2012-03-16

Centralized Visualization of Distributed Collaborative Note-taking

Aaron W. Johnson
Brigham Young University - Provo

Centralized Visualization of Distributed Collaborative Note-taking

Aaron Johnson

A selected project submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Instructional Psychology and Technology

Brigham Young University

April 2012

ABSTRACT

Centralized Visualization of Distributed Collaborative Note-taking

Aaron W. Johnson
Department of Instructional Psychology and Technology, BYU
Master of Science

This development project originated in response to the enormous daily increase of information that becomes available on the internet as a result of social media activities. Twitter, a quintessential example of social media, can also be considered a framework for collaborative note-taking. The October 2010 General Conference of the Church of Jesus Christ of Latter-day Saints provided an interesting example of distributed collaborative note-taking as thousands of Twitter users took notes on the conference addresses. The result was a collection of 26,479 tweets. The purpose of this project is to describe a novel information visualization algorithm that generates a centralized visual representation of the conference tweets to facilitate absorption of the ideas presented therein. This algorithm could feasibly be used in many other massively distributed collaborative note-taking activities. It is hoped that this algorithm, as well as the variant approaches that it may inspire, will assist larger groups to deal with the potential information overload that can arise in these collaborative note-taking activities.

Keywords: collaborative note-taking, information visualization, Twitter, conferences

ACKNOWLEDGMENTS

Above all, I wish to thank my dear, patient wife, who through it all remained my greatest support and the best cheerleader ever. I know it wasn't always easy. This is for you!

Also, I owe a sincere debt of gratitude to my two boys. Your early arrival gave me the necessary motivation and courage to finish this up as quickly as possible.

Next, I thank all the faculty of the Department of Instructional Psychology and Technology at Brigham Young University. Each of you enthusiastically provided me with new tools, new perspectives, and new appreciation.

Finally, I thank my committee members, David Wiley, Peter Rich, and Rick West. It was my singular experience to benefit from your inexhaustible patience, to grow from your wise counsel, and to have a good time along the way.

Table of Contents

List of Tables

List of Figures

Introduction

The recent rise and proliferation of cheap, portable personal computing devices has resulted in convenient access to historically unprecedented amounts of information. This rising swell of available information has made increasingly pertinent the concept of *information overload*, a condition in which individuals find themselves unable to consume an overabundance of information due to the relative scarcity of their own available time and attention.

Information overload is daily becoming a more relevant obstacle to modern education. A significant amount of time and resources in the mainstream educational research community has focused on harnessing the connective capabilities of personal computing devices and using them to facilitate social learning situations. However, the educational benefits of increased connectivity come with potential limiting factors; any social learning situation that encourages students to create and share content could potentially result in a state of information overload.

One such example of an educational activity that can suffer from information overload is collaborative note-taking. In the broadest sense, collaborative note-taking is any activity in which a group of participants share with each other their personal notes describing a topic of common interest. Some collaborative note-taking activities can result in a large amount of shared information, largely textual in nature. If the number of participants is too great, or if the information is shared in real time, a state of information overload can decrease the efficacy of the exercise as participants struggle to consume the incoming notes from other members of the group.

One particularly effective way of coping with information overload is to apply a variety of visualization algorithms to the raw information in order to produce visual summaries of the trends and patterns in the information. Various algorithms have been developed to visualize specific types of data (e.g., numerical, categorical, time series, etc.). The input devices commonly available on most personal computing equipment lend themselves well to text input; the data produced during technology-supported collaborative note-taking activities therefore tends to be textual in nature. Consequently, in order to effectively visualize this information, we must apply an algorithm that is well suited to the visualization of textual data.

This is not always a simple matter. Depending on one's objectives, and owing to the innate complexity of human language, it can be difficult to find a visualization algorithm for textual data that gives insight into questions of interest. Such an algorithm would strike a suitable compromise between highlighting the low-level details of the text (e.g., presence or absence of individual words, phrase counts, etc.) and details of a more contextual, high-level nature (e.g., author sentiment, repetition of ideas, etc.)

This project report will chronicle my efforts to develop a novel visualization algorithm that can be applied to certain types of textual data, striving to convey low-level information (such as word and phrase counts) while still maintaining high-level details and the context of the environment in which the data was gathered. Specifically, I will apply this algorithm to the textual data produced during an instance of massively distributed collaborative note-taking activity.

At present, I will further pursue the concepts of collaborative note-taking, giving examples of several recent collaborative note-taking systems. Additionally, I will further discuss information overload and information visualization in relation to collaborative note-taking. I will follow this with a review of several of the most common text visualization algorithms and then proceed to describe in detail a novel visualization algorithm for textual data.

After giving the implementation details of the algorithm, I will proceed with a discussion of the implications of this algorithm for collaborative note-taking and similar multi-user, text-heavy collaborative activities. Finally, I will conclude this report with a summary of results from a user survey and with an outline for future improvements to the algorithm.

