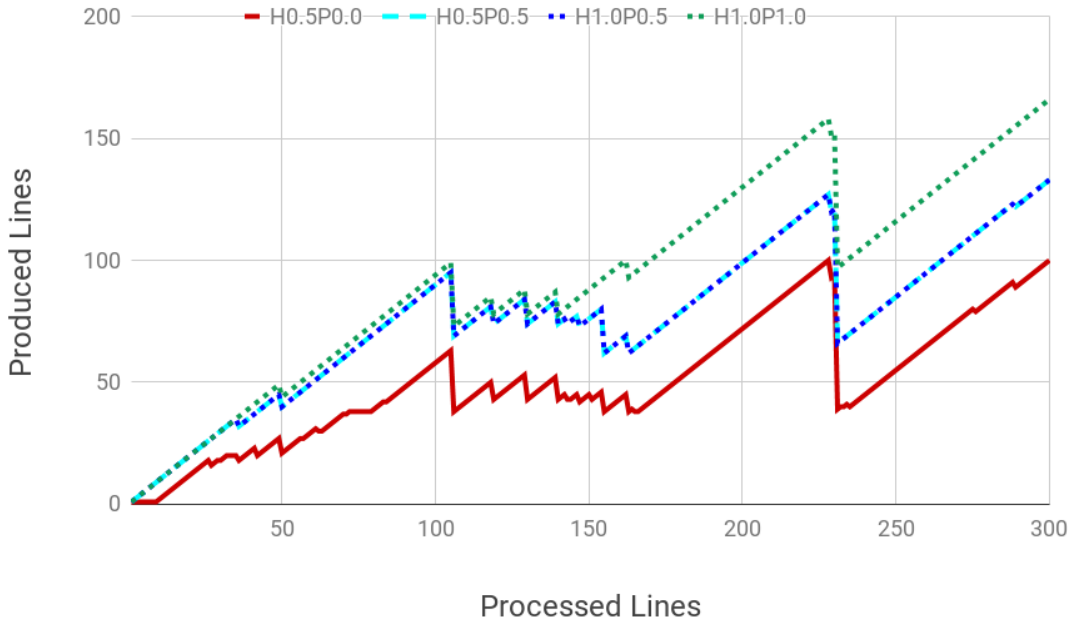


Fig. 73. Growth of compacted *MementoMap* vs. lines processed from an input *MementoMap*. This plot illustrates a very small portion of the entire process to highlight the compaction behavior at a micro level. The size of the output *MementoMap* decreases each time a roll up happens. A roll up at smaller depth often reduces the size more significantly.

here because *H1* means shrinking everything into a single record of "*" key, *H2* would require out-of-band information (because not every *TLD* is equally popular), and *P0* being the root of the path has nothing to roll up into (though compaction might happen in the relevant host segment independently). We fit the remaining values of *Mean Child* count on Power Law [84] curves (other curve fittings are also possible) for both host and path segments to find a and k parameters and use these empirical values for compaction decision making.

$$\text{RedQ}_d = \frac{|\text{HxPx Keys}_{\geq d}| - |\text{URI Keys}_d|}{|\text{HxPx Keys}|} \tag{12}$$

8.5.4 MEMENTOMAP COST AND ACCURACY

Web archives are messy collections that contain many malformed records often caused by configuration issues in web servers, poorly written web applications, bugs in archiving tools, incompatible file transformations, or even security vulnerabilities [18]. Archive profiling can

Table 26. *MementoMap* generation, compaction, and lookup statistics for *Arquivo.pt*. Output of one step is used as the input of the next step in a chain as the next step has at least one smaller weight. The first record was created using some *Linux* commands instead of the script, that is why some values are reported as *N/A*.

Input	W_h	W_p	**Lines**	**Size (bytes)**	**Gzipped (MB)**	**Rollups**	**Time (sec)**	**RelCost**	**Accuracy**
CDXJ	∞	∞	447,107,301	30,753,644,382	3,449	N/A	N/A	0.464	0.946
$H_{\infty}P_{\infty}$	4.00	4.00	27,010,037	1,443,292,676	218	4,574,305	8,643	0.028	0.646
$H_{4.00}P_{4.00}$	4.00	2.00	14,143,676	662,171,623	119	703,394	507	0.015	0.600
$H_{4.00}P_{2.00}$	4.00	1.00	7,528,548	315,946,553	63	537,341	264	0.008	0.539
$H_{4.00}P_{1.00}$	4.00	0.50	4,269,344	162,132,599	35	483,779	151	0.004	0.482
$H_{4.00}P_{0.50}$	4.00	0.25	3,054,686	107,784,353	24	411,843	87	0.003	0.426
$H_{4.00}P_{0.25}$	4.00	0.00	1,673,784	40,446,417	11	1,411,579	70	0.002	0.275
$H_{4.00}P_{4.00}$	2.00	4.00	24,937,984	1,316,371,599	205	9,572	500	0.026	0.626
$H_{2.00}P_{4.00}$	2.00	2.00	12,867,647	585,142,758	111	669,670	468	0.013	0.588
$H_{2.00}P_{2.00}$	2.00	1.00	6,584,376	257,905,766	58	512,413	241	0.007	0.525
$H_{2.00}P_{1.00}$	2.00	0.50	3,615,997	121,452,813	32	458,681	124	0.004	0.472
$H_{2.00}P_{0.50}$	2.00	0.25	2,542,869	76,274,453	21	349,700	70	0.003	0.422
$H_{2.00}P_{0.25}$	2.00	0.00	1,529,328	33,658,544	10	1,171,377	56	0.002	0.270
$H_{2.00}P_{4.00}$	1.00	4.00	23,840,710	1,252,548,065	196	4,671	466	0.025	0.581
$H_{1.00}P_{4.00}$	1.00	2.00	12,313,036	555,628,348	107	640,163	448	0.013	0.549
$H_{1.00}P_{2.00}$	1.00	1.00	6,307,180	244,402,690	56	489,942	232	0.007	0.501
$H_{1.00}P_{1.00}$	1.00	0.50	3,465,689	114,755,789	30	439,647	116	0.004	0.453
$H_{1.00}P_{0.50}$	1.00	0.25	2,437,451	71,797,863	20	333,087	67	0.003	0.403
$H_{1.00}P_{0.25}$	1.00	0.00	1,474,541	31,881,496	10	1,117,830	58	0.002	0.261
$H_{1.00}P_{4.00}$	0.50	4.00	22,315,969	1,162,107,385	184	6,516	447	0.023	0.540
$H_{0.50}P_{4.00}$	0.50	2.00	11,729,408	525,115,243	101	594,779	420	0.012	0.520
$H_{0.50}P_{2.00}$	0.50	1.00	6,056,959	232,945,804	53	461,516	218	0.006	0.476
$H_{0.50}P_{1.00}$	0.50	0.50	3,342,092	109,798,250	29	417,912	112	0.003	0.433
$H_{0.50}P_{0.50}$	0.50	0.25	2,358,976	68,957,985	20	316,782	65	0.002	0.388
$H_{0.50}P_{0.25}$	0.50	0.00	1,434,084	30,800,396	9	1,071,071	51	0.001	0.253
$H_{0.50}P_{4.00}$	0.25	4.00	21,197,676	1,096,034,790	174	9,533	416	0.022	0.511
$H_{0.25}P_{4.00}$	0.25	2.00	11,217,682	498,573,523	97	558,528	392	0.012	0.495
$H_{0.25}P_{2.00}$	0.25	1.00	5,842,652	223,237,207	51	435,916	204	0.006	0.461
$H_{0.25}P_{1.00}$	0.25	0.50	3,241,589	105,791,213	28	398,097	109	0.003	0.420
$H_{0.25}P_{0.50}$	0.25	0.25	2,298,413	66,763,014	19	302,762	64	0.002	0.377
$H_{0.25}P_{0.25}$	0.25	0.00	1,404,993	30,018,340	9	1,031,775	53	0.001	0.249
$H_{0.25}P_{4.00}$	0.00	4.00	17,391,655	882,144,079	142	118,082	392	0.018	0.391
$H_{0.00}P_{4.00}$	0.00	2.00	9,453,810	410,205,661	81	560,039	324	0.010	0.385
$H_{0.00}P_{2.00}$	0.00	1.00	5,054,662	187,327,280	43	471,696	179	0.005	0.373
$H_{0.00}P_{1.00}$	0.00	0.50	2,901,796	91,419,782	25	440,818	95	0.003	0.354
$H_{0.00}P_{0.50}$	0.00	0.25	2,107,245	59,036,815	17	366,330	57	0.002	0.326
$H_{0.00}P_{0.25}$	0.00	0.00	1,339,475	27,946,167	8	986,664	48	0.001	0.236

uncover some of these as we found many malformed *MIME-Type* [11] and *Status Code* [12] entries in *Arquivo.pt*.

To run our experiments we decided to filter only the clean records out from these *CDXJ* files. We further limited our scope to only *HTML* pages that returned a `200` status code. Additionally, we excluded any `robots.txt` and `sitemap.xml` files that were served wrongly as "`text/html`". With these filters in place we reduced mementos by almost half of the total index size to only 2,671,653,766. Now, there are 962,832,513 filtered unique *URI-Rs*, which means the γ value is increased slightly to 2.77. Also, the *HxPx Keys* count is reduced to 447,107,301, which is 39% of the overall number. From these keys we created the baseline *MementoMap* with compressed file size of 3.4G (as shown in the first record of Table 26) which is already reduced to 1.3% of the original index size. This baseline *MementoMap* has 46.4% *Relative Cost* (i.e., the ratio of reduced number of unique lookup keys vs. number of unique *URI-Rs*) that yields 94.6% *Accuracy*.

In the next step we supplied this baseline *MementoMap* as input for compaction with host and path compaction weights $W_h = 4.00$ and $W_p = 4.0$ respectively. These weights are multiplied by their corresponding estimated *Mean Child* value at each depth to find the cutoff number when the sub-tree is to be rolled up. A small weight will roll the sub-tree up more aggressively than a large value, resulting in a more compact *MementoMap*. This process produced a *MementoMap* with only 27,010,037 lines (i.e., 6.0% of the baseline or 2.8% *Relative Cost*) after going through 4,574,305 recursive roll ups. The process took 2.4 hours to complete on our Network File System (NFS) storage. The time taken to complete the compaction process is a function of the number of lines to process from the input, number of lines to be written out, and the number of roll ups to occur (along with the read and write speeds of the disk). Since the process is I/O intensive, using faster storage can reduce the time significantly, which we verified by repeating the experiment on *TMPFS* [231]. We generated 36 variations of *MementoMaps* with all possible pairs of W_h and W_p weights from values 4.00, 2.00, 1.00, 0.50, 0.25, and 0.00 as shown in Table 26. To generate *MementoMaps* with smaller weights we used *MementoMaps* of immediate larger weight pairs as inputs (e.g., input one with $W_h = 2.00, W_p = 0.50$ to generate one with $W_h = 1.00, W_p = 0.25$). This technique of chaining the output as input to the next step reduced the generation time for subsequent *MementoMaps* from hours to a few minutes and also illustrated that *MementoMaps* can easily be compacted further when needed.

Figure 73 shows a portion of the roll up activity during the compaction process. The size of the output grows linearly, but on a micro-scale whenever there is a roll up activity,

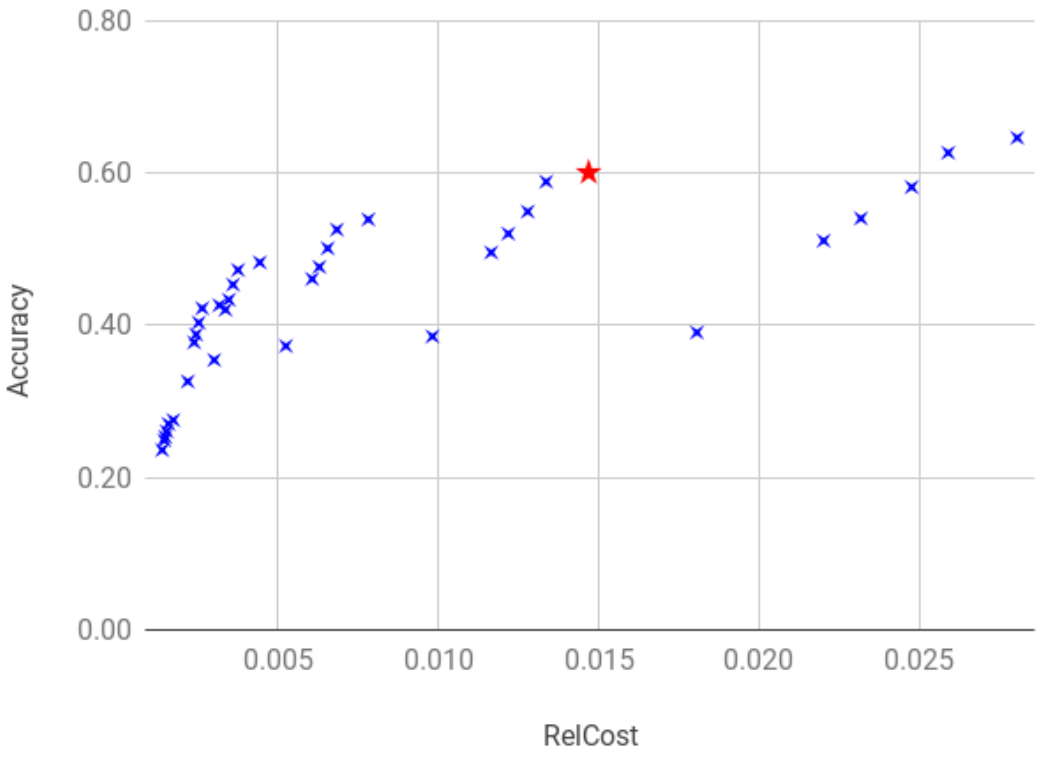


Fig. 74. *Relative Cost* vs. *Lookup Routing Accuracy.* A *MementoMap* generated/compacted using $W_h = 4.00$ and $W_p = 2.0$ yielded 60% *Accuracy* with only 1.5% *Relative Cost* (highlighted with the red star).

the output size goes down depending on at what depth roll up happened and how big of a sub-tree was affected.

Finally, we used *MemGator* logs to perform lookups in these 36 *MementoMaps* generated with different host and path weight pairs to see how well they perform. Figure 74 shows the *Relative Cost* and corresponding *Lookup Routing Accuracy* of these *MementoMaps*. The *Accuracy* here is defined as the ratio of correctly identified *URIs* for their presence or absence vs. all the lookup *URIs*. In this experiment *MementoMaps* with weights $W_h = 4.00, W_p = 2.00$ and $W_h = 2.00, W_p = 2.00$ yielded about 60% *Routing Accuracy* with less 1.5% *Relative Cost* without any false negatives (i.e., 100% *Recall*). Since *Arquivo.pt* had only a 3.35% hit rate in the past three years, *MemGator* could have avoided almost 60% of the wasted traffic to *Arquivo.pt* without missing any good results if *Arquivo.pt* were to advertise its holdings via a small *MementoMap* of about 111MB in size. The accuracy can further be improved by

1) exploring other optimal configurations for sub-tree pruning, 2) generating *MementoMaps* with the full index, not just a sample, and 3) including entries for absent resources from the "Zero" row of the Figure 68.

8.6 CHAPTER SUMMARY

In this chapter we first described UKVS, the file format we use for serialization of *MementoMaps*. We illustrated a list of sorted SURTs with wildcards and organized them in a tree to elaborate on the structure of *URI-Keys*. We then described various structures and attributes of *MementoMap* files suitable for different needs.

We discussed a reference implementation that allows generation and compaction of *MementoMaps* as well as fast lookup in them. Then we described a few standard means of disseminating generated *MementoMaps* by archives or third parties.

To evaluate the serialization format we used and its effectiveness we used the complete index of *Arquivo.pt* and our *MemGator* service logs of over three years. We measured the overlap of archived vs. accessed resources. We then analyzed the holdings of *Arquivo.pt* in detail and the shape of their URI tree. Finally, we compared the cost and routing accuracy of these *MementoMaps* and found that a *MementoMap* of less than 1.5% Relative Cost can correctly identify the presence or absence of 60% of the lookup URIs in the corresponding archive without any false negatives. We identified that inclusion of an absence list (i.e., an archival voids profile) will improve the accuracy significantly.

CHAPTER 9

MEMENTO ROUTING

In the past three chapters we established the *MementoMap* framework to discover and quantify archival holdings and voids for different URI spaces. Moreover, we established an efficient and flexible means of serializing this information for dissemination. We evaluated those *MementoMaps* for individual web archives without allowing any false negatives. Now, we can leverage these serialized archive profiles (MementoMaps) to make routing decisions by a Memento aggregator for multiple web archives together. This chapter addresses our third research question, "***RQ3**: How to utilize archive profiles for the routing of URI lookup requests?*"

9.1 MEMENTO AGGREGATION AND ROUTING

It is inconvenient and impractical for users to perform URI lookups in many different public web archives one by one to find any or the most suitable Memento of a resource closest to a given time in the past. It is a tedious task to repeatedly perform the same lookup in tens of public web archives. Moreover, users might not be aware of all the public web archives available at a given time as new web archives come to existence and existing ones disappear or move to other domains every now and then [42, 43, 44, 45, 46]. To address this issues there exist services and tools that provide a unified interface to interact with many different public web archives at once using their Memento API. This practice is called *Memento Aggregation* and such tools/services are called *Memento Aggregators* (as described in Section 2.5 of Chapter 2). LANL runs a public Memento Aggregator called the "Time Travel Service" and we developed and released an open-source Memento Aggregator called "MemGator" that allows people to run their own Memento Aggregators as a CLI tool or a web service.

Memento Aggregators are configured to aggregate a set of known public web archives. In the most basic form an aggregator might poll all members of the set of web archives for every single lookup request. This naive approach is called broadcasting and is wasteful as only a few web archives return any good results for a given lookup URI. Though, the subset of web archives that return good results can be different for different lookup URIs. In

Chapter 1 we gave examples to establish that broadcasting can be problematic for small web archives with limited capacity to serve web requests, which they would not want to waste for lookup requests that are not in their collection scope. Moreover, another downside of broadcasting is that the client needs to wait for responses from all the web archives, which sometimes causes a long delay by some archives that eventually do not return any good results. Identifying potentially suitable candidate archives for a given lookup URI and aggregating archives selectively is called *Memento Routing*. Our *MementoMap* framework enables this capability by profiling web archives and creating a high level summary of their archival holdings and/or archival voids.

