Visualizing Digital Collections of Web Archives

Final Report for Columbia Libraries Web Archiving Incentive Program

Michele C. Weigle, Michael L. Nelson
Department of Computer Science
Old Dominion University, Norfolk, VA 23529
{mweigle,mln}@cs.odu.edu

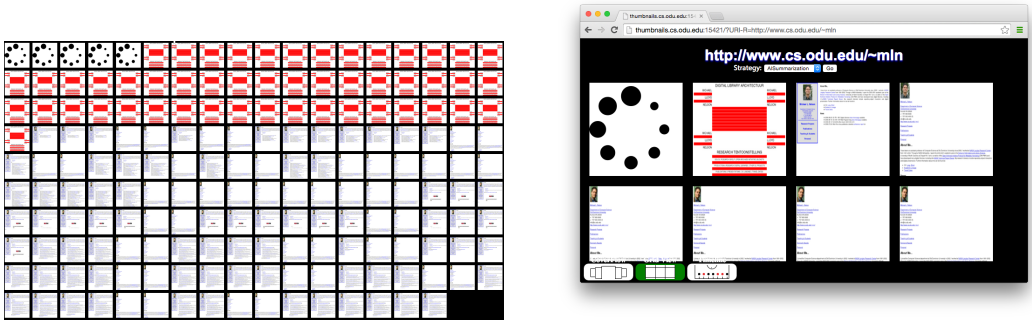
1 Overview

An important problem in web archiving is understanding and presenting how a single page changes over time. This is not only important for researchers, but can also be useful in educating the general public about the temporal and dynamic nature of the web. A common method for presenting webpage change is to display a set of thumbnails of the mementos, or archived pages. Although this can be a useful way to display mementos, a problem arises when there are too many thumbnails to display. For example, the Internet Archive has over 17,000+ mementos for `cnn.com` over a 14 year span. So even if the Internet Archive has thumbnails for all the mementos, some form of sampling will be necessary because the cognitive load of processing all the mementos will be beyond what the user can handle.

Our goal in this work was to develop tools that implement thumbnail summarization for TimeMaps. To do this we have developed the following tools:

- a web service that allows anyone to view a TimeMap using thumbnail summarization,
- a *wayback* add-on that can allow any archive using *wayback* to provide this service for their users,
- an embeddable version to allow web page authors to embed an overview of past versions of their page into the live web version of the page itself; this will help to further the integration of the live web with the past web.

In Figure 1(a) we show the entire set of thumbnails generated from the mementos of `http://www.cs.odu.edu/~mln`. We used our thumbnail summarization technique to reduce the number of thumbnails and provide a more concise summary of the webpage over time, shown in Figure 1(b).



(a) All mementos (b) Summarized view

Figure 1: Mementos from http://www.cs.odu.edu/~mln

2 Work Accomplished

We report on our accomplished work based on the tasks outlined in the original proposal.

Task 1: Thumbnail Summarization Implementation.

Our previous work (AlSum and Nelson, ECIR 2014) provided a theoretical framework for performing thumbnail summarization. This task involves implementing and testing the SimHash similarity and threshold grouping techniques and developing a data structure for storing the metadata (both extracted and derived) for the thumbnail. Once the data structure has been developed, we can apply various visualization techniques to the thumbnails.

Thumbnail Summarization Implementation

The thumbnail summarization process is based on a single URI-R (original resource). Given this URI, a request is sent to the Memento aggregator to obtain a comprehensive list of URI-Ms (archived pages, or mementos, for that URI-R). Each URI-M is then fed into a PhantomJS script that fetches the HTML of the URI. The SimHash algorithm is run on the HTML and the resulting hash is stored in a temporally sorted array (the oldest memento is in slot 0, next newest in slot 1, etc).

We then calculate the Hamming distance between the SimHashes of two mementos to determine significant changes. The algorithm starts with the oldest memento as the baseline and compares the SimHash of consecutive mementos to the baseline. A Hamming distance threshold of $k = 4$ is used based on the work of AlSum and Nelson (ECIR 2014). Once the threshold has been met, that URI-M is recorded and is used as the new baseline, or pivot. The URI-Ms of the mementos calculated as significant are then sent to another PhantomJS script that generates and stores the thumbnails.

