Text Visualization of Historical Newspapers

David Chanin Ian Christopher Mircea Bobby Georgescu T.J. Purtell
Computer Science Department
Stanford University
{chanind,ifc,bgeorges,tpurtell}@.stanford.edu

ABSTRACT

Newspaper archives are a treasure trove of historical information but their sheer volume presents problems for journalists and social scientists trying to understand macroscopic trends. In our work, we sought to address this problem. First, we interviewed a number of journalists and researchers to understand their workflow and the types of questions they are most interested in answering using these archives. Based on this feedback, we developed several prototypes to visualize these archives and iteratively improved our designs with further feedback. In this paper, we propose two different visualizations to help gain insight into these newspaper archives.

INTRODUCTION

Newspapers chronicle current events in great detail and so by utilizing newspaper archives we can gain a window into the past. However these newspaper archives are often massive in size and are filled with detailed text information. Because this information is typically specific to a topic and so large, effective visualization techniques are needed to help researchers pick out macroscopic trends from these archives.

Among the macroscopic trends we initially thought journalists would be interested in were the lifetime of individual stories, the periodic rise and fall of topics, and the coverage of long term events over time (for example a foreign war). All of these examples would be difficult for someone skimming through these archives to identify and measure. After interviewing several journalists, the type of trends we wanted to identify changed but the problem of identifying these macroscopic trends was reaffirmed.

In this paper, we develop two different visualizations to help gain insight into these newspaper archives. Our approach was first interviewing different journalists about their workflows and interests and then moving on to iterative development of prototypes based on their feedback. We conclude with the two visualizations but also propose future work to help merge them together.

The archive data set we used to develop these visualizations has been provided by Geoff McGhee, a Knight Fellow at Stanford's Bill Lane Center for the American West. The data was provided in the form of a thirteen gigabyte archive of more than a decade of articles from the Los Angles Times, Baltimore Sun, and the Chicago Tribune. In total, the set nearly contained two and a half million XML files, each of which corresponded to an article. Additionally the XML files contained metadata such as author, date, section name, page number, title, abstract, and full text of the article.

RELATED WORK

The news corpus that we are working with is one of the most expansive that has been explored visually. Automatic classification of entities [15] as well as interaction network analysis [9] have been performed on the publically available Reuters news data set. This data set covers only a single year so it can not be used to answer the same types of questions our decade of data can answer. Even more interesting questions might be approachable using long-term historcal data sets that reach back to the beginning of the century. The occurrence of terms in relation to local geography has also been studied on large scale news data sets [10].

Creative visualization techniques have been applied to full-text visualization including TileBars which suggessts a small visual cue that indicates the location of term hits within a document [8]. Tools like Text Map Explorer have been used to transform a text corpus into map of interesting features from the data [11]. The most complex and fascinating examples of-text visualization are PhraseNets, which map patterned full-text matching expressions into visual elements [13].

Sentiment analysis is an engaging capability to offer to users of a visualization. Previous sucessful work has focused entirely on display sentiment analysis [14]. We included online sentiment analysis as a capability of system because it mapped directly onto features of our query engine. Rather than focus on crafting sentiment word lists for the news analysis task, we used the General Inquirer database as a source of about a hundred sentiments [1].

Researcher have focused on using named entities extracted from full-text data in order provide a visualization interface that encourages rapid exploration of topics [7]. A tool that is capable of helping users understand the role of people and organizations in the news would be valuable to journalist. However, there are significant limitations of current techniques for automatic entity extraction. Not all extracted entities can be resolved against one another, thus even a single concept within an article may show up under separate names. We chose to use the Stanford CoreNLP tools to facilitate faceted exploration by named entity and part of speech [6, 12]. We address the lack of perfect resolution of terms by allowing a visual metaphor for selecting related terms.

One of our original goals was to use the WordNet database as a means to provide guidance in term selection. Visualization of the taxonomy of Enlgish words usually takes the form of a node link diagram [3, 2]. Docuburst takes a different approach, allowing for a visual fingerprint of document to be derived from the full text combined with WordNet [5].

Unfortunately, these presentations would consume too much space to be of practical use in our visualizations. Intelligent auto-complete against related words is more suitable for our purposes. There are a wide variety of other techniques that could also feed smart term suggestions such as using data extracted from Wikipedia or Yahoo Terms [4].

DISCOVERY INTERVIEWS

Over the course of development, we interviewed the following domain experts to gain insights and iterate on our solutions.