9.2 METHODOLOGY

In this chapter our goal is to assign normalized scores to each web archive from a given set of web archives for a given lookup URI. Based on those scores, an application (such as a Memento aggregator) can decide which subset of archives it wants to poll for the lookup URI. An application may choose to route a lookup request to the top-k archives or every archive above a certain threshold.

We first define a *density* score that is derived from the *frequency* value reported in the *MementoMap* as described in Section 9.2.1. This represents how densely a subtree of the URI space (represented by a URI keys) is archived in a web archive. Then we define a *closeness* score between a lookup URI key and the longest matching URI key prefix in a *MementoMap* as described in Section 9.2.2. This score depends on the length of the matching URI key prefix and the difference in lengths of the two keys. Finally, we define a *routing* score as a product of these two scores as described in Section 9.2.3. We address many special cases as well. Moreover, we describe both uniform and weighted normalized relative routing scores when multiple web archives are combined for ranking.

We describe an inverted index with examples and how we use it to consolidate *MementoMaps* from multiple web archives for efficient and scalable lookup. Finally, we illustrate how we search the inverted index, extract matching records, and transform them into a consumable result for a Memento aggregator.

9.2.1 DENSITY SCORE

We define *density* as a normalized score that captures the essence of how much of the archival efforts in a web archive are concentrated around a given *URI Key* relative to the overall size of the web archive. It is the logarithmic ratio of number of URI-Rs under a

given *URI Key* in a web archive to the total number of URI-Rs in the web archive. This value ranges from "0" to "1" (inclusive of both, i.e., $0 \leq \mu \leq 1$) where a value of "0" means URIs corresponding to the key are not archived and a value of "1" means all the archiving activity is concentrated to URIs under the scope of the URI key. We denote *density* by $\mu_{a,k}$ as shown in Equation 13 for a key "k" in an archive "a" with a set of URI-Rs "R_a".

$$\mu_{a,k} = \frac{\log(1 + r_{a,k})}{\log(1 + |R_a|)} \tag{13}$$

The quantity $r_{a,k}$ is an exact or approximate value of the number of URI-Rs under the URI key "k" in the archive "a". It is calculated based on the *frequency* value (as described in Section 8.2.2 of Chapter 8) returned from the *MementoMap* lookup as shown in Equation 14. The *frequency* value has the `[<urim-count>]/[<urir-count>]` form. The *frequency* value may also contain one of the three approximation notations when the reported values are not exact, but we ignore these notations for *density* estimation because their primary purpose is to minimize the number of insignificant changes in the *MementoMap*. For *density* estimation we are interested in the URI-R count. However, it is possible that for a given URI Key either the URI-R count (C_r) or the URI-M count (C_m) is missing. If the URI-R count is missing, we can estimate it based on the reported URI-M count with the help of γ vale as defined earlier in Equation 2 in Chapter 6. It is worth noting that if there is an exact match of the lookup key (i.e., not a partial wildcard match) then the URI-R count is "1". If γ is not known for an archive, a default value between "2" and "3" can be used because we found that many web archives have the average URI-M to URI-R ratio close to this range (e.g., for the Internet Archive $\gamma = 2.1$ and for Arquivo.pt $\gamma = 2.5$).

$$r_{a,k} = \begin{cases} 1, & \text{if there is an exact match of the lookup key} \\ C_r, & \text{if the URI-R count is present} \\ \frac{C_m}{\gamma}, & \text{otherwise} \end{cases} \tag{14}$$

We are using a log scale for both the numerator and denominator in Equation 13 to make sure large archives with diverse set of URIs are not punished. Moreover, we are adding "1" in both the log function inputs to avoid the condition of undefined "$log(0)$". Also, this addition of "1" nicely translates to a *density* value of "0" for archival voids that are represented with a *frequency* value of "0" as described in Chapters 7 and 8.

Suppose we perform a lookup for "`org,example)/a/b/c/d/e`" in a *MementoMap* of a web archive that has over a million unique URI-Rs in it. Suppose we find the longest prefix match

in the *MementoMap* as "`org,example)/a/b/c/*`". We know it is not an exact match, so there might be multiple URI-Rs present in the archive with prefix "`org,example)/a/b/c/`". If the *MementoMap* returns a *frequency* of "200/100" or "/100" then we know there are 100 URI-Rs under the matching URI key prefix (we can ignore the URI-M count when URI-Rs counts are reported). However, if the *frequency* value is "200" or "200/" then we only know the URI-M count and need to estimate the number of URI-Rs for the matching URI key prefix. In this case we can divide the URI-M count with the value of γ (the average number of URI-Ms per URI-R for the archive, say "2") to get an estimated number of URI-Rs (say, "100"). This means we get a *density* score $\mu = \log(1 + 100)/\log(1 + 1000000) \approx 0.334$ for the URI key in the archive.

9.2.2 CLOSENESS SCORE

We define *closeness* as a normalized score to describe how closely a returned *MementoMap* URI key matches with the lookup URI (in SURT format). We denote *closeness* by $\chi_{l,k}$ as shown in Equation 15 for a lookup URI "l" and corresponding returned URI key "k" in a given *MementoMap*. The quantities L_l and L_k are representing the number of segments (both host and paths combined) in the lookup URI and the returned URI key, respectively. The difference in lengths of the two SURTs (l and k) $L_l - L_k$ will be zero if there is an exact match or the only additional segment in the URI key is a wildcard character "`*`" (which represents zero or more segments), otherwise a positive integer (because $L_l \geq L_k$ as a matching prefix key cannot be longer than the lookup URI). The larger the difference (in denominator) is, the poorer the match is, resulting in a smaller *closeness* score. By adding "1" in the denominator we ensure that we avoid division by zero and at the same time keeping the *closeness* value between "0" and "1" (inclusive of both, i.e., $0 \leq \chi \leq 1$). We also want the length of the longest matching URI key prefix (L_k) to play a role in the *closeness* score, but effect should not be as prominent as the difference in lengths. Also, as we go from left to right in the a SURT, the importance of segments goes down as they become more specific. For these two reasons, we used a log scale in the numerator. We added "1" in the argument of the log function to avoid the condition of undefined "$log(0)$" when there is no matching key returned (i.e., L_k is "0"). Moreover, this addition of "1" nicely translates to a *closeness* value of "0" when there is no matching key found in the *MementoMap*. To keep the *closeness* value normalized (i.e., between zero and one) we make sure the numerator never goes above "1" as the smallest value the denominator can have is "1". For this reason, we use a *min* function with one of its arguments as "1". This condition will arrive when the length of the

URI key is greater than or equal to the base "b" of the log function. A suitable value of "b" depends on the longest URI keys in a *MementoMap* that represent a significant portion of the *MementoMap* (i.e., excluding any long but rare keys). From Chapter 8 we learned that the bulk of URIs (about 95%) from a web archive can be represented in a *MementoMap* with keys smaller than 10 segments (Figure 71 on Page 157 shows that there are very few URIs with host segments > 4 and paths segments > 6), which means a log of base 10 (i.e., $b = 10$) is a suitable choice for the general case.

$$\chi_{l,k} = \frac{\min(\log_b(1 + L_k), 1)}{1 + L_l - L_k} \tag{15}$$

Suppose we perform a lookup for the key "`org,example)/a/b/c/d/e`" (i.e., $L_l = 7$) in a *MementoMap* and find the matching URI key prefix "`org,example)/a/b/c/*`" (i.e., $L_k = 5$). This means we get a *closeness* score $\chi = \min(\log(1 + 5), 1)/(1 + 7 - 5) \approx 0.259$ between the lookup key and corresponding matching URI key prefix.

In our TimeMap lookup use case the URI key is always going to be a prefix of the lookup URI key. However, this *closeness* score can be generalized for any two independent URIs if a use case emerges in the future. We first convert the two URIs to their SURT forms (say, "s_1" and 's_2'), then identify the longest common URI key prefix between them (say, "k"), and then take the product of the *closeness* scores of the two SURTs against their common prefix (i.e., "$\chi_{s_1,k} \cdot \chi_{s_2,k}$") while using the length of the longest of the two SURTs as the base of the log (i.e., "$b = \max(L_{s_1}, L_{s_2})$"). This way, the product of the *closeness* scores will be a "1" if the two URIs (or their SURTs) are identical, a "0" if their TLDs are different, and a fraction between the two otherwise.

Suppose we want to find the *closeness* score between "`org,example)/a/b/c/d/e`" (i.e., $L_{s_1} = 7$) and "`org,example)/a/x/y`" (i.e., $L_{s_2} = 5$). The longest length of the two SURTs to serve as the base of the log function is 7 (i.e., "$\max(7, 5)$"). The common prefix in this case is "`org,example)/a`" (i.e., $L_k = 3$). In this case the *closeness* scores of "s_1" and "s_2" to the common prefix "k" would be $\chi_1 = \min(\log_7(1 + 3), 1)/(1 + 7 - 3) \approx 0.142$ and $\chi_2 = \min(\log_7(1 + 3), 1)/(1 + 5 - 3) \approx 0.237$, respectively. This means we get the combined *closeness* score $\chi = \chi_1 \cdot \chi_2 \approx 0.142 \cdot 0.237 \approx 0.034$ for the two URIs.

9.2.3 ROUTING SCORE

With the key metrics *density* μ and *closeness* χ described above, we define the *routing* score as a normalized score to assess whether a lookup URI is present in an archive with a *MementoMap* (i.e., whether the request should be routed to the archive). We denote the

routing score with $\rho_{a,l,k}$ as shown in Equation 16 for the likelihood of a lookup URI "l" to be found in an archive "a" with a *MementoMap* that returns the longest matching URI key prefix "k" with the corresponding *frequency* score when queried.

$$\rho_{a,l,k} = \begin{cases} 1, & \text{if there is an exact match of the lookup key and } \mu_{a,k} > 0 \\ \mu_{a,k} \cdot \chi_{l,k}, & \text{otherwise} \end{cases} \tag{16}$$

The above *routing* score is a means to estimate the likelihood of finding a lookup URI in an archive independent of any other web archives that are being aggregated by a Memento aggregator. It is not a linear quantity (due to logarithmic metrics involved), instead, a normalized score that can be used to set a heuristic threshold for Memento routing. It is a product of two normalized scores, both often being fractions, which means it is $\leq$ the smaller of the two scores. This means if there are two web archives with *routing* scores "ρ_{a_1}" and "ρ_{a_2}" for a lookup URI, where "$\rho_{a_1} = 2\rho_{a_2}$", it does not necessarily mean that a_1 is twice as likely to return a good result as a_2, instead, it means that a_1 is more likely to return a good result than a_2. If the heuristic routing threshold falls between the two scores then the higher one will be routed and the other will not.

Suppose we perform a lookup for the key "`org,example)/a/b/c/d/e`" in a *MementoMap* of a web archive and find the matching URI key prefix "`org,example)/a/b/c/*`" with *frequency* value "200/100". The corresponding *density* score would be $\mu \approx 0.334$ (as illustrated in Section 9.2.1) and the *closeness* score would be $\chi \approx 0.259$ (as illustrated in Section 9.2.2). Since this is not an exact match, the *routing* score would be $\rho = \mu \cdot \chi \approx 0.334 \cdot 0.259 \approx 0.087$ for this lookup. If we have a cut-off threshold of "≥ 0.01" for the web archive then the lookup will be routed to it, but if the cut-off threshold is set to "≥ 0.1" then the lookup will not be routed to the web archive.

Relative and Weighted Routing Scores

If a ranked ordered list of web archives with their corresponding *normalized relative routing* scores is desired for certain applications (as illustrated in Figure 11c in Chapter 1 on Page 16) it can be calculated for each archive by dividing their *routing* score with the sum of the *routing* scores of all the aggregated archives. We denote the *normalized relative routing* score of each archive "a_i" from a set of "N" aggregated archives $a_i \in \{a_1, a_2, ..., a_N\}$ and their corresponding returned URI keys $k_i \in \{k_1, k_2, ..., k_N\}$ for a lookup URI "l" by ρ'_{a_i,l,k_i} as shown in Equation 17.

```
if archive not profiled:
  # Return a default non-zero routing score
  # if neither the holdings nor the voids of the archive are profiled
  return DEFAULT_ROUTING_SCORE

if archival holdings profiled:
  # Initialize with a routing score of 0
  # if the holdings of the archive are profiled
  # irrespective of the state of the voids profiling
  routing_score = 0
else:
  # Initialize with a routing score of 1
  # if only the voids of the archive are profiled
  routing_score = 1

if uri_key prefix match found:
  # Calculate and update the initialized routing score
  # if a corresponding URI key is found in any profile of the archive
  routing_score = calculate_routing_score(archive, lookup_uri, uri_key)

# Return the initialized/updated routing score
return routing_score
```

Fig. 75. Routing Score Calculation Procedure for Archives With Different Types of Profiles

$$\rho'_{a_i,l,k_i} = \frac{\rho_{a_i,l,k_i}}{\sum_{j=1}^{N} \rho_{a_j,l,k_j}} \tag{17}$$

It is worth noting that if a Memento aggregator does not have *MementoMaps* for all the archives it aggregates, it should assign some default *routing* scores to web archives it does not have a *MementoMap* of to make sure those archives show up in the ranked ordered list. Moreover, we can assign different weights (or importance factor) $W_i \in \{W_1, W_2, ..., W_N\}$ to each aggregated archive to calculate corresponding *normalized relative weighted routing* scores ρ''_{a_i,l,k_i} as shown in Equation 18.

$$\rho''_{a_i,l,k_i} = \frac{W_i \cdot \rho_{a_i,l,k_i}}{\sum_{j=1}^{N} W_j \cdot \rho_{a_j,l,k_j}} \tag{18}$$

It is not necessary to have either of the holdings or voids profiled for each web archive being aggregated by a Memento Aggregator. Moreover, not every lookup URI will have a matching URI key prefix in a *MementoMap*. However, in order to create a ranked ordered list of archives for a given lookup URI we must assign some *routing* scores to each archive. In Figure 75 we illustrate a procedure to achieve this. This procedure returns a default *routing* score for web archives that are not profiled yet. If an archive's holdings are profiled (irrespective of whether its voids are profiled), we initialize the *routing* score with "0" so that an appropriate score is assigned only if a matching URI key is found. However, if only the voids of an archive are profiled then we initialize the *routing* score with "1", which will be changed to "0" later if a matching URI key is found. The latter condition is suitable for big web archives (such as the Internet Archive) that return good results for the most lookups, but want to avoid frequently requested resources that they do not contain or not willing to return as per their policies. The function `calculate_routing_score` in the pseudo code is not illustrated as it refers to the *routing* score calculation as described in Equation 16.

9.2.4 INVERTED INDEX

We take the analogy of documents and inverted indexes as described in Section 2.9 of Chapter 2 and apply it to our *MementoMap* framework for Memento routing. In our case a collection is a set of web archives that are aggregated by a Memento aggregator, each web archive represented by its corresponding *MementoMap* is a document, URI-Rs in the archive are words that are canonicalized and stemmed to form URI keys of *MementoMaps*, and their corresponding *frequency* values as term frequencies. We may choose to transform the *frequency* values from *MementoMaps* in a more compact and normalized form *density* (as described in Section 9.2.1) before adding to an inverted index. Suppose, we have a set of "N" archives ($\{a_1, a_2, ...a_N\}$) that each contain a subset of URI keys from a set of "M" keys ($\{k_1, k_2, ...k_N\}$) with *density* scores corresponding each archive and URI key pair from a set of densities ($\{\mu_1, \mu_2, \mu_3...\}$). A sample inverted index for these web archives is illustrated in Figure 76.

9.2.5 MEMENTOMAP AND INVERTED INDEX LOOKUP

In Chapter 8 we introduced a file format called UKVS and used it to serialize *MementoMaps* of web archives. This format is sort friendly, which when used for *MementoMaps*, allows binary search in URI keys on disk. This means if a Memento aggregator is aggregating a small number of web archives, it can perform searches in *MementoMaps* of each

```
1 k1: [(a1, μ1), (a2, μ2)]
2 k2: [(a2, μ3)]
3 k3: [(a1, μ4), (a2, μ5), (a3, μ6)]
```

Fig. 76. A Sample Inverted Index of Web Archive Profiles

archive in parallel to identify which archives are likely to return good results for the lookup URI.

However, this parallel search is not scalable beyond a small number of archives. To work on a large set of web archives we can process their *MementoMaps* and make a combined inverted index from them. Moreover, we can leverage our UKVS file format to serialize inverted index as well, so that we get all the benefits of arbitrary split and merge, binary search on disk, and incremental updates that have for *MementoMaps*. Additionally, if an archive has published a more detailed *MementoMap* than needed by an application (e.g., a Memento aggregator), then it is possible for the application to abridge the *MementoMap* when ingesting it for an inverted index using the *MementoMap* Compaction algorithm described in Chapter 8. An inverted index works for small number of archives as well, but its benefits would be more visible at large scale.

Suppose we have three *MementoMaps* from three different archives (as illustrated in Figure 77):

- `arquivo.pt` – with both of its archival holdings and voids profiled
- `perma.cc` – with only its archival holdings profiled
- `archive.is` – with only its archival voids profiled

We can combine these to create an inverted index using UKVS format in many different ways as shown in Figure 78. Numeric scores in the inverted index files do not match with that in the *MementoMap* files because the value column on *MementoMap* shows the *frequency* score while we have transformed it to the *density* score in inverted indexes. These keys and values are for illustration purpose only, they do not reflect holdings or voids of actual archives. In Figure 78a we placed the identifier of corresponding archive as a secondary key column while we collapsed this with corresponding score in the JSON block in Figure 78b. The latter format grows slower when many archives have a significant number of shared URI

```
!context ["https://oduwsdl.github.io/contexts/ukvs"]
!id {uri: "https://arquivo.pt/"}
!fields {keys: ["surt"], values: ["frequency"]}
!meta {type: "MementoMap", profile: "both"}
!meta {updated_at: "2020-02-05T21:49:03Z"}
com,example)/          200
com,facebook)/*        0
org,example)/a/*       20
```

(a) A Sample *MementoMap* from Arquivo.pt With Both Holdings and Voids Profiles

```
!context ["https://oduwsdl.github.io/contexts/ukvs"]
!id {uri: "https://perma.cc/"}
!fields {keys: ["surt"], values: ["frequency"]}
!meta {type: "MementoMap", profile: "holdings"}
!meta {updated_at: "2020-04-22T05:11:09Z"}
com,example)/          100
org,example)/a/b/c/*   10
```

(b) A Sample *MementoMap* from Perma.cc With Only Holdings Profile

```
!context ["https://oduwsdl.github.io/contexts/ukvs"]
!id {uri: "https://archive.is/"}
!fields {keys: ["surt"], values: ["frequency"]}
!meta {type: "MementoMap", profile: "voids"}
!meta {updated_at: "2019-11-14T10:26:32Z"}
com,example)/          0
org,example)/a/*       0
```

(c) A Sample *MementoMap* from Archive.is With Only Voids Profile

Fig. 77. *MementoMap* Samples from Different Web Archives

keys. Both of these inverted index formats benefit from the spatial locality on disk when performing binary search for a lookup URI.