The thumbnail summarization code is largely written in Node.js (`https://nodejs.org`). This platform was chosen because it is freely available, frequently used in production systems for scalable, asynchronous processing of data, and serves as a good medium for handling JSON-formatted data. The service can be run on any server with Node.js installed. Once invoked, the server (where the node script resides) can handle requests at `localhost:15421` with URI-R and access as HTTP GET-based query string variables. An example call to the server from a remote client might look like
`http://localhost:15421/?access=interface&URI-R=http://matkelly.com`

Thumbnail Data Structure

Once the thumbnail summarization process is complete, we build a JSON data structure to represent each selected thumbnail. We chose JSON as the data format as it is commonly used in web services. The JSON data structure we developed contains the following fields:

- `captureTimeDelta` - number of seconds between the current memento and the previous memento (-1 if this is the first memento)
- `datetime` - Memento datetime of the current memento
- `rel` - original Link relation value from the TimeMap, as defined in RFC 7089, Section 2.2
- `screenshotURI` - filename of generated thumbnail
- `simhash` - value of SimHash of the HTML of this memento, used for comparing the current memento to the previous memento
- `uri` - URI-M (archived URI) of this memento

An example of the JSON returned is provided below:

```
[{
 captureTimeDelta: -1
 datetime: "Sun, 14 May 2006 12:35:11 GMT"
 rel: "first memento"
 screenshotURI: "httpwebarchiveorgweb20060514123511httpwwwmatkellycom.png"
 simhash: "ca27981f8ad851cf8645ad8a3937aac2"
 uri: "http://web.archive.org/web/20060514123511/http://www.matkelly.com/"
```

```
},
{
 captureTimeDelta: 2090341
 datetime: "Tue, 16 May 2006 21:38:52 GMT"
 hammingDistance: 1
 rel: "memento"
 screenshotURI: "httpwebarchiveorgweb20060516213852httpwwwmatkellycom.png"
 simhash: "ca27981f8ad851cf8645ad8a3937aac3"
 uri: "http://web.archive.org/web/20060516213852/http://www.matkelly.com/"
}
]
```


Caching

A portion of the processing time on querying the service is used in fetching the HTML and generating SimHashes for each memento. To mitigate this expense for subsequent access, we developed a caching system that stores all URI-Ms that resolved to a 200 HTTP status code. The per-URI-R cache files contain a collection of URI-Ms with corresponding SimHash (as generated by the service) and a timestamp, as extracted from the TimeMap. The basis for relying on the datetime of the URI-M as presented by the TimeMap stems from the datetime embedded in the URI-M sometimes not being accurate nor present in archives who obfuscate the URI-M. Caching this returned and derived data greatly increases the speed of thumbnail summarization to the client, even if thumbnails still need to be generated for mementos at runtime (AlSum's "online" generation).

Task 2: Visualization Development and Deployment as a Service.

The web service is modeled on the interfaces from Archive.today and the UK Web Archive, where the user can specify the URI to view in the service's URI. For instance, `http://archive.today/www.bbc.co.uk/` displays the TimeMap of `www.bbc.co.uk` in Archive.today. The web service we have developed provides an interface that allows for a basic grid view of the summarized thumbnails and other interactive views.

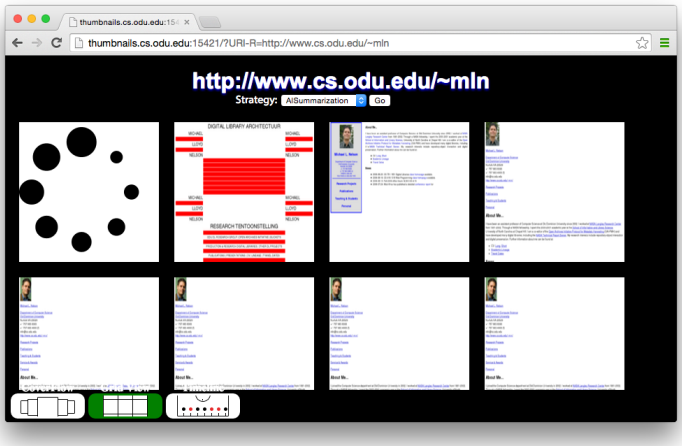
Visualization Development

Figure 2 shows our visualization interfaces. All three of these re-use the same JSON.