Characteristics of Collaborative Note-taking

In recent years, desktop, laptop, and tablet computers, as well as PDAs, cell phones, video recorders, and even MP3 players are growing in popularity as educational support tools. The interconnected nature of these devices via the Internet and other computer networks provides the ability to quickly share information among members of a group.

The collaborative opportunities that these devices can bring to the classroom have inspired many recent developments in teaching and learning. One such activity is collaborative note-taking (e.g., Denoue, Singh, & Das, 2004), a group learning activity that typically consists of three core elements: multiple collaborators, a topic of interest, and the sharing of topic-relevant information among collaborators.

While the practice of collaborative note-taking is literally centuries old (c.f., the account of team note-taking in 15th-century Florence in Blair, 2008), the progressive rise of personal computing devices in recent decades has provided fertile ground for research in new collaborative note-taking methods. In fact, many recent collaborative note-taking methods entirely move away from traditional pen-and-paper note-taking paradigms and gravitate instead toward personal computing devices as the sole medium for sharing information.

Many different types of technologies can be used in various ways to facilitate collaborative note-taking. For example, Chiu and Chen (2010) explore students’ attitudes toward using wikis and blogs for collaborative note-taking activities. In their study, students in the wiki group assigned responsibility for specific aspects of the lecture topic to each member of the group. During the course of the lecture, group members proceeded to record in wiki pages the facts and ideas relevant to their respective areas of responsibility. After the pages had been created and populated with notes, students in the group could subsequently read and edit group members' pages, completing the collaborative loop by adding their own notes and commentary. Students in the blog group used blogs in a similar manner and for similar purposes.

While Chiu and Chen capitalized on the non-restrictive, generic hardware requirements of web technologies, Berque, Bonebright, and Whitesell (2004) describe DyKnow, a software application with more specific hardware requirements, designed to facilitate collaborative note-taking in computer science classrooms. They describe a context in which each student in a classroom has a tablet computer and uses it to record their notes during a lecture. At any time during the lecture, and with the permission of the student, the instructor can use the DyKnow software to broadcast a student's notes to the tablet screens of everyone else in the classroom.

Another innovative use of personal computing hardware for collaborative note-taking is described by Singh, Denoue, and Das (2005). They developed a software program to enable users of PDAs to participate in collaborative note-taking by connecting the users together and sharing their notes in real time. It also allows users to re-use words from other students' notes, thus eliminating a substantial portion of the burden of text input on these small devices.

These examples begin to give an idea of the many ways in which educators are using personal computing devices to facilitate collaborative note-taking by enabling the sharing of information. A multitude of hardware components, software choices, and options for structuring the interaction of collaborators can be combined in various ways to produce many variations on the basic idea of collaborative note-taking, each tailored to meet the technological and pedagogical needs of a specific group of learners.

Information overload. Despite the many possible forms it can take, collaborative note-taking has always suffered from one striking limitation—information overload. If either the number of collaborators or the scope of the topic of collaboration increases, the members of the collaborative group will find it increasingly difficult to consume all of the information captured and shared by the group because of time and attention constraints. Thus, while two, three, or four collaborators can typically manage to share and consume information effectively, the task of sharing and consuming information among a group of 10 or 20 quickly becomes cumbersome and difficult to sustain. Yet with the advent of social media sites such as Twitter, it is now conceivable to find a group of a hundred, a thousand, or even a million or more collaborators sharing information on a specific topic of interest.

However, while the specific collaborative note-taking activities proposed by researchers in the previous section aim to improve the *sharing* of information, they do little to facilitate the *consumption* of that information. Consequently, many collaborative note-taking methods have evolved in a constrained manner under the assumption that it is necessary to somehow limit the amount of information shared. Yet one can easily imagine scenarios in which each bit of information makes a valid contribution to an understanding of the topic as a whole. In such a scenario it is less than satisfactory to simply sidestep the issue of information overload by imposing an artificial limit on the amount of available information.

Information visualization. Instead of simply reducing the amount of information, we can reduce the magnitude of the information-to-attention mismatch by increasing the efficiency of available time and attention. Visualization algorithms are specifically designed to facilitate the comprehension of large quantities of information, especially in information overload situations where the consumption of that data would otherwise be impossible. They accomplish this by attempting to graphically depict the data in a manner that ensures that similar data appear visually similar, while dissimilar data result in visual contrast. A visualization algorithm thus pre-processes and arranges the data in order to improve the efficiency of a user's time and attention.

By presenting complex data in this way, we translate it into a format that enables us to apply the powerful capabilities of the human visual system. The components of this system constitute a high-bandwidth, low latency information processing network that is able to rapidly consume large amounts of visual information and quickly recognize patterns and trends, despite the complexity of noise in the data. A great deal of this processing occurs on a subconscious level, reducing the cognitive burden on the conscious mind and improving the efficiency of information consumption activities (Ware, 2004).