Once we have an inverted index in place, we can perform lookups in it to find records to estimate the likelihood of finding a lookup URI (e.g., "`http://example.org/a/b/c/d/e?x=y`") in many different web archives. We first transform the lookup URI to SURT and remove any query parameters from it to form a lookup key (e.g., "`org,example)/a/b/c/d/e`"). We then perform a binary search in the inverted index to fetch all the records related to the longest URI key prefix matches for each archive. In a sorted inverted index these records are collocated (e.g., the last two lines of Figure 78b), making it easy and efficient to perform

```
!context ["https://oduwsdl.github.io/contexts/ukvs"]
!id {uri: "https://memgator.cs.odu.edu/"}
!fields {keys: ["surt", "archive"], values: ["density"]}
!meta {type: "InvertedIndex"}
!meta {updated_at: "2020-07-11T21:38:19Z"}
com,example)/          archive.is  0
com,example)/          arquivo.pt  0.2
com,example)/          perma.cc    0.08
com,facebook)/*        arquivo.pt  0
org,example)/a/*       archive.is  0
org,example)/a/*       arquivo.pt  0.02
org,example)/a/b/c/*   perma.cc    0.01
```

(a) Inverted Index With the Archive Secondary Key Column and Density Score as the Value Column

```
!context ["https://oduwsdl.github.io/contexts/ukvs"]
!id {uri: "https://memgator.cs.odu.edu/"}
!fields {keys: ["surt"], values: []}
!meta {type: "InvertedIndex"}
!meta {updated_at: "2020-07-11T21:44:25Z"}
com,example)/         {"archive.is": 0, "arquivo.pt": 0.2, "perma.cc": 0.08}
com,facebook)/*       {"arquivo.pt": 0}
org,example)/a/*      {"archive.is": 0, "arquivo.pt": 0.02}
org,example)/a/b/c/* {"perma.cc": 0.01}
```

(b) Inverted Index With the Archive Key and Density Score Collapsed in the JSON Block

Fig. 78. Variations of Inverted Indexes in UKVS Format from the Same Set of *MementoMaps*

the search. After ingesting all the matching records, we calculate many derived attributes and metricses. Finally, we return a composite result as illustrated in Figure 79. This figure includes a default *routing* score of 0.8 for `archive.org` which is not profiled and is not part of the inverted index shown in Figure 78. The list of archives with their resulting properties is sorted in the descending order of the *routing* score, as a result `archive.is`, which has a matching archival voids record, is placed at the end with a "0" score.

9.3 CLASSIFIER REBORN

Classifier Reborn is an open-source classifier module written in Ruby [81]. It has a simple implementation of a Bayesian classifier [129, 206] among a few other types of classifiers. This module is designed to work with string input, which it converts to tokens and counts their frequencies to perform necessary statistical calculations.

In addition to our *routing* score calculation approach described above, we wanted to use a

```
{
  "lookup_uri": "http://example.org/a/b/c/d/e?x=y",
  "lookup_key": "org,example)/a/b/c/d/e",
  "lookup_key_length": 7,
  "archives": [{
    "archive": "archive.org",
    "routing_score": 0.8,
    "relative_routing_score": 0.588
  }, {
    "archive": "perma.cc",
    "uri_key": "org,example)/a/b/c/*",
    "uri_key_length": 5,
    "keys_distance": 2,
    "density": 0.01,
    "routing_score": 0.39,
    "relative_routing_score": 0.287
  }, {
    "archive": "arquivo.pt",
    "uri_key": "org,example)/a/*",
    "uri_key_length": 3,
    "keys_distance": 4,
    "density": 0.02,
    "routing_score": 0.17,
    "relative_routing_score": 0.125
  }, {
    "archive": "archive.is",
    "uri_key": "org,example)/a/*",
    "uri_key_length": 3,
    "keys_distance": 4,
    "density": 0,
    "routing_score": 0,
    "relative_routing_score": 0
  }]
}
```


Fig. 79. A Sample Inverted Index Lookup Result

simple classifier that can take the certain basic features from the result of a *MementoMap* (or inverted index of *MementoMaps*) lookup as shown in Figure 79 (e.g., `lookup_key_length`, `uri_key_length`, `keys_distance`, and `density`) and predict whether the lookup URI is present in the corresponding web archive. We could have used one of the many off-the-shelf classifier implementations, but we decided to use Classifier Reborn instead as we could

Table 27. Datasets of Web Archival Holdings and Voids Profiles

Archive	Unique URI-Rs	Prevalence	Holdings Keys	Voids Keys
Arquivo.pt	2.0B	2.89%	14M	2.3K
Archive-It	1.9B	3.96%	13M	0.7K
UKWA	0.7B	1.18%	5.2M	0.9K
Stanford	12M	0.02%	134K	1.0K

change it more easily to fit our needs and have better understanding of its internals. In the process we contributed a significant amount of code and documentation to improve this open-source code repository. Our first major contribution to this repository was to overhaul its architecture to support modular backends to store the model instead of keeping everything in the memory of the process. We then implemented a couple of storage backends including the process memory backend (to preserve the original behavior) and an independent Redis backend [80]. This change allowed us to persist the trained model more easily, independent of the lifecycle of the training and prediction process, while the model could be built and used collaboratively by independent processes running on different hosts. Our second major contribution was the implementation of a classifier evaluation and validation module [7]. Moreover, we modified the code for our private use to make it work with the kind of input data we have for Memento routing.

9.4 EVALUATION

To evaluate our Memento routing models we used CDX datasets from four different web archives of varying sizes that we have used in the previous chapters as well. In earlier chapters we focused on establishing baselines while making sure we do not allow any false negatives. In this chapter, we do allow false negatives to see how many requests we can safe from being routed while giving up on some recall. We evaluated three approaches, two of which are based on our heuristic *routing* score: a cut-off threshold and top-k archives and third is a rudimentary machine learning model.

9.4.1 DATASETS

To create archival holdings profiles for Memento routing evaluation we reused the CDX datasets from Archive-It, UKWA, and Stanford web archives that we used for evaluation in

Chapter 6 and from Arquivo.pt that we used in Chapter 8. When generating *MementoMaps*, we used weights $W_h = 4.00$ and $W_p = 2.0$ which yielded significant accuracy with very little relative cost (as reported in Chapter 8). However, the Stanford web archive was two orders of magnitude smaller than the rest, so we decided to generate a more detailed profile for this and for that we used weights $W_h = 8.00$ and $W_p = 4.0$.

We also used access logs data from Arquivo.pt to create archival voids profile that we used in Chapter 7. To create archival voids profile we only considered URIs that were requested at least 100 times and have always returned a "`404 Not Found`" response. There were about 24,000 such URIs that turned into about 23,000 unique URI keys in the voids profile. The reduction ratio seems poor because we did not allow any roll ups in archival voids profiles as we wanted to make sure they are very specific (as discussed in Chapter 7).

To create the archival voids profile for the remaining three archives we used longitudinal access logs data of our MemGator server and selected URIs for each archive that have always returned a "`404 Not Found`" response. These archival voids profiles are smaller in size than the one for Arquivo.pt because our MemGator server receives a relatively smaller amount of traffic than the Arquivo.pt server. When creating these archival voids profiles we took a conservative approach to select only high-frequency (> 100) URI-Rs and prevented them from getting rolled up in smaller URI keys to make sure they do not produce any additional false negatives.

Usage-based profiles generally boost accuracy significantly, but are likely to produce an increasing number of false negatives over time. This behavior was observed by Klein et al. [162]. Our own data in Table 15 of Chapter 7 (on Page 129) suggests the same as it shows more than 5% increase in "`200 OK`" responses during a period of three and a half years from URI-Rs that were returning "`404 Not Found`" responses in the past. This means a stale usage-based profile (based on access logs) may produce false negatives for these. Moreover, periodic attempts to recreate archival voids profiles from the updated access logs may introduce new keys to the voids profile (for URIs that are gaining popularity, but are not present in the archive), but will not be helpful in removing existing keys as those were not requested due to their presence in the voids profile (i.e., a cyclic dependency issue). Archival voids profiles should ideally be created by the archives themselves (as discussed in Chapter 7), but if an aggregator decides to create these based on its past observations then it should use some reliable techniques to update the voids profiles frequently. Below are a couple of ways to achieve this:

- Periodically poll each archive for URIs that constituted their corresponding archival

Table 28. Archival Holdings of Primary Targets

Archive	Primary Target	Holdings %
Arquivo.pt	`*.pt`	61.42
Archive-It	`*.{edu,gov}`	12.20
UKWA	`*.uk`	98.95
Stanford	`*.stanford.edu`	0.90

voids profiles (this requires bookkeeping of complete URIs, not just the rolled up URI keys).

- Continuous updates by allowing a small percentage of requests to hit the archive, even if the lookup URIs had a matching URI key prefix in the voids profile of an archive.

We randomly sampled one million unique URIs from our longitudinal MemGator access logs to create a sample lookup dataset. We annotated these sample lookup URIs with their presence or absence in each of the four archives. We also annotated these lookup URIs with their number of occurrences in MemGator logs. These one million URIs collectively appeared more than 1.6 million times in MemGator access logs. The prevalence of these lookup URIs is below 4% in any individual web archive and as low as 0.02% in the smallest one and about 7.9% of these sample URIs are present in at least one archive. This low prevalence means the overlap of what people look for in web archives and what is being archived is small, which means profiling web archives for Memento routing is important. While MemGator's access logs are not necessarily representative of what is being accessed from each web archive individually, we chose this dataset for evaluation because Memento routing is needed for Memento aggregators, not for individual web archives. Moreover, in Chapter 6 we used four different lookup URI datasets and found that the maximum prevalence was not significantly different from this.

Using these datasets as our gold standard we evaluated our *routing* score and classifier models for Memento routing. A summary of this dataset is shown in Table 27.

9.4.2 COLLECTION DIFFUSION

Table 28 shows the primary target URI space of each archive and the percentage of holdings that belong to their primary scopes. Arquivo.pt was founded with the aim to

Table 29. Recall, Accuracy, and Request Cost of Various Baseline Routing Policies (With "R" lookup requests to a Memento aggregator of "N" web archives)

Policy	**Recall**	**Accuracy**	**Request Cost**
Random-1	0.276	0.02	R
Random-2	0.531	0.02	$2R$
Top-1	0.918	0.07	R
Top-2	0.987	0.04	$2R$
Broadcast	1.000	0.02	$N \cdot R$

preserve web content of interest to the Portuguese community [117], but we see that about 40% of their collection belongs to URIs other than "`.pt`" TLDs. Archive-It is a subscription-based web archiving service with its primary customers being educational institutions and libraries. We would assume that they would primarily focus on collecting URIs from "`.edu`" and "`.gov`" TLDs. However, only about 12% of their URIs belong to these TLDs while containing about 55% and 20% of "`.com`" and "`.org`", respectively. Among all these web archives we used for evaluation, UKWA has the tightest collection policy, but even that has over 1% of leakage of TLDs other than its primary scope (i.e., "`.uk`"). Stanford has a small collection of web pages archived by the University. We would assume that their collection will focus on their university URIs (i.e., `*.stanford.edu`). However, we found less than 1% prevalence of such URIs in their collection while 52% and 28% of "`.com`" and "`.gov`" TLDs, respectively. It is worth noting that their collection might have changed over time as the dataset we have was transferred to us when their archive was only one year old. These findings reinforce assessment that web archival collections diffuse over time and simple TLD-based Memento routing is insufficient.

9.4.3 BASELINE ROUTING

There are about 79,000 sample lookup URIs that are present in at least one archive. Table 29 illustrates recall and request cost if an aggregator decides to poll from only a few archives for each lookup request. In the case of broadcasting the recall is 100%, but it comes with a cost of "$N \cdot R$" requests (where "R" is the number of lookup requests received by a Memento aggregator and "N" is the number of upstream web archives it is aggregating from). The Accuracy in Table 29 is poor (between 2% and 7%) for all policies because in

this routing method the *routing* score does not determine how many archives will be polled for a given lookup request, or in other words, we do not attempt to identify true negatives. Since prevalence of lookup dataset in each archive is small, false positives would be high. Moreover, very few URIs are present in more than one archives so a higher value of k would increase the wastage.

If an aggregator randomly selects "k" archives (i.e., random-k) for each lookup request (to reduce the request cost to "$k \cdot R$") and each lookup URI is present in exactly one of the "N" archives, then the theoretical recall value would be "k/N". Our experimental results in Table 29 confirm this, but our recall values are slightly higher than the theoretical estimate because a small percentage of our sample lookup URIs is present in more than one archives (though these overlaps are small, as shown in Figure 7 of Chapter 1 on Page 11).

When we selected only the topmost archive based on the *routing* score for each of these lookup URIs we missed 8.23% of these, but when we selected the top-2 archives we only missed 1.31%. While these results are promising, top-k approach should not be used in isolation because it will generate many false positives for those URIs that are not present in any archive. To avoid the bulk of those false positives while still leveraging the top-k approach an application should in conjunction use a low cut-off threshold for the *routing* score. Another point to note when deciding the value of k for the top-k approach for Memento routing is whether an application is looking for some, most, or all good results as a smaller k would cover fewer good results.

9.4.4 HEURISTIC ROUTING

We started by combining *MementoMaps* (both holdings and voids) from all of the four web archives into a single inverted index. By doing so the number of records dropped from about 32 million (all *MementoMap* lines combined) to about 25 million unique URI keys (i.e., about 22% reduction). During this process we also converted *frequency* values (in this case, URI-M counts) of *MementoMaps* to corresponding *density* scores as described in Section 9.2.1.

After that we calculated the *routing* score for each of the one million sample lookup URI against each of the four web archives using the model described in Section 9.2.3 and annotated sample URI dataset with these scores. Since we already annotated each sample URI with its presence or absence information in each archive in the process of gold dataset creation, we only needed to evaluate how effective these *routing* scores are at minimizing false positives and false negatives (i.e., maximizing accuracy while keeping a reasonably

Table 30. Recall, Accuracy, and Request Saving With the Heuristic Routing Score Threshold on Holdings and Voids Profiles

		Holdings Only			With Voids	
Archive	**Threshold**	**Recall**	**Accuracy**	**Savings %**	**Accuracy**	**Savings %**
Arquivo.pt	$\geq 10^{-1}$	0.894	0.903	89.40	0.964	94.72
	$\geq 10^{-2}$	0.933	0.825	80.92	0.857	86.83
	$\geq 10^{-3}$	0.991	0.638	64.61	0.681	71.97
Archive-It	$\geq 10^{-1}$	0.881	0.929	91.33	0.948	96.11
	$\geq 10^{-2}$	0.916	0.838	84.06	0.879	90.94
	$\geq 10^{-3}$	0.972	0.680	70.17	0.727	76.42
UKWA	$\geq 10^{-1}$	0.843	0.874	88.94	0.889	93.28
	$\geq 10^{-2}$	0.882	0.793	81.00	0.812	87.19
	$\geq 10^{-3}$	0.912	0.607	63.82	0.648	71.85
Stanford	$\geq 10^{-1}$	0.809	0.802	85.38	0.819	89.14
	$\geq 10^{-2}$	0.867	0.675	70.96	0.701	77.88
	$\geq 10^{-3}$	0.898	0.586	63.22	0.603	69.63

good recall) when we choose different cut-off thresholds for Memento routing.

In Table 30 we show our routing evaluation results. In this table we have three different cut-off points (i.e., 10^{-1}, 10^{-2}, and 10^{-3}) for the *routing* score for each web archive. If a lookup URI in the gold dataset has a *routing* score above the cut-off threshold for any archive, we consider that lookup to be routed to that archive. The *Accuracy* value represents how many of the one million lookup URIs we have correctly identified to be present in or absent from the corresponding archive with the given threshold. The *Recall* value shows how many of the URIs present in an archive were routed to that archive. A *Recall* value of 0.9 would mean that we missed 10% of the URIs that were present in an archive (i.e., a 10% false negatives).

The *Savings* column represents the percentage of 1.6 million MemGator requests (that include repetitions of the one million unique lookup URIs we chose for our gold dataset) that would have been avoided from being routed. When the prevalence is small the value of request savings should be very close to the accuracy value because in that case true negatives play the most significant role in deciding what lookups should not be routed. However, if

the popularity of request URIs is not uniformly distributed in FP, FN, TP, and TN then depending on the access pattern the savings percentage can be different.

There are two sets of results in this table, one pair of *Accuracy* and *Savings* represents routing with only the archival holdings profiles and the other pair is for the case where we combined both holdings and voids profiles for each archive. The *Recall* value remains the same for both the cases as we created archival voids profile with strict conditions that do not produce any additional false negatives. With the inclusion of archival voids profile we get a more significant increase in *Savings* than in *Accuracy* because for archival voids profiling we have used URIs that are requested more frequently (i.e., once correct true negative identification affects multiple requests).

We see an inherent trade-off between *Accuracy* and *Recall*. For example, in the case of Arquivo.pt, if we allow about 11% reduction in recall we can get an accuracy of about 96% that would save over 94% of our lookup requests. However, if we increase the recall to about 99%, the accuracy goes down to about 64% with only about 60% savings in lookup routing. Archive-It had slightly better prevalence, so we would expect better routing scores for it, but Arquivo.pt had a larger archival voids profile, which has likely increased its scores. The poor recall in UKWA and Stanford archives was due to their significantly low prevalence. These results show a significant improvement over LANL's classifier-based routing that produced a recall of 0.73 with many wasted requests [162].