- Christine - Journalism Ph.D. student
- Geoff - Bill Lane Center for the American West at Stanford
- Peter - Consultant and New York Times tech journalist
- Ann - Director of the Graduate School of Journalism at Stanford
- Justin - Knight Fellow, Formerly at the Washington Post
- Phillip - In Charge of Data Visualizations at the Sacramento Bee

Our goal for interviewing these users was to gain a preliminary understanding of how and why journalists and researchers query newspaper archives, as well as to gain a general understanding of the mindset and viewpoint of our users. We were interested in what sorts of questions journalists are trying to answer when they look through newspaper archives, and what tools and processes they employ.

Through these informal preliminary interviews a number of interesting insights emerged. Currently, most journalistic research occurs by querying for docuements directly through tools such as Lexis Nexis or Google. Journalists typically look through these archives when doing research on a person or event for a current story, and are often interested in researching what was said about a person or event in the past. Querying for individual documents is a relatively solved problem through tools such as Lexis Nexis and Google, but no tools exist which can give journalists insights into aggregate trends across a large number of articles. The journalists we talked with view data as a collection of stories in hiding, so any interesting trends which can be discovered in old newspaper archives could themselves be the source of a story.

Since our dataset comprises 3 newspapers which are in decline, there was interest among the people we spoke with about being able to see what that decline means in terms of articles and topics. For instance, as newspaper staffs shrink, are more stories devoted to celebrity gossip rather than hard reporting? Are the topics of stories shifting? Is the sentiment in stories changing over time?

There was also interest from our users in interactive data visualization as the future of journalism. As newspapers are replaced with laptops and ipads, journalists are struggling to find interactive ways to tell stories rather than just using plain text. Currently, creating interactive visualizions online requires newspapers to pull together a team consisting of artists, data analysists, programmers, and journalists. Any tool which can make the process of finding stories in data, creating visualizations, and publishing easier is greatly needed by journalists, especially as staff sizes continue to shrink.

DATA PIPELINE

In order to provide rich plots of the importance of topics over time, it is necessary to parse the full text of the documents. The two most important features to extract are word stem occurrences and named entity instances. The ability to determine the set of relevant documents that contain specific entities or terms is the fundamental building block of our topic visualization. Rather than try to extract topics using machine learning algorithms, our approach is to provide interactive visual query building elements which enable the user to quickly construct an expression which will allow for targeted exploration of the content of a full text corpus.

Interactive query building implies a system which is capable providing feedback to the user about what query terms might be interesting to add to an expression. The terms of interest for the purposes of news exploration will commonly be named entities. For example, one might wish filter the documents down to a set that contain the word election, inspect the list of people who appear in those documents, and plot the frequency of mentions for specific presidential candidates in order to assess the interplay of reporting and the election process. Exploring the mechanics of this query building work flow leads to a few specific queries that must be supported.

- What are all the documents that contain a word derived from stem xyz?
- What are all the documents that contain a specific named entity?
- What are the most important entities mentioned over a specific set of documents?
- What are all the documents that contain a combination of terms, possibly filtered by other features, such as date, page, or publication?
- Given that the user has typed abc, what other terms might they want to include?

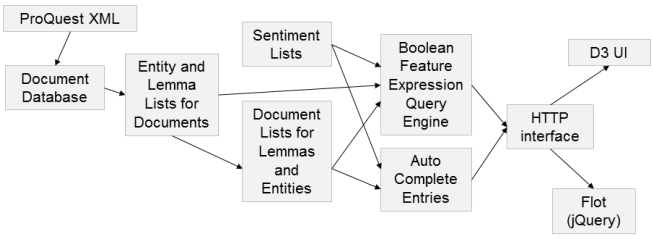

Figure 1. The overall data processing pipeline. The document, entity, and lemma lists are all computed off-line.

Servicing these queries in real-time requires pre-computation of several indexes to provide a responsive user experience. The process of extracting entities and lemmas from the full-text documents alone requires 18 hours of processing time when performed on a Core i7 i920 processor (2.9GHz. 4-core, 8 threads). Unfortunately handling of advanced queries

against the data restricts the type of offline processing that can be used. A user might create a query which selects an arbitrary set of documents and request the most important named entities contained within those documents. Thus, the data need to be stored in a format that supports efficient aggregation.