For these reasons, visualization algorithms in general are of special interest to us as we seek to overcome the potential information overload problems associated with collaborative note-taking. Specifically, visualization algorithms for textual data will be most useful to us, as the data resulting from collaborative note-taking activities tends to be textual in nature. In the next section, I will discuss several existing algorithms for visualizing textual data that are relatively well-known.

Existing Text Visualization Algorithms

Information visualization is an emerging field that is often more art than science, although academic efforts are being made to establish a robust theoretical foundation (Kosara, 2010). Such efforts aim to help the field progress beyond guidelines for best practice toward established, empirically validated methods for constructing optimal visualizations of specific data types.

One theory of information visualization (Wilkinson, 2005; Wickham, 2009) suggests that meaningful visualizations can be created by connecting a finite set of foundational elements together in regular and meaningful ways. This is highly analogous to the way in which we combine individual words using grammatical conventions in order to construct a meaningful sentence.

The logical conclusion of this “grammar of graphics” theory is that an infinite number of unique visualizations may proceed from a finite set of foundational graphical elements. Accordingly, the purpose of this section is not to provide an exhaustive list of all possible textual visualizations, nor will I give a treatment of the aforementioned theory (interested readers may refer to Wilkinson (2005) for a thorough explanation.) Rather, I will present several textual visualizations that provide illustrative examples, setting an appropriate context for the following discussion of my novel visualization algorithm for collaborative note-taking data.

It is worth noting that for the purposes of demonstrating the following text visualization methods, I have selected the first chapter of the novel *Robinson Crusoe* by Defoe (1919) as my sample text data. I selected this text primarily because it is in the public domain and therefore unencumbered by copyright restrictions. Any other textual work in the public domain would likely have served equally well for the purposes of demonstrating these visualization methods.

Word frequency tables. One common approach to the analysis of textual data is to quantify phenomena of interest (Hunston, 2002). Quantification is a preliminary step in many visualization algorithms in general, and many of the most commonly used textual visualization algorithms produce depictions of quantified aspects of a language sample.

The most fundamental example of this is the simple word frequency table. While some may argue that a simple table does not by itself constitute a visualization, it nonetheless performs many of the same functions as a visualization, and its widespread usage as a basic tool of textual analysis warrants mention. We can produce word frequency tables by simply breaking apart a text sample on word boundaries and counting the individual words. We can then arrange this list of word counts in descending order (see Figure 1) to compare the relative frequencies of the words used in the text.

In Figure 1, non-grammatical ("content") words are shown in ***bold italics.*** The table shows that in this text sample the vast majority of the 100 most frequent words are functional or grammatical words (e.g., *a, the, it, this*). It also enables us to easily compare the relative and absolute frequencies of the different words used in this text.

However, we are often not interested in the absolute frequencies of words, but only in their relative frequencies. Note that Figure 1 lists only the 100 most frequent words in the text sample. What if it were necessary to compare *all* of the words in the text? There are 1,162 unique words used in the first chapter of *Robinson Crusoe*. Comparing word frequencies in a table containing all of these words would be a tedious, cumbersome process because the table would necessarily span over several pages. Because we are often interested only in comparing the relative frequencies of words, it may be beneficial to construct a visualization that eliminates the absolute frequency data. Such a visualization would make more efficient use of screen space yet still allow comparison of relative frequencies.