Moreover, we learned that an application should use different cut-off threshold for different archives based on how detailed their profiling is and how large the archive is. Alternatively, a weighted normalized relative routing score can be used to achieve similar effects.

9.4.5 MACHINE LEARNING-BASED ROUTING

In this technique of we did not use our heuristic *routing* score, *closeness*score, or any cut-off thresholds. Instead, we fed `lookup_key_length`, `uri_key_length`, `keys_distance`, and `density` metrics as returned from our inverted index lookup (as shown in Figure 79) and the ground truth from the gold dataset to our Classifier Reborn implementation and allowed it to build a statistical Bayesian classifier model. We built these models for each web archive individually as well as a combined one that leverages data from all the web archives. Finally, we performed a 10-fold cross-validation in which the gold dataset is partitioned in ten equal parts, then nine parts are used for train and one part is set aside for testing, this process is rotated for all ten possible combinations and a combined report is generated.

In Table 31 we show our routing evaluation results based on our Machine Learning model.

Table 31. Recall, Accuracy, and Request Savings in Machine Learning-Based Routing

Archive	**Recall**	**Accuracy**	**Savings %**
Arquivo.pt	0.863	0.978	96.22
Archive-It	0.844	0.941	94.92
UKWA	0.629	0.933	94.11
Stanford	0.535	0.968	95.27
Combined	0.827	0.948	95.73

The overall pattern we see in these results is that the accuracy is better than our heuristic approach, but the recall is significantly poorer. For the smallest archive with the lowest prevalence we missed over 46% of the URIs that were present in the Stanford archive while the accuracy was almost 97%. This is not surprising because there were only about 0.02% lookup URIs dataset that were present in the Stanford web archive. As a result the model was biased towards predicting almost every request to be not present in the archive while still having good a accuracy score. Apparently, we cannot learn "a cat" without learning "not a cat". It will be a good future work to see how various machine learning models perform on Memento routing using our input features when they are trained on a dataset that has a better prevalence.

Finally, we combined the input data for every lookup URI and web archive pair to train a generic model. We found that it performed well even for those archives with low prevalence, though the recall was still below 83%.

9.5 CHAPTER SUMMARY

In this chapter we addressed our third research question, "***RQ3**: How to utilize archive profiles for the routing of URI lookup requests?*" We combined what we learned in the last three chapters and put them to work. We first revisited what Memento aggregation and Memento routing mean, why they are important, and where *MementoMap* framework fits in the ecosystem.

We described our methodology in which we introduced and defined three different scores *density* (a normalized transformation of the *frequency*), *closeness* (a measure of how similar two URI keys are), and *routing* score (a means to estimate whether a lookup URI is present in an archive). Then we described inverted index with examples and illustrated how we can

use it to combine multiple *MementoMaps* for efficient binary searching in a file on disk. We described how the lookup is preformed in *MementoMaps* (or inverted index) and how the results are prepared for downstream consumption.

We described our external open-source contributions to Classifier Reborn, a machine learning module written in Ruby, and how we customized and used it for our work. Our contributions included a significant overhaul of its architecture, a brand new evaluation and validation component, and documentation.

Finally, we evaluated Memento routing using our *MementoMap* framework. We first described the gold dataset we prepared that consists of *MementoMaps* of four different web archives and one million unique URI samples from our MemGator's longitudinal access logs. We illustrated a significant diffusion of URI scope in different web archives, which concludes that a simple TLD-based routing is not adequate. We found that using the heuristic *routing* score threshold technique we can achieve over 96% accuracy if we accept about 89% recall and for a recall of 99% we managed to get about 68% accuracy, which translates to about 72% saving in wasted lookup requests. Moreover, when using top-k archives for routing and choosing only the topmost archive, we missed only about 8% of the sample URIs that are present in at least one archive, but when we selected top-2 archives, we missed less than 2% of these URIs. We also evaluated a machine learning-based routing approach, which resulted in an overall better accuracy, but poorer recall due to low prevalence of the sample lookup URI dataset in different web archives.

CHAPTER 10

CONTRIBUTIONS, FUTURE WORK, AND CONCLUSIONS

Due to the inherent nature of the web and web crawlers, as web archives grow larger over time, their collections get dispersed, despite tightly controlled collection policies (cf. Figure 8 on Page 12). As a result, it becomes difficult to describe their holdings in the URI space, even if at first glance it would seem trivial for national archives to correspond to their respective TLDs (e.g., `.pt` for Arquivo.pt and `.uk` for UK National Library). In this work we established the *MementoMap* framework to tackle this problem space.

In this chapter we summarize major contributions of our archive profiling work and related publications in the form of algorithms, software, tools, datasets, and specification drafts. Furthermore, we discuss potential future research and development based on the foundational framework laid out by this work. Finally, we conclude this work by summarizing all the previous chapters and stating take away findings.

10.1 CONTRIBUTIONS

Our contributions include many components that were essential for this work as well as some by-products that were created to help our research process. Many of our contributions have the potential to be leveraged in the future while some of our tools have already been utilized by others.

10.1.1 ALGORITHMS

We introduced the Random Searcher Model (RSM) to discover samples of holdings of web archives [30]. This algorithm utilizes fulltext search interface of web archives (if available) to query the archive with a set of keywords and collects resulting URIs that are present in the archive. Our system works with or without a static set of search keywords as a representative set of search keywords might not be available for certain languages and disciplines that the archival holdings belong to. We have described RSM in detail in Chapter 6 under Section 6.4.1. Our algorithm yielded a routing accuracy of 0.8 with a recall of 0.9 after discovering only 10% of the archival holdings.

To generate *MementoMaps* from a large list of URIs we introduced a space and time efficient algorithm [32]. Our algorithm takes a stream of sorted list of URIs (in SURT form)

and yields a *MementoMap* based on the configuration options in a single pass. The algorithm requires a constant (and small) amount of memory to operate, irrespective of the size of the input. The same algorithm can be used to compact down an existing more detailed *MementoMap* into a smaller and less detailed one by adjusting configuration parameters. Moreover, the algorithm can also be used to merge multiple *MementoMaps* into one that were either created in parallel or updated incrementally. We have described the algorithm in detail in Chapter 8. Our algorithm produced *MementoMaps* that have an accuracy of >0.6 with a relative cost <1.5% while producing zero false negatives (i.e., a recall of 1).

10.1.2 TERMINOLOGY AND METRICS

During our research we needed many terms to describe things and many metrics for evaluations. We reused existing terms and metrics when we found something suitable, but described and defined a few new ones:

- **URI Key** – We extended SURT with the wildcard support to describe subtrees of the URI space in the form of URI prefixes.
- **Archival Holdings** – A measure to describe holdings of an archive.
- **Archival Voids** – A measure to describe what an archive is missing.
- **Relative Cost** – The ratio of the number of URI keys used to describe summarized holdings of an archive over the total number of unique URI-Rs in the archive.
- **Frequency Score** – A means to represent the number of URI-Ms and/or URI-Rs under a URI key.
- **Density Score** – A normalized score derived from the frequency score to describe the archiving activity under a URI key.
- **Closeness Score** – A normalized score to describe how similar or different two URI keys are.
- **Routing Score** – A normalized score to represent how likely it is that an archive has a URI.

10.1.3 SOFTWARE/TOOLS

During our research on this archive profiling work we developed many tools to aid our research process and/or as part of our deliverables. We discussed these in Chapter 5 and outlined their functional role in the web archiving ecosystem. Below is a list of various significant tools and software that we released publicly. However, this is not a comprehensive list of all the software/tools/scripts we have created during this work.

- **InterPlanetary Wayback (IPWB)** – A novel archival replay system based on IPFS [152, 20, 23, 22, 19].
- **Reconstructive** – A client-side URL rerouting and archival banner injection Service-Worker script [21, 24, 17].
- **MemGator** – A portable Memento aggregator CLI and server [27, 10].
- **Random Searcher** – A script to sample holding of web archives via fulltext search (an implementation of our RSM algorithm) [30, 6].
- **AccessLog Parser** – A web server access log parser with added features for web archives [5].
- **Archive Profiler** – A set of script to generate URI keys, profile web archives, and analyze their holdings [6].
- **MementoMap** – A tool and module to generate, manage, and utilize *MementoMaps* (archive profiles) in the UKVS serialization format (an implementation of our single-pass *MementoMap* generation algorithm) [32, 9].
- **Classifier Reborn** – A Ruby implementation of Naive Bayes classifier (it is not our original software, but we contributed extensively to its development and overhaul) [81, 7].

10.1.4 DATASETS

To evaluate our work we collected different types of datasets from various sources as outlines below:

- WARC and/or CDX files from a few different web archives

- Access logs from a few web archives and Memento aggregators
- Longitudinal access logs from our own MemGator service with response behavior of each upstream archive that we aggregate results from
- Historical list of URIs extracted from DMOZ and Reddit
- List of words from different sources
- List of public domain suffixes

We wrote many scripts to process these datasets to clean/fix malformed records, filter unwanted entries, and create derivative datasets. Some data sources (e.g., Arquivo.pt, Archive-It, Stanford Web Archive, and the UK Web Archive) granted us with exclusive access to their datasets. We plan to make some of our original and/or derived datasets available in the future as we get permission from the upstream providers to share them publicly.

10.1.5 SPECIFICATIONS

Our work yielded a file format specification draft called Unified Key Value Store (UKVS) as described in Chapter 8. It is the file format we use to serialize *MementoMaps* for dissemination. This format is an evolution of a prior file format specification of ours called CDXJ, which was fusion of CDX and JSON formats with added support for metadata. It is an extensible and flexible and friendly file format to work with traditional Unix command line text processing tools.

To make IPWB (our archival replay system) truly decentralized, we created an early draft of a history-aware name resolution system for IPFS called IPNS Blockchain. We proposed two potential approaches to add history component to IPNS (the naming system form IPFS). One of our approaches requires a Blockchain to function while the other approach works with centralized immutable database that is publicly audited.

10.2 FUTURE WORK

The concept of *URI Key* we introduced in the form of a SURT prefix, both based on the static profiling policies we introduced in Chapter 6 and dynamic roll up with the wildcard we introduced in Chapter 8, can be used in many places where a set of URIs needs to be

summarized structurally. Another potential use case of this concept is in creating a URI diversity measure for a web collection [196, 13].

The concept of *Archival Voids* we introduced in Chapter 7 can be further investigated as a crawl quality measure. For example, if a crawler job is initiated with a set of seed URIs and is configured to collect resources within certain scopes (e.g., all the URIs under `.gov` and `.mil` TLDs) then it is desired to know how well the configured scopes were crawled and how many resources were missed.

The *UKVS* file format we introduced in Chapter 8 has many potential applications in the web archiving ecosystem and beyond where a lookup key is used to retrieve associated values. It would be worth investigating the impact of using a unified solution for *MementoMap*, archival indexing, annotations, access control, archival fixity, etc. Furthermore, independent parser libraries need to be created for the file format in various programming languages to make it easier to work with this file format.

In this work we have laid out the foundation for multi-faceted archive profiles that contain more than one dimension (e.g., URI, language, and date range) in their lookup keys. However, we have only evaluated our work for routing efficiency based on URIs, which is the readily available and the most useful facet for our primary application. There is a potential research opportunity to explore profiling based on different permutations of these facets to identify sources of additional routing accuracy gains and application where these facet permutation can be useful.

In all of our evaluations under Chapters 6, 7, and 8 we took the conservative approach to not allow any false negatives (i.e., we maintain a 100% recall) to establish the baseline. We believe that the accuracy can be increased by allowing a small amount of false negatives as it would reduce the more prevalent false positives. However, this hypothesis needs to be evaluated in a future work. It is worth noting that the cost of false positives affects the infrastructure while the cost of false negatives affects users, and for this reason we chose not to allow any false negatives in our baseline evaluations.

In our evaluations we reported relationship between *Accuracy* and *Recall* value pairs. We noted the inherent tradeoff between the two, as an increase in one value cause decrease in the other. This allows the consumer of *MementoMaps* to choose between optimizing their system for either false negatives or false positives to justify their operational cost or application needs. However, there is a future research opportunity of cost-sensitive learning where these scores can be combined into one metric by assigning different weights to them [98]. Furthermore, cost-sensitivity can be applied at the level of individual archives to cooperate

with their differential capacities. For example, if an archive is resourceful enough to handle extra traffic, we may want to allow more false positives to minimize false negatives. On the other hand, we can assign larger weights to the *Accuracy* in the case of smaller archives to allow them to flourish without the increased burden.

In Chapter 8 we used weight-based configurations that control how to roll up a branch of a URI tree. There is a potential research opportunity to explore other policies and methodologies to determine when to roll a branch up. We believe that such exploration may yield in better routing accuracy, especially, if it is informed by the external knowledge of the distribution of URIs on the web under different domain public suffixes or TLDs.

In Chapter 9 we defined a *routing* score metric and used it with heuristically determined threshold values for the routing decision. We also used the Naive Bayes classifier to make the routing decision as an alternate approach. We found that the classifier was biased toward *Accuracy* at the cost of poor *Recall* due to low prevalence. One potential approach to mitigate this issue would be to use the Bayesian method to get a probability score and use it in place of the *routing* score to be able to adjust the balance between FPs and FNs rather than relying on the classifier to make the routing decision. Moreover, there is a scope of trying other classification techniques and deep learning systems on the data that is returned from our *MementoMap* inverted index lookup to see if the routing decision can be improved.

By virtue of the Memento API, Memento aggregators can be chained in a hierarchy. This means we can have multiple Memento aggregator instances at different locations or with different policies that aggregate a subset of all the web archives. This means any one instance would be responsible for creating and maintaining *MementoMaps* of only a small number of web archives. These aggregators can then be used independently and/or aggregated by a larger aggregator. It will be a good future work to explore the potential of such hierarchical federated Memento aggregation architecture while leveraging the *MementoMap* framework for profiling web archives at each level.

Many web archives are creating event-specific collections. However, these collections usually do not contain a standardized metadata that allows cross-archive search in collections. Our *MementoMap* framework in this work is primarily based on lookup URIs. It will be useful to explore options to enable cross-archive collection discovery and browsing. We think *MementoMap* has the potential to allow such extensions in the future as it does not restrict the key fields to be URIs.

We would like to work towards the adoption of the *MementoMap* framework by public web archives. This would require development of more libraries and integration with many

existing archiving systems. Once the adoption reaches a critical mass, we can work towards finalizing the standardization process and register the `mementomap` link relation and well-known URI in the IANA registry.

10.3 CONCLUSIONS

In Chapter 1 we briefly introduced the web archiving discipline and related terminology (e.g., Memento) that were necessary to understand the problem space this work addresses. Then with the help of some real life examples, we highlighted the need of aggregating archives to address issues like limited coverage of the web in any single web archive, censorship of web archives, transient errors and attacks, lack of timely captures of important events, potentially unintentional exclusions, a means for the validity and fixity, and the lack of variety in mementos. Then, we demonstrated the significance of aggregating small archives by showing the little overlap among archives and some unique resources that are only captured by small focused archives. Furthermore, we described the usefulness of understanding the holdings of various web archives by creating their archive profiles (or *MementoMaps*). Using a real life example of an incident, we demonstrated how broadcasting lookup requests from Memento aggregators can be wasteful and problematic. Finally, we established our three primary research questions:

- **RQ1**: How to learn about the holdings and voids of an archive?
- **RQ2**: How to build an archive profile that will best summarize an archive's holdings/voids and allow for dissemination and exchange?
- **RQ3**: How to utilize archive profiles for the routing of URI lookup requests?

In Chapter 2 we provided necessary background information about various related terminologies to help readers understand the problems this work is addressing as well as their solutions.

We covered topics including HTTP and its components, access logs, web archiving, web archives, collection policies, Memento and related terminologies such as TimeGate and TimeMap, HTTP status codes in the context of web archives, Memento Aggregators, URIs and their transformations such as normalization/canonicalization and SURT, file formats like WARC, WAT, WANE, WET, CDX, and, CDXJ, syndication protocols like RSS/Atom feed, Sitemaps, robots.txt, and well-known URIs, and Inverted Indexes. We included necessary

illustrations for each terminology and established the purpose, relevance, and usefulness of each in the context of this work.

In Chapter 3 we reviewed published scholarly literature in various related areas. We covered many topics including surface web, deep/hidden web, dark web, focused crawling, textual database summarization, on-premise indexing, search form detection, query routing, Bloom filters, size of search engine indexes, size of the indexable web, archival coverage of the web, the web that cannot be archived by public web archives, public and private web archive integration, large-scale web archive indexing, and initial efforts on Memento routing via both content-based and usage-based profiles. In each section of this chapter we established the relevance of prior work with our work and any shortcomings that we need to address.

In Chapter 4 we outlined our *MementoMap* framework, described our three primary research questions in detail, and established the evaluation plan. We described three major components of the framework parallel to corresponding research questions. We briefly introduced various statistical measures for evaluation such as cost, accuracy, freshness, and efficiency.

In Chapter 5 we described various open-source tools, libraries, and scripts we built during our research as part of our deliverables and to help our research process. Some of our open-source tools have been utilized in other research and development works by many other people beyond the scope of archive profiling.

In Chapter 6 we addressed the first part of the **RQ1** to explore ways to describe archival holdings. We outlined four approaches of learning an archive's holdings:

1. CDX Profiling
2. Fulltext Search Profiling
3. Sample URI Profiling
4. Response Cache Profiling

We described the data structure of archive profiles, defined the term *URI-Key*, and outlined numerous profiling policies with varying levels of details and generation methods. Moreover, we described relevant statistical measures and summarized various datasets that we collected for evaluation. We evaluated the overlap among archives, growth of profiles with different policies, and their cost vs. routing efficiency. We found that the growth of the profile with respect to the growth of the archive follows Heaps' law, but the values of free parameters are archive-dependent. With accuracy defined as correctly predicting that the

requested URI-R is present or not present in the archive, we gained about 78% routing accuracy with less than 1% relative cost and 94% routing accuracy with less than 10% relative cost without any false negatives. The registered domain profile doubles the routing precision with respect to the *TLD-only* profile, while a profile with complete hostname and one path segment gives ten fold routing precision. For fulltext search profiling we then introduced our contribution called RSM. We described its data structure, policies, operation modes, procedures, and reference implementation. Finally, we evaluated the routing efficiency of profiles created using RSM and concluded that the DDom profile (that has less than 1% cost as compared to the URIR profile) can have about 0.9 Recall while correctly routing (or not routing) more than 80% of the URIs by only knowing 10% of the archive.