Our processing pipeline consists of several phases. First, the metadata and full-text fields are extracted from the original ProQuest XML dump using XPath expression. These fields are stored in a SQL database. Next, the Stanford CoreNLP tools are run over the corpus to extract word stems, parts of speech, named entity terms, and named entity classes. A unique id is allocated for word form and entity and the full this mapping is recorded in another pair of SQL tables. Simultaneously, packed and sorted lists of hit counts are produced for the input documents, e.g. word#1 - 10 hits - word#5 - 20 hits - word#1000 - 1 hit. One row per document containing the list is stored in the SQL database. Later, these lists will be rapidly aggregated by the query engine to compute the most important terms over a document collection.

After the full-text has been converted into term hit lists, the lists are transformed so that they can be queried in the other direction. For each entity or word stem, one row is inserted into the database that has a packed list of documents and the frequency of the term within those documents. These are packed and sorted just like the previous hit lists, e.g. document#20 - 5 hits, document #100 - 1 hit, document #253 - 3 hits. The query engine aggregates these lists to determine the set of documents which match a query provided by the user.

Sentiment lists extracted from the General Inquirer database are also imported to allow for online sentiment analysis from the visualization. The sentiments are lists of word stems, so the same code that processes normal user queries can expand the sentiments terms into set of word terms.

Once the core inputs to the query engine have been generated, potential auto-complete candidates are computed. Autocomplete is handled by doing a simple prefix query on an SQL table that contains mappings between input words and potential completion word stems, entities, or sentiment names. Each candidate completion is assigned a score based on its global frequency in the data set; the SQL database sorts the candidates by this score so that the most likely or interesting completions are presented to the user. Since named entities may contain several words including unimportant honorifics, each individual word in an entity is used as an auto-complete key.

QUERY ENGINE

Our query engine allows for arbitrary boolean query expressions to select a set of documents. Documents can be matched based on metadata such as date, publication, or section title. They also can be matched based on whether the document contains a word form or named entity. Each terminal item matches a subset of the entire corpus and returns a packed hit count for how many times that term matches the document. Each internal item aggregates or filters its arguments. Terms can also transform the type of the result sets from a list of document hit to a list of words or entities contained in those documents. An example query in parsed form can be seen in figure 2.

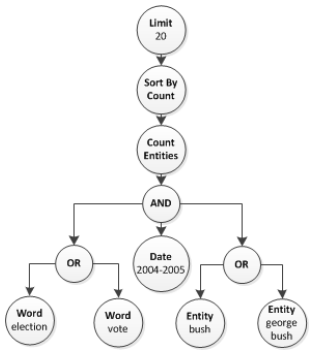

Figure 2. An example query that returns the top 20 entities referenced in articles that mention the election of George Bush.

Queries that return quantitative results, such as document hit counts, request a complete data matrix in a single request. This minimizes overhead due to network latency and the limited number of simultaneous connections provided by a web-browser. As shown in figure 3, the queries have three main parameters: a global filter expression, an array of expressions to aggregate separately, and an array of buckets that filter documents into horizontal axis segments.

```
{
 filter_:YearTerm(2010)
 series_:[
  AllDocsTerm()
 ],
 buckets_:[
  PageTerm(1),
  PageTerm(2),
 ],
}
```

Figure 3. An example query expression in 'JSON' using some syntactic sugar functions.

The query engine is built in Java using the Jersey RESTful web service framework. Combining this framework with the Jackson JSON Processor allows for the automatic mapping of HTTP requests with a JSON payload into Java method calls that take Java objects as arguments. This makes it very easy to build structured query expressions in Javascript and have them processed by the query service.

The query process runs in several phases, figures 4, 5. First all the input expressions are scanned and validated. Once the query expression is validated, the series expressions that were submitted are crossed with the requested aggregation buckets to produce a family of query expressions. Each one is submitted to the evaluator, which checks to see if it has recently processed any similar sub-expressions. If possible it reuses old results as it aggregates the results recursively. Once the evaluator finishes, the specific logic for the query type applies

any additional data transformations required by the front end, e.g. loading the article metadata for matching documents.

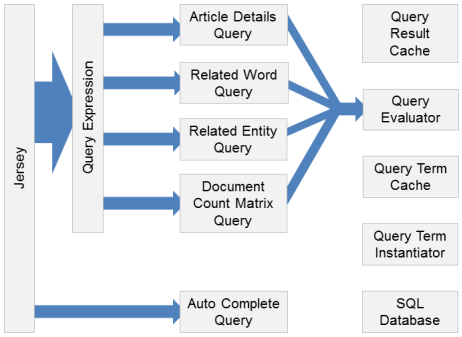

Figure 4. The four main query types are handled by the same query evaluator.