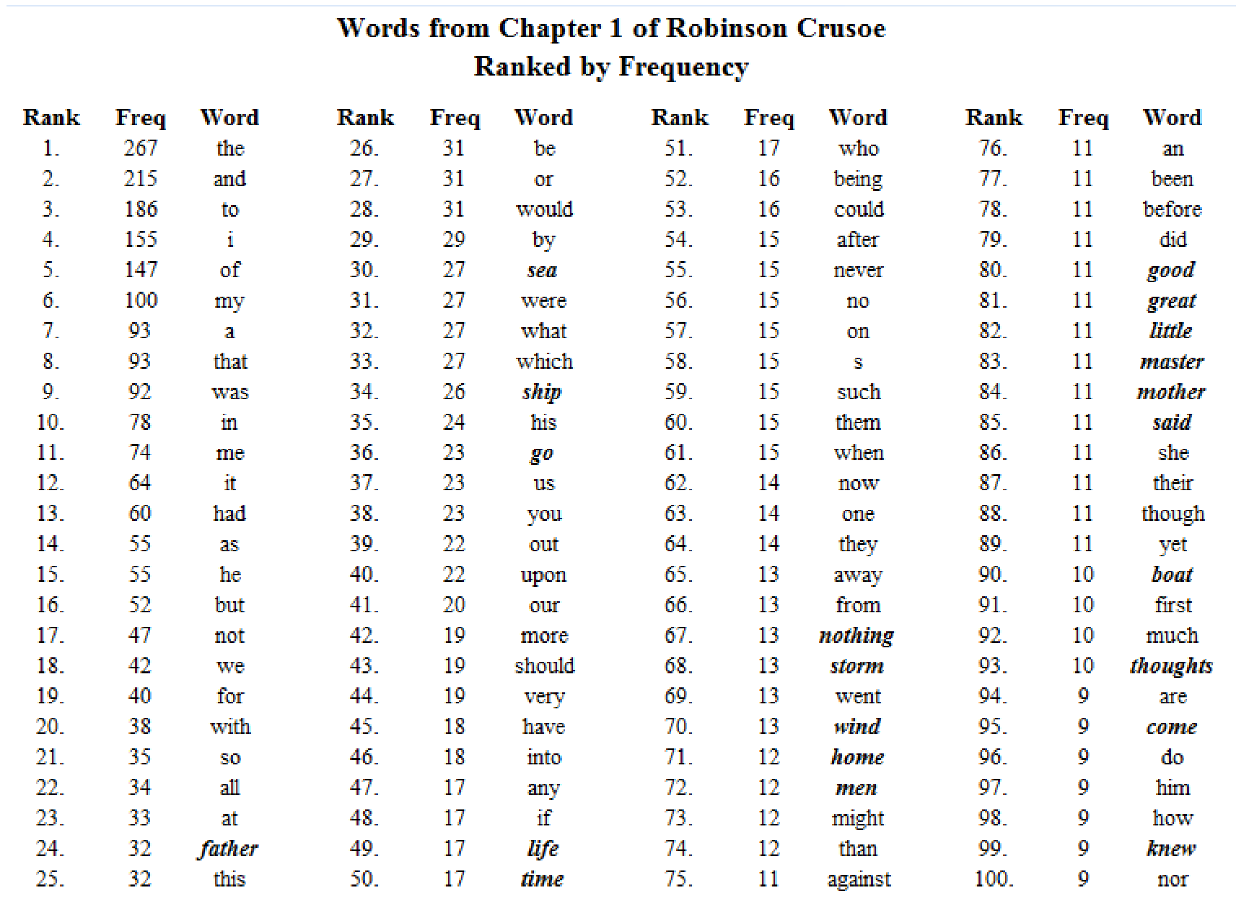
Tag clouds. Tag clouds (see Hearst, 2008 for an interesting treatment) are one possible solution to the problem of excessively large frequency tables. Like frequency tables, tag cloud visualizations are based on the frequency of words in the text sample. Unlike frequency tables, tag clouds do not convey the absolute frequency information. Instead, they communicate the relative frequencies by rendering each word with a font size proportional to its relative frequency.

Figure 2 shows a tag cloud for the same text sample (Chapter 1 from Robinson Crusoe). The tag cloud has a number of advantages over the simple frequency table. One obvious advantage is that the words are given in alphabetical order, enabling one to locate a word of interest more quickly. Another advantage is that common functional words have been removed, allowing for more interesting content words to be displayed.

The words *father, ship, sea*, and *storm* achieve visual prominence due to their greater font size, which is proportional to their frequency relative to other words. By comparing the font size of these words with that of the word *inclination*, for example, one quickly gains a rough understanding of their relative frequencies.

Word clouds. Closely related to tag clouds are so-called “word clouds,” of which the most famous example is probably the “Wordle” (Feinberg, 2011). The Wordle algorithm also produces a simple relative frequency visualization, but in a much more aesthetically pleasing manner.

The Wordle website (http://wordle.net) makes creating these Wordle visualizations simple. By simply copying and pasting text into the web browser, anyone can create a custom



Words from Chapter 1 of Robinson Crusoe
Ranked by Frequency

Rank	Freq	Word	Rank	Freq	Word	Rank	Freq	Word	Rank	Freq	Word
1.	267	the	26.	31	be	51.	17	who	76.	11	an
2.	215	and	27.	31	or	52.	16	being	77.	11	been
3.	186	to	28.	31	would	53.	16	could	78.	11	before
4.	155	i	29.	29	by	54.	15	after	79.	11	did
5.	147	of	30.	27	***sea***	55.	15	never	80.	11	***good***
6.	100	my	31.	27	were	56.	15	no	81.	11	***great***
7.	93	a	32.	27	what	57.	15	on	82.	11	***little***
8.	93	that	33.	27	which	58.	15	s	83.	11	***master***
9.	92	was	34.	26	***ship***	59.	15	such	84.	11	***mother***
10.	78	in	35.	24	his	60.	15	them	85.	11	***said***
11.	74	me	36.	23	***go***	61.	15	when	86.	11	she
12.	64	it	37.	23	us	62.	14	now	87.	11	their
13.	60	had	38.	23	you	63.	14	one	88.	11	though
14.	55	as	39.	22	out	64.	14	they	89.	11	yet
15.	55	he	40.	22	upon	65.	13	away	90.	10	***boat***
16.	52	but	41.	20	our	66.	13	from	91.	10	first
17.	47	not	42.	19	more	67.	13	***nothing***	92.	10	much
18.	42	we	43.	19	should	68.	13	***storm***	93.	10	***thoughts***
19.	40	for	44.	19	very	69.	13	went	94.	9	are
20.	38	with	45.	18	have	70.	13	***wind***	95.	9	***come***
21.	35	so	46.	18	into	71.	12	***home***	96.	9	do
22.	34	all	47.	17	any	72.	12	***men***	97.	9	him
23.	33	at	48.	17	if	73.	12	might	98.	9	how
24.	32	***father***	49.	17	***life***	74.	12	than	99.	9	***knew***
25.	32	this	50.	17	***time***	75.	11	against	100.	9	nor