In Chapter 7 we addressed the second part of the **RQ1** to explore ways to describe archival voids. We defined and discussed *Archival Voids* and established a means to represent portions of URI spaces that are not present in web archives. We illustrated how archival voids and archival holdings interact with each other in a hierarchical manner to describe holdings and voids in more specific portions of the URI spaces. We discussed various sources of truth that can be used to create archival voids profiles. For evaluation we used access logs from Arquivo.pt for which we first described access patterns and surfaced various corner cases that were present in it. We discussed prevalent Soft-404 TimeMaps in the access logs for many years, and techniques we used to remedy that in order to make a more meaningful analysis of the dataset. We discussed the distribution of HTTP response status codes in the access logs and reported how these status codes changed over time for various URIs. We evaluated the routing accuracy against various archival voids profiles created from these access logs and found that we could have avoided more than 8% of the false positives (on top of the accuracy we got from archival holdings profile as discussed in Chapter 6) if Arquivo.pt were to provide an archival voids profile based on URIs that were requested hundreds of times and never returned a success response. Finally, we discussed who should create archival voids profile and provided some guidelines based on our understanding.

In Chapter 8 we addressed **RQ2** to explored serialization and dissemination methods of archive profiles. We first described UKVS, the file format we use for serialization of *MementoMaps*. We illustrated a list of a sorted SURTs with wildcards and organized them in a tree to elaborate on the structure of *URI-Keys*. We then described various structures and attributes of *MementoMap* files suitable for different application needs. We described our single-pass efficient algorithm for generation and compaction of *MementoMaps* and a reference implementation of it along with binary search in file on disk. Then we described a

few standard means of disseminating generated *MementoMaps* by archives or third parties. To evaluate the serialization format we used and its effectiveness, we measured overlap of archived resources in Arquivo.pt against accessed resources by MemGator. We analyzed holdings of Arquivo.pt in detail and the shape of their URI tree. Finally, we compared the cost and routing accuracy of these *MementoMaps* and found that a *MementoMap* of less than 1.5% Relative Cost can correctly identify the presence or absence of 60% of the lookup URIs in the corresponding archive without any false negatives.

In Chapter 9 we addressed **RQ3** in which we combined profiles of various web archives and routed Memento lookup queries to suitable archives. We described our methodology in which we introduced and defined three different scores: *density*, *closeness*, and *routing*. Then we described inverted index and how we used it to combine multiple *MementoMaps* for efficient binary searching and reported results. We described our significant external open-source contributions to Classifier Reborn. We illustrated a significant diffusion of URI scope in different web archives, which concludes that a simple TLD-based routing is not adequate.

Finally, we evaluated our Memento routing models and found that using the *routing* score threshold technique we can achieve over 96% accuracy if we accept about 89% recall and for a recall of 99% we managed to get about 68% accuracy, which translates to about 72% saving in wasted lookup requests. Moreover, when using top-k archives for routing and choosing only the topmost archive, we missed only about 8% of the sample URIs that are present in at least one archive, but when we selected top-2 archives, we missed less than 2% of these URIs. We also evaluated a machine learning-based routing, which resulted in an overall better accuracy, but poorer recall due to low prevalence of the sample lookup URI dataset in different web archives.

To summarize, we began our work by identifying serious issues related to Memento aggregation and Memento routing, a niche yet important field, that was not explored well. We performed thorough research by quantifying the scale of the problems, assessing their impacts, identifying potential approaches to solve them, and evaluating those approaches. We introduced a novel framework called *MementoMap* as a deliverable of our research, which allows a flexible and scalable means to summarize and express holdings and voids of web archives. The primarily purpose of the framework is to make Memento routing more efficient, but it can be useful in many other applications. We reported promising results as part of our evaluations, and we hope that these results will encourage practitioners in the web archiving community to adopt the *MementoMap* framework to allow new players to

flourish while maximizing the yield for the end users. We hope that numerous algorithms, terminologies, metrics, specification, datasets, and software tools we produced as part of our work will inform future research works in related fields and be useful in many services and applications.

REFERENCES

[1] Eugene Agichtein and Luis Gravano. 2003. Querying Text Databases for Efficient Information Extraction. In *Proceedings of the 19th International Conference on Data Engineering* (ICDE '03), 113–124. DOI: 10.1109/ICDE.2003.1260786.

[2] Eugene Agichtein, Panagiotis Ipeirotis, and Luis Gravano. 2003. Modeling Query-Based Access to Text Databases. In *International Workshop on Web and Databases* (WebDB '03), 87–92.

[3] Scott Ainsworth, Ahmed Alsum, Hany SalahEldeen, Michele C. Weigle, and Michael L. Nelson. 2011. How Much of the Web Is Archived? In *Proceedings of the 11th ACM/IEEE Joint International Conference on Digital Libraries* (JCDL '11), 133–136. DOI: 10.1145/1998076.1998100.

[4] Reem Al-Masri and James Cain. 2017. In Jordan, the 'Invisible Hand' Blocks Internet Archive. https://www.7iber.com/technology/the-invisible-hand-blocks-internet-archive/. (2017).

[5] Sawood Alam. 2019. AccessLog Parser and CLI. https://github.com/oduwsdl/accesslog-parser. (2019).

[6] Sawood Alam. 2014. Archive Profiler: Scripts to Generate Profiles of Various Web Archives. https://github.com/oduwsdl/archive_profiler. (2014).

[7] Sawood Alam. 2017. Classifier Validation. https://jekyll.github.io/classifier-reborn/validation. (2017).

[8] Sawood Alam. 2016. Memento: Help Us Route URI Lookups to the Right Archives. https://netpreserveblog.wordpress.com/2016/01/08/memento-help-us-route-uri-lookups-to-the-right-archives/. (2016).

[9] Sawood Alam. 2019. MementoMap: A Tool to Summarize Web Archive Holdings. https://github.com/oduwsdl/MementoMap. (2019).

[10] Sawood Alam. 2015. MemGator: A Memento Aggregator CLI and Server in Go. https://github.com/oduwsdl/MemGator. (2015).

[11] Sawood Alam. 2019. Portuguese Archive MIME Types Count. https://gist.github.com/ibnesayeed/bb167fe19c5719d87c1c1f665001d44b. (2019).

[12] Sawood Alam. 2019. Portuguese Archive Status Codes Count. https://gist.github.com/ibnesayeed/7307f0bf1783357db99f8b2357249dd0. (2019).

[13] Sawood Alam. 2020. Seed Analyzer. https://github.com/oduwsdl/seed-analyzer. (2020).

[14] Sawood Alam. 2019. Unified Key Value Store (UKVS). https://github.com/oduwsdl/ORS/blob/master/ukvs.md. (2019).

[15] Sawood Alam. 2018. Web ARChive (WARC) File Format. https://www.slideshare.net/ibnesayeed/web-archive-warc-file-format. (2018).

[16] Sawood Alam. 2020. Web Archives Size. https://github.com/ibnesayeed/web-archives-size. (2020).

[17] Sawood Alam and John A. Berlin. 2017. Reconstructive: A ServiceWorker for Client-Side Reconstruction of Composite Mementos. https://oduwsdl.github.io/Reconstructive/. (2017).

[18] Sawood Alam, Charles L. Cartledge, and Michael L. Nelson. 2014. Support for Various HTTP Methods on the Web. Technical report arXiv:1405.2330.

[19] Sawood Alam and Mat Kelly. 2016. InterPlanetary Wayback: Peer-to-Peer Permanence of Web Archives. https://github.com/oduwsdl/ipwb. (2016).

[20] Sawood Alam, Mat Kelly, and Michael L. Nelson. 2016. InterPlanetary Wayback: The Permanent Web Archive. In *Proceedings of the 16th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '16), 273–274. DOI: 10.1145/2910896.2925467.

[21] Sawood Alam, Mat Kelly, Michele C. Weigle, and Michael L. Nelson. 2017. Client-Side Reconstruction of Composite Mementos Using ServiceWorker. In *Proceedings of the 17th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '17), 237–240. DOI: 10.1109/JCDL.2017.7991579.

[22] Sawood Alam, Mat Kelly, Michele C. Weigle, and Michael L. Nelson. 2018. InterPlanetary Wayback: A Distributed and Persistent Archival Replay System Using IPFS. DWeb Summit '18. (2018).

[23] Sawood Alam, Mat Kelly, Michele C. Weigle, and Michael L. Nelson. 2018. InterPlanetary Wayback: The Next Step Towards Decentralized Web Archiving. IPFS Lab Day '18. (2018).

[24] Sawood Alam, Mat Kelly, Michele C. Weigle, and Michael L. Nelson. 2018. Unobtrusive and Extensible Archival Replay Banners Using Custom Elements. In *Proceedings of the 18th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '18), 319–320. DOI: 10.1145/3197026.3203881.

[25] Sawood Alam, Ilya Kreymer, and Michael L. Nelson. 2015. Object Resource Stream (ORS) and CDX-JSON (CDXJ) Draft. https://github.com/oduwsdl/ORS. (2015).

[26] Sawood Alam and Michael L. Nelson. 2019. CS 431/531 Web Server Design: A Course to Develop a Standard Compliant HTTP Web Server. https://cs531-f19.github.io/. (2019).

[27] Sawood Alam and Michael L. Nelson. 2016. MemGator - A Portable Concurrent Memento Aggregator: Cross-Platform CLI and Server Binaries in Go. In *Proceedings of the 16th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '16), 243–244. DOI: 10.1145/2910896.2925452.

[28] Sawood Alam, Michael L. Nelson, Herbert Van de Sompel, Lyudmila L. Balakireva, Harihar Shankar, and David S. H. Rosenthal. 2015. Web Archive Profiling Through CDX Summarization. In *Proceedings of the 19th International Conference on Theory and Practice of Digital Libraries* (TPDL '15), 3–14. DOI: 10.1007/978-3-319-24592-8_1.

[29] Sawood Alam, Michael L. Nelson, Herbert Van de Sompel, Lyudmila L. Balakireva, Harihar Shankar, and David S. H. Rosenthal. 2016. Web Archive Profiling Through CDX Summarization. *International Journal on Digital Libraries*, 17, 3, 223–238. DOI: 10.1007/s00799-016-0184-4.

[30] Sawood Alam, Michael L. Nelson, Herbert Van de Sompel, and David S. H. Rosenthal. 2016. Web Archive Profiling Through Fulltext Search. In *Proceedings of the 20th International Conference on Theory and Practice of Digital Libraries* (TPDL '16), 121–132. DOI: 10.1007/978-3-319-43997-6_10.

[31] Sawood Alam, Michele C. Weigle, Michael L. Nelson, Martin Klein, and Herbert Van de Sompel. 2019. Supporting Web Archiving via Web Packaging. Technical report arXiv:1906.07104. https://arxiv.org/abs/1906.07104.

[32] Sawood Alam, Michele C. Weigle, Michael L. Nelson, Fernando Melo, Daniel Bicho, and Daniel Gomes. 2019. MementoMap Framework for Flexible and Adaptive Web

Archive Profiling. In *Proceedings of the 19th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '19), 172–181. DOI: 10.1109/JCDL.2019.00033.

[33] Rosa Alarcon, Erik Wilde, and Jesus Bellido. 2011. Hypermedia-driven RESTful Service Composition. *Service-Oriented Computing*, 111–120.

[34] Abdulrahman Alarifi, Mansour Alghamdi, Mohammad Zarour, Batoul Aloqail, Heelah Alraqibah, Kholood Alsadhan, and Lamia Alkwai. 2012. Estimating the Size of Arabic Indexed Web Content. *Scientific Research and Essays*, 7, 28, 2472–2483. DOI: 10.5897/SRE11.1708.

[35] Lulwah M. Alkwai, Michael L. Nelson, and Michele C. Weigle. 2017. Comparing the Archival Rate of Arabic, English, Danish, and Korean Language Web Pages. *ACM Transactions on Information Systems*, 36, 1, 1:1–1:34. DOI: 10.1145/3041656.

[36] Lulwah M. Alkwai, Michael L. Nelson, and Michele C. Weigle. 2015. How Well Are Arabic Websites Archived? In *Proceedings of the 15th ACM/IEEE-CE Joint Conference on Digital Libraries* (JCDL '15), 223–232. DOI: 10.1145/2756406.2756912.

[37] Yasmin AlNoamany, Ahmed AlSum, Michele C. Weigle, and Michael L. Nelson. 2014. Who and What Links to the Internet Archive. *International Journal on Digital Libraries*, 14, 3-4, 101–115. DOI: 10.1007/s00799-014-0111-5.

[38] Ahmed AlSum. 2011. Memento Server. https://code.google.com/archive/p/memento-server/. (2011).

[39] Ahmed AlSum, Michele C. Weigle, Michael L. Nelson, and Herbert Van de Sompel. 2013. Profiling Web Archive Coverage for Top-Level Domain and Content Language. In *Proceedings of the 17th International Conference on Theory and Practice of Digital Libraries* (TPDL '13), 60–71. DOI: 10.1007/978-3-642-40501-3_7.

[40] Ahmed AlSum, Michele C. Weigle, Michael L. Nelson, and Herbert Van de Sompel. 2014. Profiling Web Archive Coverage for Top-Level Domain and Content Language. *International Journal on Digital Libraries*, 14, 3-4, 149–166. DOI: 10.1007/s00799-014-0118-y.

[41] Apache HTTP Server. 2013. Common Log Format and Combined Log Format. https://httpd.apache.org/docs/trunk/logs.html. (2013).

[42] Mohamed Aturban. 2020. *A Framework for Verifying the Fixity of Archived Web Resources*. PhD thesis. Old Dominion University. DOI: 10.25777/pc8d-y213.

[43] Mohamed Aturban. 2019. Where Did the Archive Go? Part 1: Library and Archives Canada. https://ws-dl.blogspot.com/2019/08/2019-08-30-where-did-archive-go-part1.html. (2019).

[44] Mohamed Aturban. 2019. Where Did the Archive Go? Part 2: National Library of Ireland. https://ws-dl.blogspot.com/2019/09/2019-09-10-where-did-archive-go-part-2.html. (2019).

[45] Mohamed Aturban. 2019. Where Did the Archive Go? Part 3: Public Record Office of Northern Ireland. https://ws-dl.blogspot.com/2019/09/2019-09-25-where-did-archive-go-part-3.html. (2019).

[46] Mohamed Aturban. 2019. Where Did the Archive Go? Part 4: WebCite. https://ws-dl.blogspot.com/2019/10/2019-10-21-where-did-archive-go-part-4.html. (2019).

[47] Mohamed Aturban, Sawood Alam, Michael L. Nelson, and Michele C. Weigle. 2019. Archive Assisted Archival Fixity Verification Framework. In *Proceedings of the 19th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '19), 162–171. DOI: 10.1109/JCDL.2019.00032.

[48] Paul Baclace. 2020. Making A Production Classifier Ensemble. https://towardsdatascience.com/making-a-production-classifier-ensemble-2d87fbf0f486. (2020).

[49] Jefferson Bailey. 2016. Preserving U.S. Government Websites and Data as the Obama Term Ends. http://blog.archive.org/2016/12/15/preserving-u-s-government-websites-and-data-as-the-obama-term-ends/. (2016).

[50] Jefferson Bailey, Vinay Goel, and Mark Sullivan. 2016. WANE Overview and Technical Details. https://webarchive.jira.com/wiki/spaces/ARS/pages/88309872/WANE+Overview+and+Technical+Details. (2016).

[51] Jefferson Bailey, Abigail Grotke, Kristine Hanna, Cathy Hartman, Edward McCain, Christie Moffatt, and Nicholas Taylor. 2014. Web Archiving in the United States: A 2013 Survey - An NDSA Report. https://blogs.loc.gov/thesignal/2014/10/results-from-the-2013-ndsa-u-s-web-archiving-survey/. (2014).

[52] Jefferson Bailey, Abigail Grotke, Edward McCain, Christie Moffatt, and Nicholas Taylor. 2017. Web Archiving in the United States: A 2016 Survey - An NDSA Report. http://ndsa.org/documents/WebArchivingintheUnitedStates_A2016Survey.pdf. (2017).

[53] Steve Bailey and Dave Thompson. 2006. UKWAC: Building the UK's First Public Web Archive. *D-Lib Magazine*, 12, 1. DOI: 10.1045/january2006-thompson.

[54] Ziv Bar-Yossef, Idit Keidar, and Uri Schonfeld. 2009. Do Not Crawl in the DUST: Different URLs With Similar Text. *ACM Transactions on the Web (TWEB)*, 3, 1, 3:1–3:31. DOI: 10.1145/1462148.1462151.

[55] Luciano Barbosa and Juliana Freire. 2007. Combining Classifiers to Identify Online Databases. In *Proceedings of the 16th International Conference on World Wide Web* (WWW '07), 431–440. DOI: 10.1145/1242572.1242631.

[56] Waldo Bastian, Ryan Lortie, and Lennart Poettering. 2010. XDG Base Directory Specification. https://specifications.freedesktop.org/basedir-spec/basedir-spec-latest.html. (2010).

[57] Sylvain Belanger. 2020. Documenting COVID-19 and the Great Confinement in Canada. https://netpreserveblog.wordpress.com/2020/07/15/documenting-covid-19-and-the-great-confinement-in-canada/. (2020).

[58] Mike Belshe, Roberto Peon, and Martin Thomson. 2010. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540. Internet Engineering Task Force, (2010).

[59] Juan Benet. 2014. IPFS - Content Addressed, Version, P2P File System. Technical report arXiv:1407.3561.

[60] Donna Bergmark, Carl Lagoze, and Alex Sbityakov. 2002. Focused Crawls, Tunneling, and Digital Libraries. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries* (ECDL '02). Volume 2458, 91–106. DOI: 10.1007/3-540-45747-X_7.

[61] John A. Berlin, Mat Kelly, Michael L. Nelson, and Michele C. Weigle. 2017. WAIL: Collection-Based Personal Web Archiving. In *Proceedings of the IEEE/ACM Joint Conference on Digital Libraries (JCDL)*, 340–341. DOI: 10.1109/JCDL.2017.7991619.

[62] Tim Berners-Lee, Roy T. Fielding, and Larry Masinter. 2005. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986. Internet Engineering Task Force, (2005).

[63] Peter Beverloo, Martin Thomson, Michaël van Ouwerkerk, Bryan Sullivan, and Eduardo Fullea. 2020. Push API. https://w3c.github.io/push-api/. (2020).