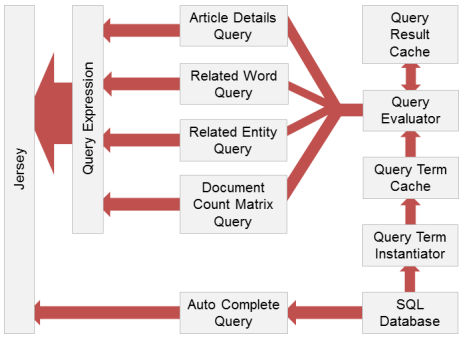

Figure 5. The query evaluator pulls information from the SQL databasse into in memory caches and temporarily caches intermediate results to maximize interactivity.

Terminal nodes imply a specific collection of documents, so the engine caches that intermediate state in memory using Java soft references. If a term has not yet been seen, code is executed against the SQL database to generate the needed intermediate state. Caching the sets of matching documents expressed by individual query terms allows for more interactive performance as queries are modified by the front end. Typically, a query term will be reused many times as a user investigates a topic.

Some of the terms requires a large amount of data access to compute. For example, computing the list of top entities over all documents in the corpus requires the entity list from all 2.5 million documents to be aggregated. All of the database queries that load the entity list are tweaked to maximize the bandwidth that it can read from data. Assuming there is adequate memory to allow for the database files to be cached in memory, the engine pulls over 2GB/s of data as it computes the results. This entire query including data load and aggregation takes less than 15 seconds the first time it is executed, but that is still not fast enough for interactive performance. The engine avoids needing to repeatedly perform similar hard queries by caching the results of each sub-expression that is evaluated.

INITIAL DESIGNS

After our initial interviews and developing the backend, we started to plan and develop our first prototypes. We choose to do this in parallel to hopefully get a more diverse set of visualizations. Ultimately we were left with three different prototypes, though two of which were similar enough to call them different versions of each other. In the next three sections we describe the explorer prototype and two versions of the entity prototype.

EXPLORER 1

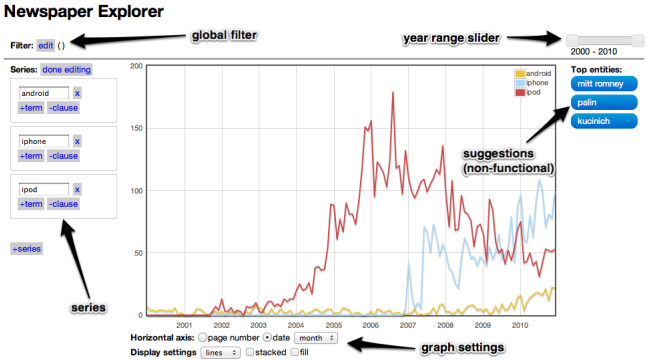

Figure 6. First iteration of Explorer interface

The first iteration of the "Explorer" interface was designed to allow the visualization of the number of articles containing a given term, bucketed either by time or article page number. It borrowed the concept of "series" from Excel and other familiar charting tools, but allowed the specification of series as unions of the sets of documents containing specified words. For instance, a series like "dog OR cat" could be specified, and the series plotted would represent the number of documents containing one or both of these words. Additionally, this version allowed the user to specify a global filter, which would restrict the view to include only articles that match the series queries as well as the filter query. For instance, if one were to use "pet" as the filter and "dog" and "cat" as series, the two plot lines would represent the result of running queries for "pet AND dog" and "pet AND cat." The version allowed the specification of arbitrary conjunctive normal form expressions for the filter. Beyond the series and filter features, this first iteration included a number of other features that are self-explanatory, including a year slider, numerous selection widgets for graph settings.

ENTITY RELATION VIEWER 1.1

The first iteration of the entity relation viewer was designed to allow the user to quickly navigate between terms and entities related to a base query. Similar to how a user of wikipedia will follow links between related articles and discover new material, users of the entity relation viewer can quickly jump from topic to topic in search of a hidden story buried in the data. This visualization was motivated by our interviews in which users expressed the desire to uncover unexpected relationships and trends in the data.