Figure 1. Sorted Frequency List from Chapter 1 of Robinson Crusoe

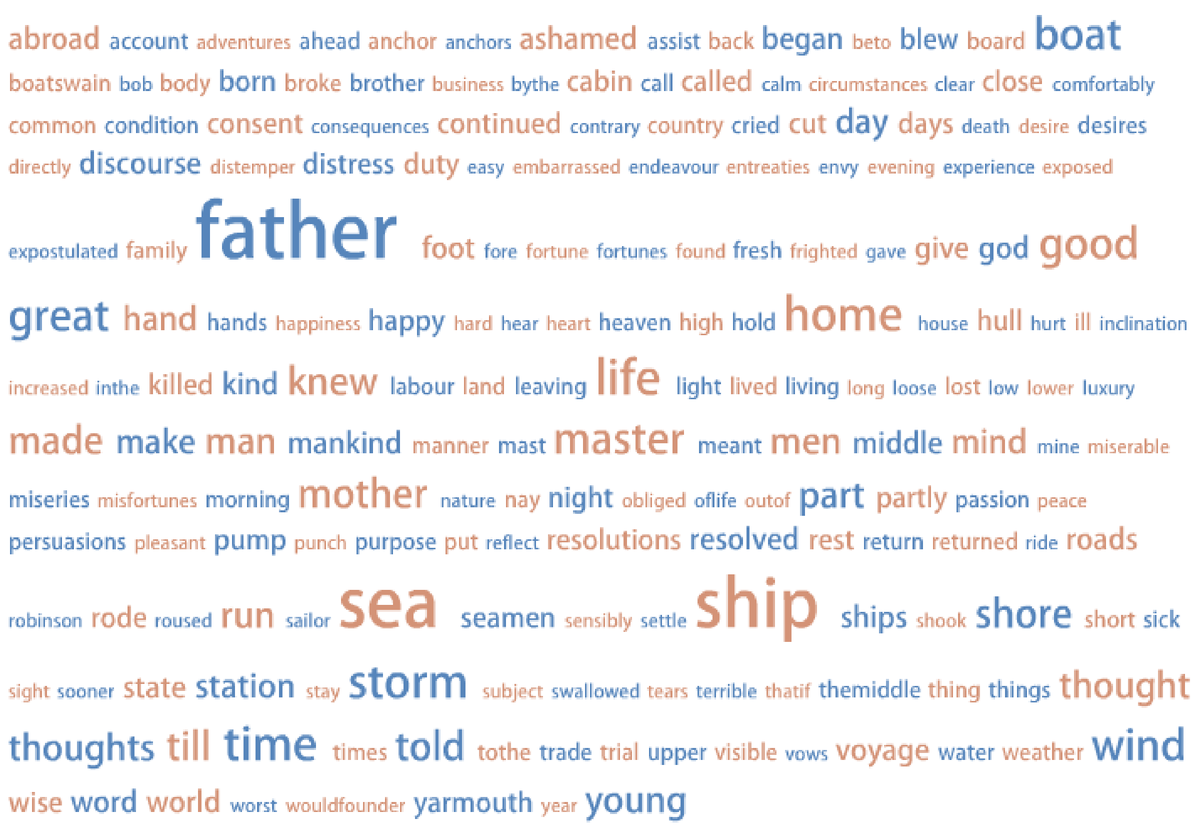


Figure 2. Tag Cloud from Chapter 1 of Robinson Crusoe

Wordle visualization. Consequently, the Wordle has enjoyed immense popularity in recent years.

Although highly popular, Wordles and tag clouds are not beyond criticism. One legitimate complaint against Wordles is that the improved aesthetics of the visualization do nothing to increase comprehension when compared to Wordle’s less glamorous cousin, the tag cloud; Wordle’s various colors have no particular meaning, nor do the orientation and location of words offer any additional insight (see Figure 3). On the contrary, there is some concern that this visualization (and tag clouds in general) actually inhibit the user’s ability to interpret data (Hearst, 2008).