[64] Krishna Bharat and Andrei Z. Broder. 1998. A Technique for Measuring the Relative Size and Overlap of Public Web Search Engines. *Computer Networks*, 30, 1-7, 379–388. DOI: 10.1016/S0169-7552(98)00127-5.

[65] Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13, 7, 422–426.

[66] Nicolas Bornand, Lyudmila Balakireva, and Herbert Van de Sompel. 2016. Routing Memento Requests Using Binary Classifiers. In *Proceedings of the 16th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '16), 63–72. DOI: 10.1145/2910896.2910899.

[67] Andrei Z. Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet L. Wiener. 2000. Graph Structure in the Web. *Computer Networks*, 33, 1-6, 309–320. DOI: 10.1016/S1389-1286(00)00083-9.

[68] Andrei Z. Broder and Michael Mitzenmacher. 2003. Survey: Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1, 4, 485–509. DOI: 10.1080/15427951.2004.10129096.

[69] Justin F. Brunelle, Mat Kelly, Michele C. Weigle, and Michael L. Nelson. 2016. The Impact of JavaScript on Archivability. *International Journal on Digital Libraries*, 17, 2, 95–117. DOI: 10.1007/s00799-015-0140-8.

[70] Justin F. Brunelle, Michele C. Weigle, and Michael L. Nelson. 2017. Archival Crawlers and JavaScript: Discover More Stuff but Crawl More Slowly. In *Proceedings of the 17th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '17), 1–10. DOI: 10.1109/JCDL.2017.7991554.

[71] Mike Burner and Brewster Kahle. 1996. Arc File Format. https://archive.org/web/researcher/ArcFileFormat.php. (1996).

[72] Chris Butler. 2018. Addressing Recent Claims of 'Manipulated' Blog Posts in the Wayback Machine. http://blog.archive.org/2018/04/24/addressing-recent-claims-of-manipulated-blog-posts-in-the-wayback-machine/. (2018).

[73] Chris Butler. 2017. Comcast's Blocking and Un-Blocking of Archive.org – What We Know So Far. https://blog.archive.org/2017/06/05/comcasts-blocking-and-un-blocking-of-archive-org-what-we-know-so-far/. (2017).

[74] Chris Butler. 2017. Statement and Questions Regarding an Indian Court's Order to Block archive.org. https://blog.archive.org/2017/08/09/statement-and-questions-regarding-an-indian-courts-order-to-block-archive-org/. (2017).

[75] Chris Butler. 2017. Who Blocked the Archive in Jordan? https://blog.archive.org/2017/04/11/who-blocked-the-archive-in-jordan/. (2017).

[76] Helena Byrne. 2020. Reviewing Football History Through the UK Web Archive. *Soccer & Society*, 21, 4, 461–474. DOI: 10.1080/14660970.2020.1751474.

[77] Jamie Callan and Margaret Connell. 2001. Query-Based Sampling of Text Databases. *ACM Transactions on Information Systems (TOIS)*, 19, 2, 97–130. DOI: 10.1145/382979.383040.

[78] Jamie Callan, Margaret Connell, and Aiqun Du. 1999. Automatic Discovery of Language Models for Text Databases. In *Proceedings of the 1999 ACM SIGMOD International Conference on Management of Data* (SIGMOD '99), 479–490. DOI: 10.1145/304182.304224.

[79] Green Cardamom. 2016. WaybackMedic Bot. https://github.com/greencardamom/WaybackMedic. (2016).

[80] Josiah L. Carlson. 2013. *Redis in Action*. Manning Publications Co. ISBN: 978-1-61729-085-5.

[81] Lucas Carlson, David Fayram, Cameron McBride, Ivan Acosta-Rubio, Parker Moore, Chase Gilliam, and Sawood Alam. 2014. Classifier Reborn. https://github.com/jekyll/classifier-reborn. (2014).

[82] Todd Carpenter, Michael L. Nelson, Bernhard Haslhofer, Shlomo Sanders, Richard Jones, Robert Sanderson, Martin Klein, Herbert Van de Sompel, Graham Klyne, Paul Walk, Carl Lagoze, Simeon Warner, Stuart Lewis, Zhiwu Xie, Peter Murray, and Jeff Young. 2017. ResourceSync Framework Specification (ANSI/NISO Z39.99-2017). http://www.openarchives.org/rs/1.1/resourcesync. (2017).

[83] Catalin Cimpanu. 2018. Mozilla to Remove Support for Built-In Feed Reader From Firefox. https://www.bleepingcomputer.com/news/software/mozilla-to-remove-support-for-built-in-feed-reader-from-firefox/. (2018).

[84] Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. 2009. Power-Law Distributions in Empirical Data. *SIAM Review*, 51, 4, 661–703. DOI: 10.1137/070710111.

[85] Cleymour. 2015. WARC Revision 1.1 (Augmentation): Specification of the WAT Format. https://github.com/iipc/warc-specifications/issues/16. (2015).

[86] John Collomosse, Tu Bui, Alan Brown, John Sheridan, Alexander L. Green, Mark Bell, Jamie Fawcett, Jez Higgins, and Olivier Thereaux. 2018. ARCHANGEL: Trusted Archives of Digital Public Documents. In *Proceedings of the ACM Symposium on Document Engineering 2018, DocEng 2018*, 31:1–31:4. DOI: 10.1145/3209280.3229120.

[87] Kate Conger. 2016. Backing Up The History of the Internet in Canada to Save It From Trump. https://techcrunch.com/2016/12/08/backing-up-the-history-of-the-internet-in-canada-to-save-it-from-trump/. (2016).

[88] Jared Cope, Nick Craswell, and David Hawking. 2003. Automated Discovery of Search Interfaces on the Web. In *Proceedings of the 14th Australasian Database Conference - Volume 17* (ADC '03), 181–189.

[89] Miguel Costa, Daniel Gomes, Francisco M. Couto, and Mário J. Silva. 2013. A Survey of Web Archive Search Architectures. In *Proceedings of the Temporal Web Analytics Workshop* (TempWeb '13), 1045–1050. DOI: 10.1145/2487788.2488116.

[90] Douglas Crockford. 2006. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627. Internet Engineering Task Force, (2006).

[91] Shuaixiang Dai, Qian Diao, and Changle Zhou. 2005. Performance Comparison of Language Models for Information Retrieval. In *Artificial Intelligence Applications and Innovations*, 721–730.

[92] Maurice de Kunder. 2018. WorldWideWebSize.com – The Size of the World Wide Web (The Internet). http://worldwidewebsize.com/. (2018).

[93] Peter Deutsch. 1996. GZIP File Format Specification Version 4.3. RFC 1952. Internet Engineering Task Force, (1996).

[94] Adrian Dobra and Stephen E. Fienberg. 2004. How Large Is the World Wide Web? In *Web Dynamics - Adapting to Change in Content, Size, Topology and Use*, 23–44.

[95] Kim Dulin and Adam Ziegler. 2017. Scaling Up Perma.cc: Ensuring the Integrity of the Digital Scholarly Record. *D-Lib Magazine*, 23, 5/6. DOI: 10.1045/may2017-dulin.

[96] Caleb Ecarma. 2018. EXCLUSIVE: Joy Reid Claims Newly Discovered Homophobic Posts From Her Blog Were 'Fabricated'. https://www.mediaite.com/online/exclusive-joy-reid-claims-newly-discovered-homophobic-posts-from-her-blog-were-fabricated/. (2018).

[97] Leo Egghe. 2007. Untangling Herdan's Law and Heaps' Law: Mathematical and Informetric Arguments. *Journal of the American Society for Information Science and Technology*, 58, 5, 702–709.

[98] Charles Elkan. 2001. The Foundations of Cost-Sensitive Learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence* (IJCAI '01), 973–978.

[99] Adam Clark Estes. 2015. Russia Is Banning the Internet Archive and Blaming It On Terrorism. https://gizmodo.com/russia-is-banning-the-internet-archive-and-blaming-it-o-1713926987. (2015).

[100] Gunther Eysenbach. 2006. Going, Going, Still There: Using the WebCite Service to Permanently Archive Cited Web Pages. In *American Medical Informatics Association Annual Symposium*.

[101] Gunther Eysenbach. 2008. Preserving the Scholarly Record With WebCite: An Archiving System For Long-Term Digital Preservation of Cited Webpages. In *Proceedings of the 12th International Conference on Electronic Publishing*, 363–377.

[102] Roy T. Fielding. 2000. *Architectural Styles and the Design of Network-Based Software Architectures*. PhD thesis. University of California.

[103] Roy T. Fielding, James Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, Larry Masinter, Paul J. Leach, and Tim Berners-Lee. 1999. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616. Internet Engineering Task Force, (1999).

[104] Roy T. Fielding, Yves Lafon, and Julian F. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Range Requests. RFC 7233. Internet Engineering Task Force, (2014).

[105] Roy T. Fielding, Mark Nottingham, and Julian F. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Caching. RFC 7234. Internet Engineering Task Force, (2014).

[106] Roy T. Fielding and Julian F. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Authentication. RFC 7235. Internet Engineering Task Force, (2014).

[107] Roy T. Fielding and Julian F. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests. RFC 7232. Internet Engineering Task Force, (2014).

[108] Roy T. Fielding and Julian F. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230. Internet Engineering Task Force, (2014).

[109] Roy T. Fielding and Julian F. Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content. RFC 7231. Internet Engineering Task Force, (2014).

[110] Roy T. Fielding and Richard N. Taylor. 2002. Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT)*, 2, 2, 115–150.

[111] Lynda Schmitz Fuhrig. 2014. Tracking Down the Elusive 'Treasure House for Learning'. `https://siarchives.si.edu/blog/tracking-down-elusive-%E2%80%98treasure-house-learning%E2%80%99`. (2014).

[112] Jean-Loup Gailly and Mark Adler. 2013. GZIP File Format. `http://www.gzip.org/`. (2013).

[113] Jennifer Gavin and Abbie Grotke. 2008. Library Partnership Preserves End-of-Term Government Web Sites. `https://www.loc.gov/item/prn-08-139/library-partnership-saves-government-sites/2008-08-14/`. (2008).

[114] Vinay Goel. 2016. Defining Web Pages, Web Sites and Web Captures. `https://blog.archive.org/2016/10/23/defining-web-pages-web-sites-and-web-captures/`. (2016).

[115] Vinay Goel and Sawood Alam. 2015. A Conversation About URI-M to URI-R Ratio. Private Communication. (2015).

[116] David M. Goldschlag, Michael G. Reed, and Paul F. Syverson. 1996. Hiding Routing Information. In *Proceedings of the International Workshop on Information Hiding, 1996*, 137–150. DOI: `10.1007/3-540-61996-8_37`.

[117] Daniel Gomes, Miguel Costa, David Cruz, João Miranda, and Simão Fontes. 2013. Creating a Billion-Scale Searchable Web Archive. In *Proceedings of the Temporal Web Analytics Workshop* (TempWeb '13), 1059–1066. DOI: 10.1145/2487788.2488118.

[118] Daniel Gomes, João Miranda, and Miguel Costa. 2011. A Survey on Web Archiving Initiatives. In *Proceedings of the 15th International Conference on Theory and Practice of Digital Libraries* (TPDL '11), 408–420. DOI: 10.1007/978-3-642-24469-8_41.

[119] Mark Graham. 2017. Robots.txt Meant for Search Engines Don't Work Well for Web Archives. https://blog.archive.org/2017/04/17/robots-txt-meant-for-search-engines-dont-work-well-for-web-archives/. (2017).

[120] Mark Graham. 2019. The Wayback Machine's Save Page Now is New and Improved. https://blog.archive.org/2019/10/23/the-wayback-machines-save-page-now-is-new-and-improved/. (2019).

[121] Fabrizio Grandoni. 2006. A Note on the Complexity of Minimum Dominating Set. *Journal of Discrete Algorithms*, 4, 2, 209–214. DOI: 10.1016/j.jda.2005.03.002.

[122] Luis Gravano, Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. 1997. STARTS: Stanford Proposal for Internet Meta-Searching. *SIGMOD Record*, 26, 2, 207–218. DOI: 10.1145/253262.253299.

[123] GreatFire.org. 2018. www.archive.org Is 100% Blocked in China. http://archive.is/JdTnl. (2018).

[124] Ed Greengrass. 2000. Information Retrieval: A Survey. https://www.csee.umbc.edu/csee/research/cadip/readings/IR.report.120600.book.pdf. (2000).

[125] Abbie Grotke, Mark Edward Phillips, and George Barnum. 2008. Preserving Public Government Information: The 2008 End of Term Crawl Project. https://digital.library.unt.edu/ark:/67531/metadc28366/. (2008).

[126] Hussam Hallak. 2018. Why We Need Private Web Archives: Almost Two-Thirds of Web Traffic IS NOT Publicly Archivable. https://ws-dl.blogspot.com/2018/07/2018-07-18-why-we-need-private-web.html. (2018).

[127] John Erik Halse. 2016. CDX Server API. https://iipc.github.io/openwayback/api/cdxserver-api.html. (2016).

[128] Joachim Hammer and Jan Fiedler. 2000. Using Mobile Crawlers to Search the Web Efficiently. *International Journal of Computer and Information Science*, 1, 1, 36–58.

[129] David J. Hand and Keming Yu. 2001. Idiot's Bayes – Not So Stupid After All? *International Statistical Review*, 69, 3, 385–398.

[130] Bernhard Haslhofer, Simeon Warner, Carl Lagoze, Martin Klein, Robert Sanderson, Michael L. Nelson, and Herbert Van de Sompel. 2013. ResourceSync: Leveraging Sitemaps for Resource Synchronization. Technical report arXiv:1305.1476.

[131] Anne Helmond and Fernando N. van der Vlist. 2019. Social Media and Platform Historiography: Challenges and Opportunities. *TMG Journal for Media History*, 22, 1, 6–34. DOI: 10.18146/tmg.434.

[132] Inma Hernández, Carlos R. Rivero, and David Ruiz. 2019. Deep Web Crawling: A Survey. *World Wide Web*, 22, 4, 1577–1610. DOI: 10.1007/s11280-018-0602-1.

[133] Allan Heydon and Marc Najork. 1999. Mercator: A Scalable, Extensible Web Crawler. *World Wide Web*, 2, 4, 219–229. DOI: 10.1023/A:1019213109274.

[134] Olga Holownia. 2020. Novel Coronavirus Outbreak: Help Us Collect Websites. https://netpreserveblog.wordpress.com/2020/02/13/cdg-collection-novel-coronavirus/. (2020).

[135] Karolina Holub, Ingeborg Rudomino, and Marta Matijevic. 2020. The Croatian Web Archive – What's New? https://netpreserveblog.wordpress.com/2020/07/15/the-croatian-web-archive-whats-new/. (2020).

[136] International Internet Preservation Consortium. 2020. Novel Coronavirus (COVID-19). https://archive-it.org/collections/13529. (2020).

[137] Internet Archive. 2006. Archive-It - Web Archiving Services for Libraries and Archives. https://www.archive-it.org/. (2006).

[138] Internet Archive. 2003. CDX File Format. http://archive.org/web/researcher/cdx_file_format.php. (2003).

[139] Internet Assigned Numbers Authority. 2019. Uniform Resource Identifier (URI) Schemes. https://www.iana.org/assignments/uri-schemes/uri-schemes.xhtml. (2019).

[140] Internet Assigned Numbers Authority. 2020. Well-Known URIs. https://www.iana.org/assignments/well-known-uris/well-known-uris.xhtml. (2020).

[141] Internet Memory Foundation. 2010. Web Archiving in Europe: A Survey Provided by the Internet Memory Foundation. https://web.archive.org/web/20110523234751/internetmemory.org/images/uploads/Web_Archiving_Survey.pdf. (2010).

[142] ISO 28500:2017. 2017. WARC File Format. https://iso.org/standard/68004.html. (2017).

[143] Andy Jackson. 2015. MementoWeb Client Java. https://github.com/ukwa/mementoweb-client-java/blob/master/src/main/java/uk/bl/wa/memento/client/MementosAggregator.java. (2015).

[144] Andy Jackson. 2014. Messy Web Archive Collections. https://twitter.com/anjacks0n/status/466690812269846528. (2014).

[145] Andy Jackson. 2015. Ten Years of the UK Web Archive: What Have We Saved? https://blogs.bl.uk/webarchive/2015/09/ten-years-of-the-uk-web-archive-what-have-we-saved.html. (2015).

[146] Shawn M. Jones. 2019. Wikis Are Archives: Integrating Memento and Mediawiki. https://ws-dl.blogspot.com/2019/06/2019-06-05-wikis-are-archives.html. (2019).

[147] Shawn M. Jones, Alexander Nwala, Michele C. Weigle, and Michael L. Nelson. 2018. The Many Shapes of Archive-It. In *Proceedings of the 15th International Conference on Digital Preservation* (iPRES '18). https://hdl.handle.net/11353/10.923619.

[148] Brewster Kahle. 2016. Geez, Now Internet Insurance? https://blog.archive.org/2016/06/16/geez-now-internet-insurance/. (2016).

[149] Brewster Kahle. 2018. Users Hitting 'Save Page Now' at 100 Per Second. https://twitter.com/brewster_kahle/status/994380510011928578. (2018).

[150] Nattiya Kanhabua, Philipp Kemkes, Wolfgang Nejdl, Tu Ngoc Nguyen, Felipe Reis, and Nam Khanh Tran. 2016. How to Search the Internet Archive Without Indexing It. In *Proceedings of the 20th International Conference on Theory and Practice of Digital Libraries* (TPDL '16). Volume 9819, 147–160. DOI: 10.1007/978-3-319-43997-6_12.

[151] Leo Kelion. 2017. Bollywood Blocks the Internet Archive. http://www.bbc.com/news/technology-40875528. (2017).

[152] Mat Kelly, Sawood Alam, Michael L. Nelson, and Michele C. Weigle. 2016. Inter-Planetary Wayback: Peer-to-Peer Permanence of Web Archives. In *Proceedings of the 20th International Conference on Theory and Practice of Digital Libraries*, 411–416. DOI: 10.1007/978-3-319-43997-6_35.

[153] Mat Kelly, Lulwah M. Alkwai, Sawood Alam, Michael L. Nelson, Michele C. Weigle, and Herbert Van de Sompel. 2017. Impact of URI Canonicalization on Memento Count. In *Proceedings of the 17th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '17), 303–304. DOI: 10.1109/JCDL.2017.7991601.