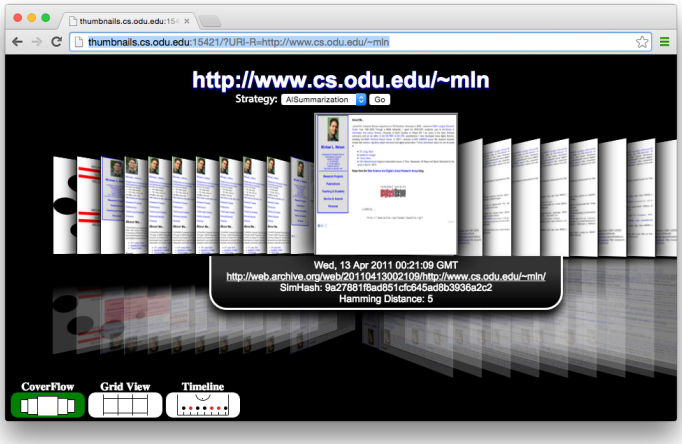
Figure 2(a) shows the basic grid view of the selected thumbnails. These are displayed chronologically. Clicking on an image displays additional information related to the memento. Currently, it displays the associated information stored in the JSON. The downside of this view is that for a large collection to be displayed, the thumbnail size must inversely decrease to prevent images from being shown beyond the viewport and require scrolling while still being large enough for meaningful side-by-side comparison.

Figure 2(b) shows the CoverFlow-like view. The interface method was popularized by Apple Computer in a variety of interfaces and has become a familiar paradigm for displaying a single item in the context of others. The user can flip through the selected thumbnails. Additional information about each memento is displayed below the image. Cover Flow as a means of visualizing thumbnails is sub-optimal, as the items not in focus are distorted or partially hidden (e.g., a second thumbnail versus the one in focus). However, Cover Flow has high information density per pixel compared to a Grid View and an aesthetic effect that can be easily accomplished with CSS transformations for web-based interfaces.

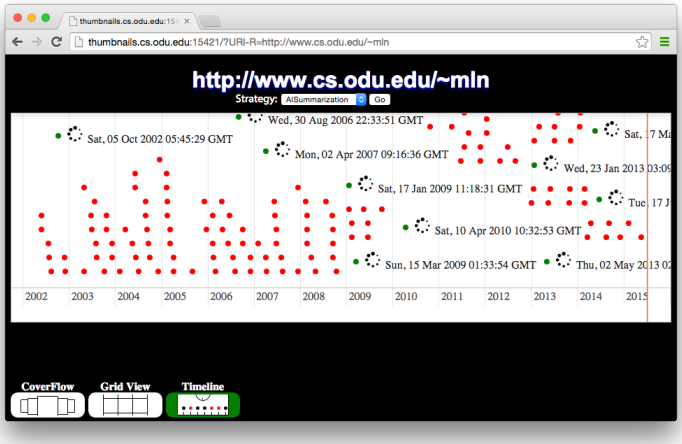
Figure 2(c) shows the timeline view. Each dot on the timeline represents a memento. The green dots represent mementos that were selected for thumbnail generation. A small thumbnail is displayed next to the green dot. The red dots represent mementos that were not selected for thumbnail generation. The user can scroll through time and zoom in and out of the timeline. This view, unlike Grid and Cover Flow, provides the opportunity to also represent data that was not selected in the summary with little space overhead. By displaying single points on a chronologically ordered, date-labeled interface, the non-included data is



(a) Grid View



(b) CoverFlow View



(c) Timeline View

Figure 2: Three methods of thumbnail visualization for the summarized TimeMap.

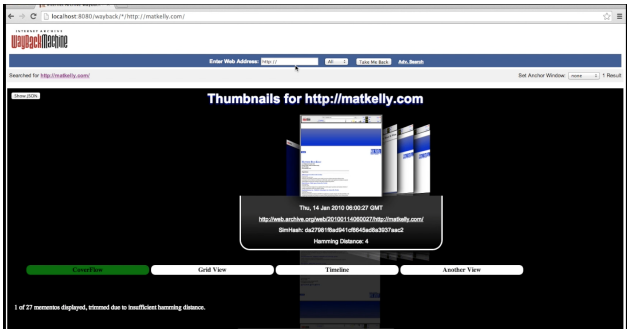


Figure 3: CoverFlow view integrated into Wayback

represented in the context of the mementos selected.

Service Deployment

We have an initial URI scheme that includes an access parameter and the URI-R that is requested. An example is
`http://thumbnails.cs.odu.edu:15421/?access=interface&URI-R=http://www.cs.odu.edu`.

The access parameter in the query string allows the script to serve as an API. Valid values for the access parameter are `interface` (the default if none is specified), `wayback` (for when the script is being included in an archive's user interface), and `embed` (when the script is called from an arbitrary website). Each of these provides appropriate UI elements, callback hooks, and contextual processing. For example, for embedded access, the current page's URI can programmatically be used as the URI-R parameter instead of needing to be explicitly defined by the end-user.