Frequency tables, tag clouds, Wordles, and many similar visualization algorithms can be applied easily to simple quantified attributes such as word frequency. However, it is sometimes desirable to perform a deeper analysis of a text using its more complex characteristics. This typically also requires more complex visualization algorithms.

Concordance lines. One such visualization is the concordance (Manning & Schütze, 1999). Also called KWIC lines (Key Word In Context), concordance lines provide much more information than simple quantitative visualizations by showing words in their immediate usage contexts. For each instance of a specific word in a specific text (or collection of texts), a concordance line set shows a predetermined number of words on either side of the instance of the word of interest. In other words, for a specific word, a set of concordance lines shows how each instance of that word was used in context. The contextual analysis afforded by concordance lines allows a researcher to look for trends and patterns in word usage that would not be evident by considering visualizing simple quantitative aspects of each word.

Typically, concordance lines appear in a specific order. They can be ordered by location in the text, or alphabetically by the words surrounding the word of interest (see Figure 4). The order chosen for the concordance lines can reveal various aspects of word usage. For example, by ordering the lines according to location in the text, it might be possible to identify shifts in the author's usage of certain words over the course of the text.

Figure 4 provides five words of context on each side of every occurrence of *father* and sorts the lines alphabetically according to the context words preceding *father*. This ordering of lines readily reveals that in Chapter 1 of Robinson Crusoe, the word *father* is almost exclusively used to refer to the protagonist's own father, although it shows that another character's father plays a somewhat significant role in this chapter as well ("his father").

Word trees. Similar to the concordance, but with a greater visual emphasis, is the word tree (International Business Machines, 2012a). The word tree combines the proportional font size attributes of the tag cloud with the functionality of a set of concordance lines. The resulting visualization provides contextual examples of word usage, but also quickly reveals the relative frequencies of words (see Figure 5).

Figure 5 reveals the same information as the set of concordance lines in Figure 4, namely, that the majority of occurrences of the word *father* are referring to the protagonist's own father, with several references to another character's father. However, the font size visual cue in the word tree visualization makes these usage patterns evident much more quickly than can be accomplished with a set of concordance lines that appear visually indistinct.

Phrase nets. One final visualization that is very experimental, but worth mentioning, is the Phrase Net (IBM, 2012b). The Phrase Net represents yet another increase in visual complexity. Much like the concordance and Word Tree, the Phrase Net is based on a central