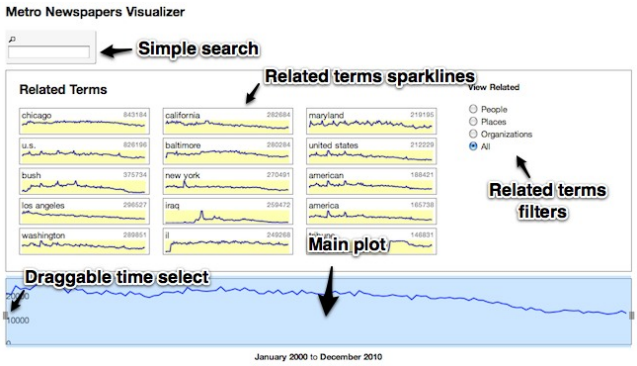

Figure 7. Entity Relation Viewer Version 1

The viewer consists of a search box where a user can enter a query term, and a large sparkline plot at the bottom which shows the number of articles containing that term through time. Above the large sparkline is a list of the top 15 related entities which co-occur in articles with the search term. In addition, each related entity is accompanied by a small sparkline with a preview of that entity's trend over time. This allows the user to see at a glance which related entities contain interesting trends and relationships that may warrant further exploration.

ENTITY RELATION VIEWER 1.2

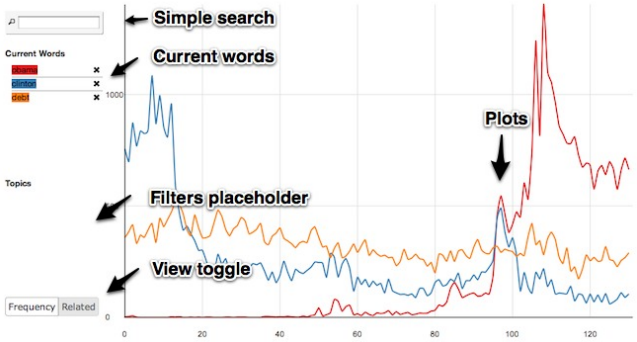

Figure 8. Entity Relation Viewer Version 2, View 1

The second version of the entity relation viewer prototype was designed with many of the design decisions as the first version. Again quick exploration of related terms and a simple query methodology were the heart of the design. This prototype does differ from the first version in one major way. In particular it allows frequency comparisons between different terms and assumes this will be at least as useful of an exploration method. As such it allows people to explore the data in two ways; one for related terms and one for frequency comparisons.

FINAL DESIGNS

After finishing our three prototypes, we sought feedback from several journalists and researchers, and iterated on our original designs.

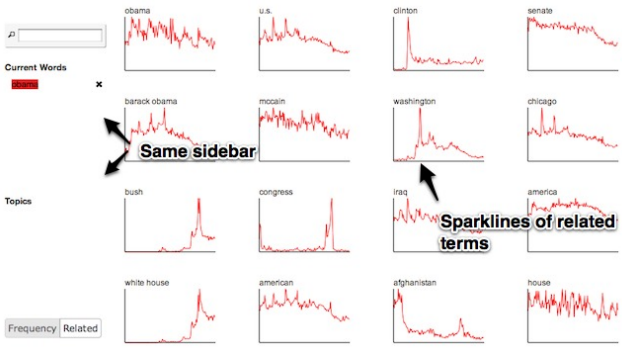

Figure 9. Entity Relation Viewer Version 2, View 2

EXPLORER 2

User testing revealed several shortcomings in first version of the "Explorer" interface. In the second iteration, we sought to address all of these. In this section, we discuss each major piece of feedback and the changes we made in response.