When a client sends a request using the above URI scheme, the summarization server (where the Node.js script resides) receives this request and access type as explicit query string parameters. The server then follows the procedure outlined in Task 1 ("Thumbnail Summarization Implementation") to select the appropriate thumbnails and generate the JSON data structure. Once this process is complete, the server returns JSON to the client that includes a text-based listing of the mementos and URIs of the generated thumbnails, which URI-M they correspond to, and additional metadata about the process (e.g., the calculated Hamming distances). In the process flow, if a URI-M has been considered significant and a thumbnail is to be generated, the server code first checks if one has already been generated and instead supplies the reference to the URI on the server.

In addition to the thumbnail generation logic, the server script also handles the HTML generation for the interfaces, removing any necessary implementation on the client's side. This is accomplished by the script opening two ports, one for access to the thumbnail generation process and a second for access to resources required for the client-side reference implementation. This allows assets of the interface to be separated from thumbnails generated and prevents coupling between the processing logic and the display logic.

Task 3: Wayback Integration.

Most of the members of the IIPC, including various national libraries and archives, use the *wayback* interface and could benefit from advanced visualizations of TimeMaps. We have provided a reference implementation for integration with IIPC's OpenWayback (`http://www.netpreserve.org/openwayback`), which uses the URI query parameter supplied by the user as an alternate interface to the standard "bubble calendar" view supplied in the OpenWayback default implementation. Figure 3 shows the CoverFlow interface integrated into Wayback.

The alternate interface is invoked in the service by supplying an additional `access` parameter in the query string to the service. If `access=wayback` is included, the service displays an interface tailored to

the calling context. In the context of Wayback integration, certain interface elements (e.g., additional metadata, summarization statistics, etc.) are hidden that are only useful when the interface is accessed directly. In the reference implementation with OpenWayback, we accomplished the query by embedding an HTML iframe in an additional results page modeled after the `CalendarResults.jsp` script of OpenWayback. This simple inclusion is more scalable than changing a user's OpenWayback instance's configuration and allows minimal work on the end-user's part to include the additional displays and summarization features for their archives. To facilitate re-use of our thumbnail summarization implementation, we have bundled the modified OpenWayback with our previously publicly deployed tool WAIL, which includes our modified results page for calls to an end-user deployed Thumbnail Summarization service.

Task 4: Integration with the Live Web.

Instead of developing a browser add-on, we have developed a service that allows webpage authors to embed the TimeMap visualizations into the live version of their webpage. This is customizable so that authors can set the visualization type, time range, and maximum number of thumbnails to display. We allow the interface to be embedded on a live web page using HTML embed, a common usage paradigm for web services like YouTube and Twitter. A user can include a snippet of HTML code into their live web page, which provides a user interface element that, when clicked (or by default on page load), queries the service and returns a summarization of the live web in the archives. This means of access is also invoked by setting the `access=embed` parameter in the service URI.

To include the embedded version, a user could include the following in their webpage:

```
<object data="http://thumbnails.cs.odu.edu:15421/?access=embed&URI-R=http://www.cs.odu.edu/~mln"
 type="text/html" width="100%" height="500px">
</object>
```

3 Outcomes and Deliverables

Here we list all of the outcomes and deliverables from the project.

- The thumbnail summarization code is available at `https://github.com/machawk1/ArchiveThumbnails`.
- The thumbnail summarization web service is available at `http://thumbnails.cs.odu.edu:15421`
- The code for integration with OpenWayback is available at `https://github.com/machawk1/ArchiveThumbnails/blob/master/CalendarResults.jsp`

4 Obstacles

We did not encounter any obstacles or issues that affected our ability to complete the proposed work.

5 Presentations and Publications

Here we list presentations and publications that have mentioned this work:

- Mat Kelly, Michael L. Nelson, and Michele C. Weigle. "Efficient Thumbnail Summarization for Web Archives", Digital Preservation, Poster Session, July 2014, Washington, DC, slides at `http://bit.ly/thumbnails-digpres14`
- Michele C. Weigle, "Tools for Managing the Past Web", presentation at the Archive-It Partners Meeting, November 18, 2014, Montgomery, AL, slides at `http://www.slideshare.net/mweigle/2014-weigleaitpublic`
- Mat Kelly, "Visualizing Digital Collections of Web Archives", presentation at the Columbia University Web Archiving Collaboration: New Tools and Models Workshop, June 4, 2015, New York, NY, slides at `http://bit.ly/thumbnails-cuwarc15`