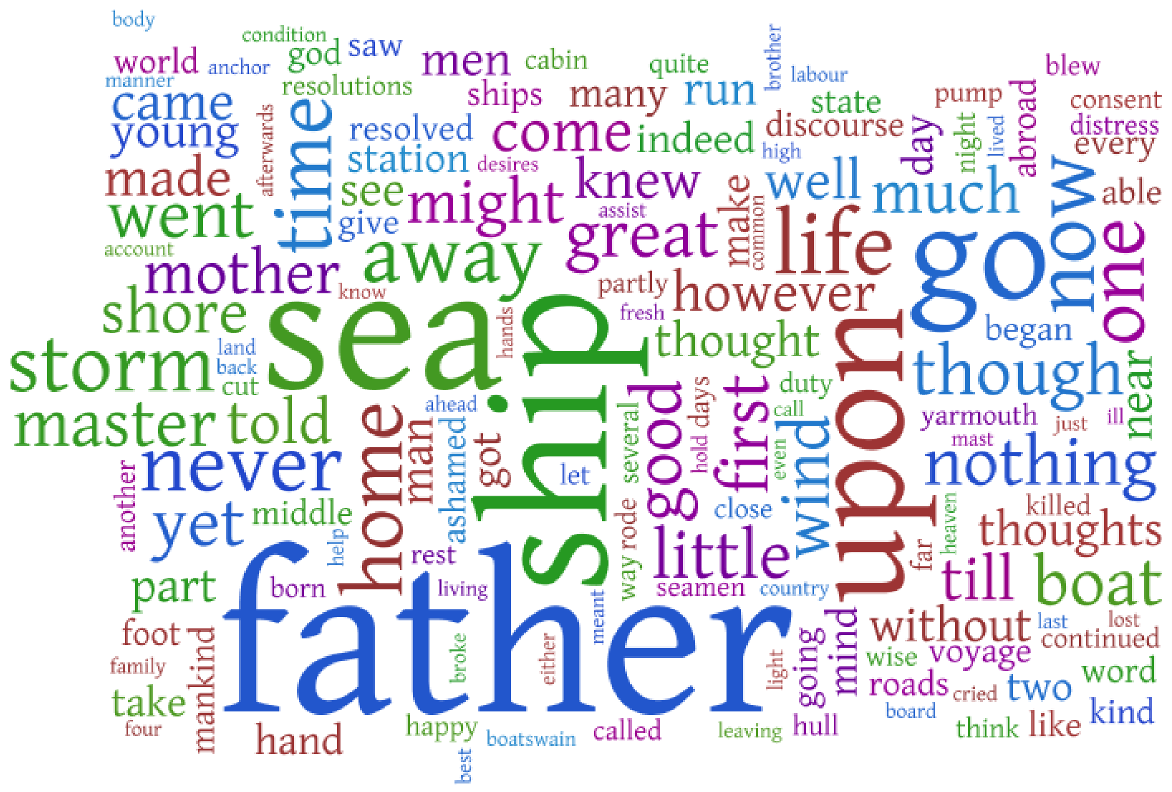
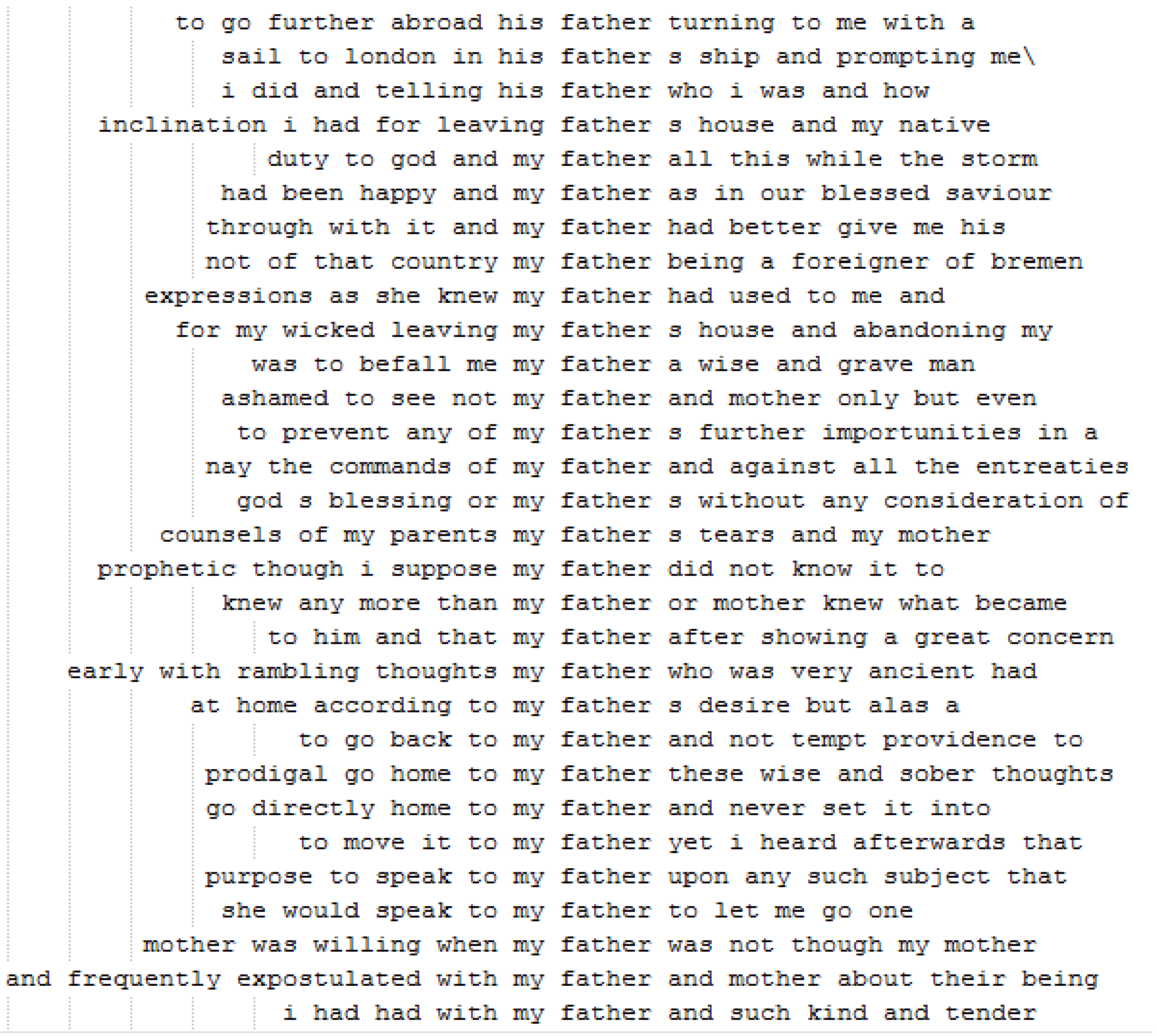