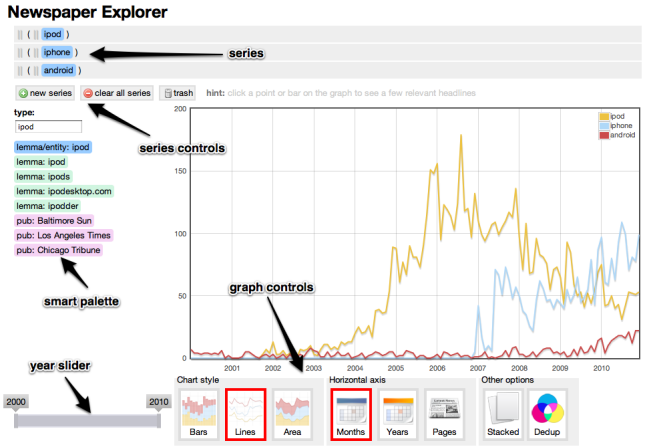

Figure 10. Second iteration of Explorer interface

The interactions required for creating series were very cumbersome and limiting series to disjunctions was too limiting for expressiveness. In order to make series more expressive, we now allow arbitrary conjunctive normal form expressions to be used for series. Additionally, we introduced a type system for terms, with support for Publication, Entity, Lemma, Page, and Sentiment types. This allows for the specification of complex queries like "first page articles mentioning New York in the Baltimore Sun or Los Angeles Times" (see figure). To make constructing and modifying expressions more intuitive, we implemented a drag and drop, touch-friendly interface. Terms and disjunctive clauses can be dragged within and between series. Series can be created by clicking the New Series button or by dropping a term or clause onto it. In the latter case, the new series will be prepopulated with the dropped clause or with a new clause containing the dropped term. Similarly, dropping a term on an existing series but outside an existing clause will create a new clause containing that term. Finally, terms, clauses, and entire

series can be dragged and dropped onto the Trash box, which will result in their deletion.

Figure 11. A complex expression demonstrating the use of conjunctive normal formed and typed terms

Terms originate in the "smart palette" sidebar on the left of the screen. The user types a term in the input box, and has the option to use a lemma/entity term containing the exact text they typed or to choose from a number of auto-complete suggestions that appear below. When possible, the suggested terms are annotated with icons representing the data class (we currently support people, places, and organizations). Terms for the three publications are also available in the palette, and page number terms become available when the user types a number.

Figure 12. Smart palette sidebar populated with typed suggestions

Users were confused about what exactly the filter functionality did. In light of this feedback as well as the increased expressiveness of the series functionality, we decided to remove the global filter in this iteration.

Testers wanted the ability to figure out why spikes occurred in series. In user testing, we found that even in its first iteration, the Explorer tool was fairly effective for finding interesting features in time series. However, our testers expressed frustration when we explained that there was no way to figure out why these features occurred. Part of this had to do with our usage license for the data, which prohibits us from exposing the full-text of the articles. To work around this issue we added the "headline inspector" view, a floating dismissible window that shows up to 20 headlines from the most recently clicked point on the graph. Empirically, we found that scanning the headlines was usually sufficient for testers to identify the cause of a spike.

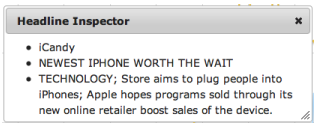

Figure 13. Headline inspector window

The graph controls were unappealing and hard to decode. Specifically, our users indicated that it was difficult to quickly change the various graph settings. We replaced the complex drop downs and radio buttons with large buttons with icons representing their function. This had the added benefit of making these controls more touch-friendly.

The interface did not allow for sharing and collaboration. To remedy this issue, we made the URL update to reflect the state of the graph interface, to allow for sharing of interesting graphs by copying and pasting the contents of the browser address bar. This is the most familiar way of sharing for many users and is also most compatible with various browser plug-ins used for sharing.

ENTITY RELATION VIEWER 2

The entity relation viewer prototype generally received positive feedback. One of the common themes of the demos was that users liked the drill down ability of the explorer prototype, yet liked the simplicity of the entity relation viewer prototype. Ultimately it seemed like we needed to add the ability to drill down further in to the newspaper data.

We decided the best way to do this, while maintaining ease of use, was to link search terms with the related terms a user clicks. That is if the user searched 'Obama' and clicked the 'Chicago' related term, then both related terms and the main plot would update with articles containing 'Obama' and 'Chicago'. We also had a breadcrumb to handle adding these additional filters. However, after a few extra filters, it wasn't uncommon for only a small number of articles to remain. In addition, this broke the free-form linking from term to term idea so we ultimately removed this feature.

Beyond this issue, there were a number of other things we needed to change after receiving feedback.

The ability to drill down to the individual article level was a common request. This actually came up in our initial interviews with journalists to understand the space as well. To address this, we added mouse over listeners on the main plot to open a dialog with appropriate article titles for the selected time. This functionality is shared between the two visualizations.

Testers complained of unintuitive relationships between the sections of the visualization. Users were not sure whether the time slider in the main dialog affected the related term sparklines and seemed to think that the related term sparklines were the primary feature of the visualization. To address this, we placed the main sparkline above the related terms and made it larger.