[154] Mat Kelly, Lulwah M. Alkwai, Michael L. Nelson, Michele C. Weigle, and Herbert Van de Sompel. 2017. Impact of URI Canonicalization on Memento Count. Technical report arXiv:1703.03302. https://arxiv.org/abs/1703.03302.

[155] Mat Kelly, Michael L. Nelson, and Michele C. Weigle. 2018. A Framework for Aggregating Private and Public Web Archives. In *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries* (JCDL '18), 273–282. DOI: 10.1145/3197026.3197045.

[156] Mat Kelly, Michael L. Nelson, and Michele C. Weigle. 2013. Making Enterprise-Level Archive Tools Accessible for Personal Web Archiving. https://www.slideshare.net/matkelly01/making-enterpriselevel-archive-tools-accessible-for-personal-web-archiving. (2013).

[157] Mat Kelly, Michael L. Nelson, and Michele C. Weigle. 2014. Mink: Integrating the Live and Archived Web Viewing Experience Using Web Browsers and Memento. In *Proceedings of the 14th ACM/IEEE-CS Joint Conference on Digital Libraries*, 469–470. DOI: 10.1109/JCDL.2014.6970229.

[158] Mat Kelly and Michele C. Weigle. 2012. WARCreate - Create Wayback-Consumable WARC Files From Any Webpage. In *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries*. ACM, 437–438. DOI: 10.1145/2232817.2232930.

[159] Matt Kelly. 2019. *Aggregating Private and Public Web Archives Using the Mementity Framework*. PhD thesis. Old Dominion University. DOI: 10.25777/1t52-wz02.

[160] Madian Khabsa and Clyde Lee Giles. 2014. The Number of Scholarly Documents on the Public Web. *PLOS ONE*, 9, 5. DOI: 10.1371/journal.pone.0093949.

[161] Ritu Khare, Yuan An, and Il-Yeol Song. 2010. Understanding Deep Web Search Interfaces: A Survey. *SIGMOD Record*, 39, 1, 33–40. DOI: 10.1145/1860702.1860708.

[162] Martin Klein, Lyudmila Balakireva, and Harihar Shankar. 2019. Evaluating Memento Service Optimizations. In *Proceedings of the 19th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '19), 182–185. DOI: 10.1109/JCDL.2019.00034.

[163] Matthias Klusch, Patrick Kapahnke, Stefan Schulte, Freddy Lécué, and Abraham Bernstein. 2016. Semantic Web Service Search: A Brief Survey. *Künstliche Intelligenz*, 30, 2, 139–147. DOI: 10.1007/s13218-015-0415-7.

[164] Martin Knakal. 2020. Web Bundles: What Are They and Do They Pose a Threat to the Web? https://www.cdn77.com/blog/web-bundles. (2020).

[165] Martijn Koster. 1996. A Method for Web Robots Control. http://www.robotstxt.org/norobots-rfc.txt. (1996).

[166] Ilya Kreymer. 2020. Web Archive Collection Zipped (WACZ) Format. https://github.com/webrecorder/wacz-format. (2020).

[167] Ilya Kreymer and David S. H. Rosenthal. 2016. Guest Post: Ilya Kreymer on oldweb.today. https://blog.dshr.org/2016/01/guest-post-ilya-kreymer-on-oldwebtoday.html. (2016).

[168] Manish Kumar and Rajesh Bhatia. 2016. Design of a Mobile Web Crawler for Hidden Web. In *Proceedings of the 3rd International Conference on Recent Advances in Information Technology (RAIT)*, 186–190.

[169] Steve Lawrence and Clyde Lee Giles. 1998. Searching the World Wide Web. *Science*, 280, 5360, 98–100. DOI: 10.1126/science.280.5360.98.

[170] Kalev Leetaru. 2015. How Much Of The Internet Does The Wayback Machine Really Archive? https://www.forbes.com/sites/kalevleetaru/2015/11/16/how-much-of-the-internet-does-the-wayback-machine-really-archive/. (2015).

[171] Yanni Li, Yuping Wang, and Erfeng Tian. 2012. A New Architecture of an Intelligent Agent-Based Crawler for Domain-Specific Deep Web Databases. In *Proceedings of the 2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology (WI-IAT)*. Volume 1, 656–663.

[172] Ling Liu. 1999. Query Routing in Large-Scale Digital Library Systems. In *Proceedings of the 15th International Conference on Data Engineering*, 154–163. DOI: 10.1109/ICDE.1999.754918.

[173] Julie Beth Lovins. 1968. Development of a Stemming Algorithm. *Mechanical Translation and Computational Linguistics*, 11, 1-2, 22–31.

[174] Jie Lu and James Callan. 2003. Content-Based Retrieval in Hybrid Peer-to-Peer Networks. In *Proceedings of the 2003 ACM CIKM International Conference on Information and Knowledge Management* (CIKM '03), 199–206. DOI: 10.1145/956863.956903.

[175] Jie Lu and Jamie Callan. 2005. Federated Search of Text-Based Digital Libraries in Hierarchical Peer-to-Peer Networks. In *Proceedings of the 27th European Conference on IR Research, Advances in Information Retrieval* (ECIR '05). Volume 3408, 52–66. DOI: 10.1007/978-3-540-31865-1_5.

[176] Marek Majkowski. 2020. MMUniq-Hash. https://github.com/cloudflare/cloudflare-blog/tree/master/2020-02-mmuniq. (2020).

[177] Marek Majkowski. 2020. When Bloom Filters Don't Bloom. https://blog.cloudflare.com/when-bloom-filters-dont-bloom/. (2020).

[178] Vijini Mallawaarachchi, Lakmal Meegahapola, Roshan Alwis, Eranga Nimalarathna, Dulani Meedeniya, and Sampath Jayarathna. 2020. Change Detection and Notification of Webpages: A Survey. *ACM Computing Surveys*, 53, 1. DOI: 10.1145/3369876.

[179] Julien Masanès. 2006. *Web Archiving*. DOI: 10.1007/978-3-540-46332-0.

[180] Luis Meneses, Richard Furuta, and Frank Shipman. 2012. Identifying 'Soft 404' Error Pages: Analyzing the Lexical Signatures of Documents in Distributed Collections. In *Proceedings of the 2nd International Conference on Theory and Practice of Digital Libraries* (TPDL '12). Volume 7489, 197–208. DOI: 10.1007/978-3-642-33290-6_22.

[181] Weiyi Meng, Clement Yu, and King-Lup Liu. 2002. Building Efficient and Effective Metasearch Engines. *ACM Computing Surveys (CSUR)*, 34, 1, 48–89. DOI: 10.1145/505282.505284.

[182] Stephen Merity. 2014. Navigating the WARC File Format. http://commoncrawl.org/2014/04/navigating-the-warc-file-format/. (2014).

[183] Alessandro Micarelli and Fabio Gasparetti. 2007. Adaptive Focused Crawling. *The Adaptive Web*, 4321, 231–262. DOI: 10.1007/978-3-540-72079-9_7.

[184] Gordon Mohr, Michael Stack, Igor Rnitovic, Dan Avery, and Michele Kimpton. 2004. Introduction to Heritrix. In *Proceedings of the 4th International Web Archiving Workshop*.

[185] Mozilla Foundation. 2015. Public Suffix List. https://publicsuffix.org/. (2015).

[186] Michael L. Nelson. 2018. The Internet Archive Can't Preserve the Web's History by Itself. https://motherboard.vice.com/en_us/article/7xdn8y/joy-reid-and-the-weaponization-of-internet-archives. (2018).

[187] Michael L. Nelson. 2018. Why We Need Multiple Web Archives: The Case of blog.reidreport.com. https://ws-dl.blogspot.com/2018/04/2018-04-24-why-we-need-multiple-web.html. (2018).

[188] Michael L. Nelson, Joan A. Smith, Ignacio Garcia del Campo, Herbert Van de Sompel, and Xiaoming Liu. 2006. Efficient, Automatic Web Resource Harvesting. In *Proceedings of the 8th ACM International Workshop on Web Information and Data Management* (WIDM '06), 43–50. DOI: 10.1145/1183550.1183560.

[189] Michael L. Nelson and Herbert Van de Sompel. 2018. Adding the Dimension of Time to HTTP. In *The SAGE Handbook of Web History*.

[190] Bryan Newbold. 2018. Fatcat Design Document (RFC). https://github.com/internetarchive/fatcat/blob/master/fatcat-rfc.md. (2018).

[191] Mark W. Newman and James A. Landay. 2000. Sitemaps, Storyboards, and Specifications: A Sketch of Web Site Design Practice. In *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, Techniques* (DIS '00), 263–274. DOI: 10.1145/347642.347758.

[192] Mark Nottingham. 2010. Web Linking. RFC 5988. Internet Engineering Task Force, (2010).

[193] Mark Nottingham. 2019. Well-Known Uniform Resource Identifiers (URIs), Internet RFC 8615. https://tools.ietf.org/html/rfc8615. (2019).

[194] Mark Nottingham and Robert Sayre. 2005. The Atom Syndication Format. RFC 4287. Internet Engineering Task Force, (2005).

[195] Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho. 2005. Downloading Textual Hidden Web Content Through Keyword Queries. In *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '05), 100–109.

[196] Alexander C. Nwala. 2018. An Exploration of URL Diversity Measures. https://ws-dl.blogspot.com/2018/05/2018-05-04-exploration-of-url-diversity.html. (2018).

[197] Alexander C. Nwala. 2015. I Can Haz Memento. https://ws-dl.blogspot.com/2015/07/2015-07-22-i-can-haz-memento.html. (2015).

[198] Maile Ohye and Joachim Kupke. 2012. The Canonical Link Relation. RFC 6596. Internet Engineering Task Force, (2012).

[199] Open Source Initiative. 2006. The MIT License. https://opensource.org/licenses/MIT. (2006).

[200] Krutarth Patel, Cornelia Caragea, Mark Phillips, and Nathaniel Fox. 2020. Identifying Documents In-Scope of a Collection From Web Archives. In *Proceedings of the 20th ACM/IEEE on Joint Conference on Digital Libraries* (JCDL '20).

[201] Sarah Perez. 2013. It's Not Just Reader - Google Kills Its RSS Subscription Browser Extension, Too. https://techcrunch.com/2013/03/15/google-kills-rss/. (2013).

[202] Rob Pike. 2000. Lexical File Names in Plan 9, or, Getting Dot-Dot Right. In *Proceedings of the General Track: 2000 USENIX Annual Technical Conference*, 85–92.

[203] Sriram Raghavan and Hector Garcia-Molina. 2001. Crawling the Hidden Web. In *Proceedings of the 27th International Conference on Very Large Data Bases* (VLDB '01), 129–138. ISBN: 1-55860-804-4.

[204] Marcel Ras and Sara van Bussel. 2007. Web Archiving User Survey. https://www.kb.nl/sites/default/files/KB_UserSurvey_Webarchive_EN.pdf. (2007).

[205] Andrée Rathemacher. 2020. Perma.cc: Prevent Link Rot in Scholarship. *University Libraries, University of Rhode Island.*

[206] Irina Rish. 2001. An Empirical Study of the Naive Bayes Classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence* number 22. Volume 3, 41–46.

[207] Dennis Ritchie and Ken Thompson. 1974. The UNIX Time-Sharing System. *Communications of the ACM (CACM)*, 17, 7, 365–375. DOI: 10.1145/361011.361061.

[208] Alfonso de la Rocha. 2020. WebBundles Are Built for Content-Addressable Networks. https://adlrocha.substack.com/p/adlrocha-webbundles-are-built-for. (2020).

[209] Lara Rode. 2020. Web Bundles: The Solution to the Internet's Shrinking Attention Span. https://medium.com/@TTTStudios/web-bundles-the-solution-to-the-internets-shrinking-attention-span-210d90d0f86a. (2020).

[210] David S. H. Rosenthal. 2016. Aggregating Web Archives. https://blog.dshr.org/2016/01/aggregating-web-archives.html. (2016).

[211] David S. H. Rosenthal. 2019. Web Packaging for Web Archiving. https://blog.dshr.org/2019/12/web-packaging-for-web-archiving.html. (2019).

[212] Alexis Rossi. 2016. Robots.txt Files and Archiving .gov and .mil Websites. https://blog.archive.org/2016/12/17/robots-txt-gov-mil-websites/. (2016).

[213] RSS Advisory Board. 2009. RSS 2.0 Specification. https://www.rssboard.org/rss-specification. (2009).

[214] Sam Ruby. 2008. RSS 2.0 and Atom 1.0 Compared. https://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared. (2008).

[215] Hany SalahEldeen and Michael L. Nelson. 2012. Losing My Revolution: How Many Resources Shared on Social Media Have Been Lost? In *Proceedings of the 16th International Conference on Theory and Practice of Digital Libraries* (TPDL '12). Volume 7489, 125–137. DOI: 10.1007/978-3-642-33290-6_14.

[216] Robert Sanderson. 2012. Global Web Archive Integration With Memento. In *Proceedings of the 12th ACM/IEEE-CS Joint Conference on Digital Libraries* (JCDL '12), 379–380. DOI: 10.1145/2232817.2232900.

[217] Robert Sanderson, Herbert Van de Sompel, and Michael L. Nelson. 2012. IIPC Memento Aggregator Experiment. http://www.netpreserve.org/sites/default/files/resources/Sanderson.pdf. (2012).

[218] Sabine Schostag. 2020. The Danish Coronavirus Web Collection – Coronavirus on the Curators' Minds. https://netpreserveblog.wordpress.com/2020/07/29/the-danish-coronavirus-web-collection/. (2020).

[219] Barry Schwartz. 2016. Google's Search Knows About Over 130 Trillion Pages. https://searchengineland.com/googles-search-indexes-hits-130-trillion-pages-documents-263378. (2016).

[220] Giuseppe Scrivano and Hrvoje Nikšić. 1996. Wget. https://en.wikipedia.org/wiki/Wget. (1996).

[221] Zach Shelby. 2010. Constrained RESTful Environments (CoRE) Link Format. RFC 6690. Internet Engineering Task Force, (2010).

[222] Chao Shen and Tao Li. 2010. Multi-Document Summarization via the Minimum Dominating Set. In *Proceedings of the 23rd International Conference on Computational Linguistics* (COLING '10), 984–992.

[223] Cheng Sheng, Nan Zhang, Yufei Tao, and Xin Jin. 2012. Optimal Algorithms for Crawling a Hidden Database in the Web. *Proceedings of the VLDB Endowment*, 5, 11, 1112–1123. DOI: 10.14778/2350229.2350232.

[224] Shiva Shivakumar. 2005. Webmaster-friendly. https://googleblog.blogspot.com/2005/06/webmaster-friendly.html. (2005).

[225] Kristinn Sigurðsson. 2016. 3 Things I Shouldn't Have To Tell You About Running a 'Good' Crawler. https://kris-sigur.blogspot.com/2016/02/3-things-i-shouldnt-have-to-tell-you.html. (2016).

[226] Kristinn Sigurðsson. 2015. URI Canonicalization in Web Archiving. https://kris-sigur.blogspot.com/2015/03/uri-canonicalization-in-web-archiving.html. (2015).

[227] Kristinn Sigurðsson, Michael Stack, and Igor Ranitovic. 2006. Heritrix User Manual: Sort-friendly URI Reordering Transform. http://crawler.archive.org/articles/user_manual/glossary.html#surt. (2006).

[228] Manveet Singh. 2014. Clearing Up Confusion – Deep Web vs. Dark Web. https://brightplanet.com/2014/03/27/clearing-confusion-deep-web-vs-dark-web/. (2014).

[229] Saurabh Singh. 2017. India Bans Wayback Machine, Makes It Harder to Catch Liars on the Web. https://www.indiatoday.in/technology/news/story/india-bans-wayback-machine-makes-it-harder-to-catch-liars-on-internet-1028631-2017-08-08. (2017).

[230] Sitemaps.org. 2008. Sitemaps XML Format. https://www.sitemaps.org/protocol.html. (2008).

[231] Peter Snyder. 1990. tmpfs: A Virtual Memory File System. In *Proceedings of the Autumn 1990 EUUG Conference*, 241–248.

[232] Brooke Sopelsa. 2018. MSNBC's Joy Reid Apologizes for 'Insensitive' LGBT Blog Posts. https://www.nbcnews.com/feature/nbc-out/msnbc-s-joy-reid-apologizes-insensitive-lgbt-blog-posts-n826091. (2018).

[233] Stanford Libraries. 2016. Data Formats and APIs. https://library.stanford.edu/projects/web-archiving/research-resources/data-formats-and-apis. (2016).

[234] Stanford University Libraries. 2013. Stanford Web Archive Portal. https://swap.stanford.edu/. (2013).

[235] Daniel Stenberg. 2014. HTTP2 Explained. *Computer Communication Review*, 44, 3, 120–128. DOI: 10.1145/2656877.2656896.

[236] Hunter Stern and Vinay Goel. 2015. Web Archive Transformation (WAT) Specification, Utilities, and Usage Overview. https://webarchive.jira.com/wiki/spaces/Iresearch/pages/14484029/Web+Archive+Transformation+WAT+Specification+Utilities+and+Usage+Overview. (2015).

[237] Atsushi Sugiura and Oren Etzioni. 2000. Query Routing for Web Search Engines: Architecture and Experiments. *Computer Networks*, 33, 1, 417–429.

[238] The GDELT Project. 2015. The Incredibly Short Lifespan of an Online News Article. https://blog.gdeltproject.org/the-incredibly-short-lifespan-of-an-online-news-article/. (2015).

[239] Thanh Tran and Lei Zhang. 2014. Keyword Query Routing. *IEEE Transactions on Knowledge and Data Engineering*, 26, 2, 363–375.

[240] Gail Truman. 2016. Web Archiving Environmental Scan. https://nrs.harvard.edu/urn-3:HUL.InstRepos:25658314. (2016).

[241] Gail Truman and Andrea Goethals. 2016. Web Archiving Environmental Scan. In *Proceedings of the 13th International Conference on Digital Preservation* (iPRES '16). http://hdl.handle.net/11353/10.502842.

[242] UK Web Archive. 2014. Crawled URL Index JISC UK Web Domain Dataset (1996-2013). https://data.webarchive.org.uk/opendata/ukwa.ds.2/cdx/. (2014). DOI: 10.5259/ukwa.ds.2/cdx/1.