A few users wanted to see specifically related people to a given person or event. To address this we added a radio switch to see only related people, places, organizations, or all three catagories.

Several users were confused about how the time slider on the main sparkline related to the related term sparklines. To address this, we added a highlight to the small sparklines showing here the currently selected timespan falls.

Several users suggested we add instructions. Because we did not want to take up much space with these instructions we added an information button in the top right of the visualization that has a pop up instructions dialog.

The ability to share views of the prototype seemed important to users. Like the explorer visualization, we added a dynamic URL feature that allows sharing via URL.

The second version of the entity relation viewer, shown in figure 9, was discontinued. Though it received generally positive feedback, it was a difficult prototype to expand in the ways that we envisioned. In particular, it was difficult to add query power while maintaining continuity between its two views. In addition, frequency comparisons did not seem as useful as we first suspected. After stopping its development, we focused on the first version of the visualization described above.

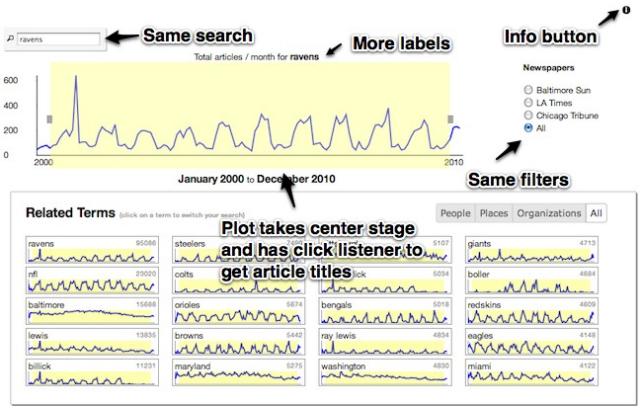

Figure 14. Final version of the entity relation viewer

FUTURE

Moving forward, we would like to continue user testing and gathering feedback on the two interfaces, with the goal of eventually distilling the most useful parts into one single tool. More normalization of the metadata (for instance, sections, authors, etc.) would also be beneficial, and would allow for the exposure of more term types in the interface. Finally, an obvious next step would be indexing the articles of more newspapers beyond the three that we have so far.

REFERENCES

1. General Inquirer. `http://www.wjh.harvard.edu/~inquirer/`.
2. WordVis, the visual dictionary. `http://wordvis.com/`.
3. C. Collins. Wordnet explorer: Applying visualization principles to lexical semantics. *Computational Linguistics Group, Department of Computer Science, University of Toronto, Toronto, Ontario, Canada*, 2006.
4. W. Dakka and P. Ipeirotis. Automatic extraction of useful facet hierarchies from text databases. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 466–475. Ieee, 2008.
5. C. Fellbaum. Wordnet. *Theory and Applications of Ontology: Computer Applications*, pages 231–243, 2010.
6. J. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics, 2005.
7. M. Grobelnik, D. Mladenić, et al. Visualization of news articles. 2004.
8. M. Hearst. Tilebars: visualization of term distribution information in full text information access. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 59–66. ACM Press/Addison-Wesley Publishing Co., 1995.
9. J. Johnson and L. Krempel. Network visualization: The" bush team" in reuters news ticker 9/11-11/15/01. *Journal of Social Structure*, 5, 2004.
10. A. Mehler, Y. Bao, X. Li, Y. Wang, and S. Skiena. Spatial analysis of news sources. *Visualization and Computer Graphics, IEEE Transactions on*, 12(5):765–772, 2006.
11. F. Paulovich and R. Minghim. Text map explorer: a tool to create and explore document maps. In *Information Visualization, 2006. IV 2006. Tenth International Conference on*, pages 245–251. IEEE, 2006.
12. K. Toutanova, D. Klein, C. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 173–180. Association for Computational Linguistics, 2003.
13. F. van Ham, M. Wattenberg, and F. Viegas. Mapping text with phrase nets. *Visualization and Computer Graphics, IEEE Transactions on*, 15(6):1169 –1176, nov.-dec. 2009.
14. F. Wanner, C. Rohrdantz, F. Mansmann, D. Oelke, and D. Keim. *Visual sentiment analysis of rss news feeds featuring the us presidential election in 2008*. Bibliothek der Universität Konstanz, 2009.
15. S. Wermter and C. Hung. Selforganizing classification on the reuters news corpus. In *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pages 1–7. Association for Computational Linguistics, 2002.