[243] Yusuke Utsunomiya and Kenji Baheux. 2019. Get Started With Web Bundles. https://web.dev/web-bundles/. (2019).

[244] Herbert Van de Sompel, Michael L. Nelson, Martin Klein, and Robert Sanderson. 2013. ResourceSync: The NISO/OAI Resource Synchronization Framework. In *Proceedings of the 17th International Conference on Theory and Practice of Digital Libraries* (TPDL '13). Volume 8092, 488–489. DOI: 10.1007/978-3-642-40501-3_70.

[245] Herbert Van de Sompel, Michael L. Nelson, and Robert Sanderson. 2013. HTTP Framework for Time-Based Access to Resource States – Memento. RFC 7089. Internet Engineering Task Force, (2013).

[246] Herbert Van de Sompel, Michael L. Nelson, Robert Sanderson, Lyudmila L. Balakireva, Scott Ainsworth, and Harihar Shankar. 2009. Memento: Time Travel for the Web. Technical report arXiv:0911.1112. https://arxiv.org/abs/0911.1112.

[247] Antal van den Bosch, Toine Bogers, and Maurice de Kunder. 2016. Estimating Search Engine Index Size Variability: A 9-Year Longitudinal Study. *Scientometrics*, 107, 2, 839–856. DOI: 10.1007/s11192-016-1863-z.

[248] Ping Wu, Ji-Rong Wen, Huan Liu, and Wei-Ying Ma. 2006. Query Selection Techniques for Efficient Crawling of Structured Web Sources. In *Proceedings of the 22nd International Conference on Data Engineering* (ICDE '06), 47–47.

[249] Jeffrey Yasskin. 2020. Loading Signed Exchanges. https://wicg.github.io/webpackage/loading.html. (2020).

[250] Jeffrey Yasskin. 2020. Signed HTTP Exchanges. https://tools.ietf.org/html/draft-yasskin-http-origin-signed-responses-09. (2020).

[251] Jeffrey Yasskin. 2020. Web Bundles. https://tools.ietf.org/html/draft-yasskin-wpack-bundled-exchanges-03. (2020).

[252] Shlomo Yitzhaki. 1979. Relative Deprivation and the Gini Coefficient. *The Quarterly Journal of Economics*, 321–324.

[253] Henner Zeller, Lizzi Harvey, and Gary Illyes. 2019. Formalizing the Robots Exclusion Protocol Specification. https://webmasters.googleblog.com/2019/07/rep-id.html. (2019).

[254] Justin Zobel and Alistair Moffat. 2006. Inverted Files for Text Search Engines. *ACM Computing Surveys*, 38, 2, 6. DOI: 10.1145/1132956.1132959.

APPENDIX A

ROBOTS EXCLUSION FILES (ROBOTS.TXT)

```
# People share a lot of sensitive material on Quora - controversial political
# views, workplace gossip and compensation, and negative opinions held of
# companies. Over many years, as they change jobs or change their views, it is
# important that they can delete or anonymize their previously-written answers.
#
# We opt out of the wayback machine because inclusion would allow people to
# discover the identity of authors who had written sensitive answers publicly and
# later had made them anonymous, and because it would prevent authors from being
# able to remove their content from the internet if they change their mind about
# publishing it. As far as we can tell, there is no way for sites to selectively
# programmatically remove content from the archive and so this is the only way
# for us to protect writers. If they open up an API where we can remove content
# from the archive when authors remove it from Quora, but leave the rest of the
# content archived, we would be happy to opt back in. See the page here:
#
# https://archive.org/about/exclude.php
#
# Meanwhile, if you are looking for an older version of any content on Quora, we
# have full edit history tracked and accessible in product (with the exception of
# content that has been removed by the author). You can generally access this by
# clicking on timestamps, or by appending "/log" to the URL of any content page.
#
# For any questions or feedback about this please visit our contact page
# https://help.quora.com/hc/en-us/requests/new

User-agent: ia_archiver
Disallow: /
```

Fig. 80. Quora's robots.txt File Excludes the Internet Archive With a Note About User Privacy

```
# robots.txt for ftp://plan9.bell-labs.com and http://plan9.bell-labs.com
# see  http://web.nexor.co.uk/mak/doc/robots/norobots.html
# and  http://www.conman.org/people/spc/robots2.html
#
# put recently-added features last in case they cause
# the reader to stop reading.
# Visit-time is a GMT time range of allowed access.
# Request-rate for us is a maximum of one page per 5 seconds, at all hours.
# Crawl-delay is apparently only for msnbot.
#
User-agent: Googlebot
User-agent: msnbot
# the corporate crawler
User-agent: LSgsa-crawler
Disallow: /RealAudio/
Disallow: /bl-traces/
Disallow: /fast-os/
Disallow: /hidden/
Disallow: /historic/
Disallow: /incoming/
Disallow: /inferno/
Disallow: /magic/
Disallow: /netlib.depend/
Disallow: /netlib/
Disallow: /p9trace/
Disallow: /plan9/sources/
Disallow: /sources/
Disallow: /tmp/
Disallow: /tripwire/
Visit-time: 0700-1200
Request-rate: 1/5
Crawl-delay: 5

User-agent: *
Disallow: /
#
# Also, note that the prefixes  ftp:// and http://  lead to the
# same files.
#
# the following are aliases for the same machine.
#        cm.bell-labs.com
#        cs.bell-labs.com
#        netlib.bell-labs.com
#        outside.cs.bell-labs.com
#        plan9.bell-labs.com
#        plan9.cs.bell-labs.com
#        www.cs.bell-labs.com
```

Fig. 81. Bell Labs' robots.txt on 2015-04-21 at 15:44:35 (UTC) `https://web.archive.org/web/20150421154435/http://cm.bell-labs.com/robots.txt`

APPENDIX B

DOWNCASING IN SURT

```
/**
 * Sort-friendly URI Reordering Transform.
 *
 * Converts URIs of the form:
 *
 *   scheme://userinfo@domain.tld:port/path?query#fragment
 *
 * ...into...
 *
 *   scheme://(tld,domain,:port@userinfo)/path?query#fragment
 *
 * The '(' ')' characters serve as an unambiguous notice that the so-called
 * 'authority' portion of the URI ([userinfo@]host[:port] in http URIs) has
 * been transformed; the commas prevent confusion with regular hostnames.
 *
 * This remedies the 'problem' with standard URIs that the host portion of a
 * regular URI, with its dotted-domains, is actually in reverse order from
 * the natural hierarchy that's usually helpful for grouping and sorting.
 *
 * The value of respecting URI case variance is considered negligible: it
 * is vanishingly rare for case-variance to be meaningful, while URI case-
 * variance often arises from people's confusion or sloppiness, and they
 * only correct it insofar as necessary to avoid blatant problems. Thus
 * the usual SURT form is considered to be flattened to all lowercase, and
 * not completely reversible.
 *
 * @author gojomo
 */
public class SURT {
    // [... TRUNCATED ...]
}
```

Fig. 82. Original SURT Implementation in Java `https://github.com/iipc/webarchive-commons/blob/master/src/main/java/org/archive/util/SURT.java`

APPENDIX C

TOOLS HELP MANUALS

```
$ ipwb -h
usage: ipwb [-h] [-d DAEMON_ADDRESS] [-v] [-u] {index,replay} ...

InterPlanetary Wayback (ipwb)

optional arguments:
  -h, --help            show this help message and exit
  -d DAEMON_ADDRESS, --daemon DAEMON_ADDRESS
                        Multi-address of IPFS daemon (default
                        /dns/localhost/tcp/5001/http)
  -v, --version         Report the version of ipwb
  -u, --update-check    Check whether an updated version of ipwb is available

ipwb commands:
  Invoke using "ipwb <command>", e.g., ipwb replay <cdxjFile>

  {index,replay}
    index               Index a WARC file for replay in ipwb
    replay              Start the ipwb replay system

$ ipwb index -h
usage: ipwb [-h] [-e] [-c] [--compressFirst] [-o OUTFILE] [--debug]
            index <warcPath> [index <warcPath> ...]

Index a WARC file for replay in ipwb

positional arguments:
  index <warcPath>      Path to a WARC[.gz] file

optional arguments:
  -h, --help            show this help message and exit
  -e                    Encrypt WARC content prior to adding to IPFS
  -c                    Compress WARC content prior to adding to IPFS
  --compressFirst       Compress data before encryption, where applicable
  -o OUTFILE, --outfile OUTFILE
                        Path to an output CDXJ file, defaults to STDOUT
  --debug               Convenience flag to help with testing and debugging

$ ipwb replay -h
usage: ipwb replay [-h] [-P [<host:port>]] [index]

Start the ipwb relay system

positional arguments:
  index                 path, URI, or multihash of file to use for replay

optional arguments:
  -h, --help            show this help message and exit
  -P [<host:port>], --proxy [<host:port>]
                        Proxy URL
```

Fig. 83. InterPlanetary Wayback CLI Reference

```
// Import Reconstructive module in a ServiceWorker script
importScripts('reconstructive.js')

// Initialize a Reconstructive object with optional customizations
const rc = new Reconstructive({
  urimPattern: `${self.location.origin}/memento/<datetime>/<urir>`,
  bannerElementLocation:`${self.location.origin}/reconstructive-banner.js`,
  bannerLogoLocation:`${self.location.origin}/reconstructive-logo.svg`,
  bannerLogoHref: `${self.location.origin}/`,
  showBanner: true,
  debug: true,
  customColor: '#0C383B'
})

// Add any custom exclusions in addition to the built-in ones
rc.exclusions.analytics = event => event.request.url.endsWith('custom-analytics.js')

// Bind fetch event to reroute requests using the Reconstructive instance
self.addEventListener('fetch', rc.reroute)
```

Fig. 84. Reconstructive Reference

```
<!-- Load a reconstructive-banner custom element in an HTML page -->
<script src="reconstructive-banner.js"></script>

<!-- Add the reconstructive-banner element in the markup with appropriate attribute values -->
<reconstructive-banner
    logo-src=""
    home-href="/"
    urir="https://example.com/"
    memento-datetime="Mon, 06 Feb 2017 00:23:37 GMT"
    first-urim="https://archive.host/memento/20170206002337/https://example.com/"
    first-datetime="Mon, 06 Feb 2017 00:23:37 GMT"
    last-urim="https://archive.host/memento/20170206002337/https://example.com/"
    last-datetime="Mon, 06 Feb 2017 00:23:37 GMT"
    prev-urim=""
    prev-datetime=""
    next-urim=""
    next-datetime="">
</reconstructive-banner>

<!--
  Note: Reconstructive ServiceWorker module injects these with appropriate values automatically
-->
```

Fig. 85. Reconstructive Banner Reference

```
$ memgator -h
   _____                  _______         __
  /      \  ______  _____  / _____/______/  |_____________
 /  Y Y  \/  __ \/     \/  \  ___\__  \   _/ _ \_   _ \
/   | |   \  ___/  Y Y  \   \_\  \/ __ |  | |_| |  | \/
\__/___\__/\_____\__|_|__/\_______/_____|__|\___/|__|

# MemGator ({Version})

A Memento Aggregator CLI and Server in Go

Usage:
  memgator [options] {URI-R}                              # TimeMap from CLI
  memgator [options] {URI-R} {YYYY[MM[DD[hh[mm[ss]]]]]} # Description of the closest Memento
  memgator [options] server                               # Run as a Web Service

Options:
  -A, --agent=MemGator/{Version} <{CONTACT}> User-agent string sent to archives
  -a, --arcs=https://git.io/archives         Local/remote JSON file path/URL of list of archives
  -b, --benchmark=                           Benchmark file location - defaults to Logfile
  -c, --contact=https://git.io/MemGator      Comment/Email/URL/Handle - used in the user-agent
  -D, --static=                              Directory path to serve static assets from
  -d, --dormant=15m0s                        Dormant period after consecutive failures
  -F, --tolerance=-1                         Failure tolerance limit for each archive
  -f, --format=Link                          Output format - Link/JSON/CDXJ
  -H, --host=localhost                       Host name - only used in web service mode
  -k, --topk=-1                              Aggregate only top k archives based on probability
  -l, --log=                                 Log file location - defaults to STDERR
  -m, --monitor=false                        Benchmark monitoring via SSE
  -P, --proxy=http://{HOST}[:{PORT}]{ROOT}   Proxy URL - defaults to host, port, and root
  -p, --port=1208                            Port number - only used in web service mode
  -R, --root=/                               Service root path prefix
  -r, --restimeout=1m0s                      Response timeout for each archive
  -S, --spoof=false                          Spoof each request with a random user-agent
  -T, --hdrtimeout=30s                       Header timeout for each archive
  -t, --contimeout=5s                        Connection timeout for each archive
  -V, --verbose=false                        Show Info and Profiling messages on STDERR
  -v, --version=false                        Show name and version

$ memgator [options] server
TimeMap:  http://localhost:1208/timemap/{FORMAT}/{URI-R}
TimeGate: http://localhost:1208/timegate/{URI-R} [Accept-Datetime]
Memento:  http://localhost:1208/memento[/{FORMAT}|proxy]/{DATETIME}/{URI-R}
About:    http://localhost:1208/about
Monitor:  http://localhost:1208/monitor - (Over SSE, if enabled)

  {FORMAT}          => link|json|cdxj
  {DATETIME}        => YYYY[MM[DD[hh[mm[ss]]]]]
  [Accept-Datetime] => Header in RFC1123 format
```

Fig. 86. MemGator CLI Reference

```
$ accesslog -h
usage: accesslog [options] [FILES ...]

A tool to parse Common Log formatted access logs with various derived fields.

positional arguments:
  files             Log files (plain/gz/bz2) to parse (reads from the STDIN, if empty or '-')

optional arguments:
  -h, --help        Show this help message and exit
  -v, --version     Show version number and exit
  -d, --debug       Show debug messages on STDERR
  -e FIELDS, --nonempty FIELDS
                    Skip record if any of the provided fields is empty (comma separated list)
  -i FIELDS, --valid FIELDS
                    Skip record if any of the provided field values are invalid
                    ('all' or comma separated list from 'host,request,status,size,referrer')
  -m FIELD~RegExp, --match FIELD~RegExp
                    Skip record if field does not match the RegExp (can be used multiple times)
  -t TFORMAT, --origtime TFORMAT
                    Original datetime format of logs (default: '%d/%b/%Y:%H:%M:%S %z')
  -f FORMAT, --format FORMAT
                    Output format string (see available formatting fields below)
  -j FIELDS, --json FIELDS
                    Output NDJSON with the provided fields
                    (use 'all' for all fields except 'origline')

formatting fields:
  {origline}        Original log line
  {host}            IP address of the client
  {identity}        Identity of the client, usually '-'
  {user}            User ID for authentication, usually '-'
  {origtime}        Original date and time (typically in '%d/%b/%Y:%H:%M:%S %z' format)
  {epoch}           Seconds from the Unix epoch (derived from origtime)
  {date}            UTC date in '%Y-%m-%d' format (derived from origtime)
  {time}            UTC time in '%H:%M:%S' format (derived from origtime)
  {datetime}        14 digit datetime in '%Y%m%d%H%M%S' format (derived from origtime)
  {request}         Original HTTP request line
  {method}          HTTP method (empty for invalid request)
  {path}            Path and query (scheme and host removed, empty for invalid request)
  {prefix}          Memento endpoint path prefix (derived from path)
  {mtime}           14 digit Memento datetime (derived from path)
  {rflag}           Memento rewrite flag (derived from path)
  {urir}            Memento URI-R (derived from path)
  {httpv}           HTTP version (empty for invalid request)
  {status}          Returned status code
  {size}            Number of bytes returned
  {referrer}        Referer header (empty, if not logged)
  {agent}           User-agent header (empty, if not logged)
  {extras}          Any additional logged fields
Default FORMAT: '{host} {date} {time} {method} {path} {status} {size} "{referrer}" "{agent}"'
```

Fig. 87. AccessLog Parser CLI Reference

```
$ mementomap -h
usage: mementomap [-h] {generate,compact,lookup,batchlookup} ...

positional arguments:
  {generate,compact,lookup,batchlookup}
    generate            Generate a MementoMap from a sorted file with the
                        first columns as SURT (e.g., CDX/CDXJ)
    compact             Compact a large MementoMap file into a small one
    lookup              Look for a SURT into a MementoMap
    batchlookup         Look for a list of SURTs into a MementoMap

optional arguments:
  -h, --help            show this help message and exit

$ mementomap generate -h
usage: mementomap generate [-h] [--hcf] [--pcf] [--ha] [--pa] [--hk] [--pk]
                           [--hdepth] [--pdepth]
                           infile outfile

positional arguments:
  infile      Input SURT/CDX/CDXJ (plain or GZip) file path or '-' for STDIN
  outfile     Output MementoMap file path

optional arguments:
  -h, --help  show this help message and exit
  --hcf       Host compaction factor (deafault: Inf)
  --pcf       Path compaction factor (deafault: Inf)
  --ha        Power law alpha parameter for host (default: 16.329)
  --pa        Power law alpha parameter for path (default: 24.546)
  --hk        Power law k parameter for host (default: 0.714)
  --pk        Power law k parameter for path (default: 1.429)
  --hdepth    Max host depth (default: 8)
  --pdepth    Max path depth (default: 9)

$ mementomap lookup -h
usage: mementomap lookup [-h] mmap surt

positional arguments:
  mmap        MementoMap file path to look into
  surt        SURT to look for

optional arguments:
  -h, --help  show this help message and exit
```

Fig. 88. MementoMap CLI Reference

VITA

Sawood Alam
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

EDUCATION

Doctor of Philosophy in Computer Science (2020)
Old Dominion University, Norfolk, Virginia, USA
Dissertation: *MementoMap: A Web Archive Profiling Framework for Efficient Memento Routing*

Master of Science in Computer Science (2013)
Old Dominion University, Norfolk, Virginia, USA
Thesis: *HTTP Mailbox - Asynchronous Restful Communication*

Bachelor of Technology in Computer Engineering (2008)
Jamia Millia Islamia, New Delhi, India
Project: *WIDE: Web Integrated Development Environment*

PUBLICATIONS

Publication Updates: `https://ibnesayeed.com/publications`
Google Scholar: `https://scholar.google.com/citations?user=zXdbEQgAAAAJ`

CONTACT

Homepage: `https://ibnesayeed.com/`
Email: `ibnesayeed@gmail.com` (Personal)
`sawood@archive.org` (Work)

Typeset using LaTeX.