Figure 3. Word Cloud (“Wordle”) from Chapter 1 of Robinson Crusoe



```
           to go further abroad his father turning to me with a
              sail to london in his father s ship and prompting me\
              i did and telling his father who i was and how
      inclination i had for leaving father s house and my native
                 duty to god and my father all this while the storm
              had been happy and my father as in our blessed saviour
             through with it and my father had better give me his
             not of that country my father being a foreigner of bremen
         expressions as she knew my father had used to me and
           for my wicked leaving my father s house and abandoning my
                was to befall me my father a wise and grave man
              ashamed to see not my father and mother only but even
               to prevent any of my father s further importunities in a
             nay the commands of my father and against all the entreaties
               god s blessing or my father s without any consideration of
          counsels of my parents my father s tears and my mother
      prophetic though i suppose my father did not know it to
              knew any more than my father or mother knew what became
                 to him and that my father after showing a great concern
    early with rambling thoughts my father who was very ancient had
            at home according to my father s desire but alas a
                   to go back to my father and not tempt providence to
             prodigal go home to my father these wise and sober thoughts
             go directly home to my father and never set it into
                   to move it to my father yet i heard afterwards that
             purpose to speak to my father upon any such subject that
              she would speak to my father to let me go one
         mother was willing when my father was not though my mother
and frequently expostulated with my father and mother about their being
                  i had had with my father and such kind and tender
```

Figure 4. Concordance Lines for “father” from Chapter 1 of Robinson Crusoe

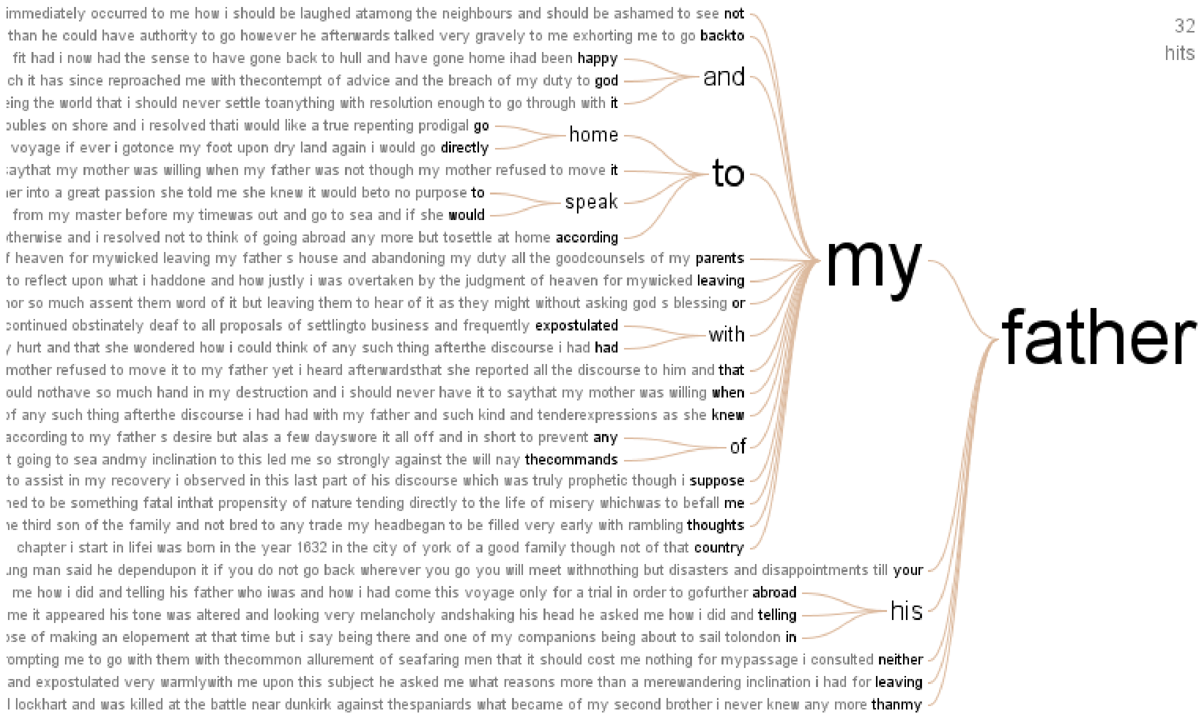

immediately occurred to me how i should be laughed atamong the neighbours and should be ashamed to see not
than he could have authority to go however he afterwards talked very gravely to me exhorting me to go backto
fit had i now had the sense to have gone back to hull and have gone home ihad been happy
ch it has since reproached me with thecontempt of advice and the breach of my duty to god
eing the world that i should never settle toanything with resolution enough to go through with it
oubles on shore and i resolved thati would like a true repenting prodigal go
voyage if ever i gotonce my foot upon dry land again i would go directly
aythat my mother was willing when my father was not though my mother refused to move it
er into a great passion she told me she knew it would beto no purpose to
from my master before my timewas out and go to sea and if she would
therwise and i resolved not to think of going abroad any more but tosettle at home according
f heaven for mywicked leaving my father s house and abandoning my duty all the goodcounsels of my parents
to reflect upon what i haddone and how justly i was overtaken by the judgment of heaven for mywicked leaving
nor so much assent them word of it but leaving them to hear of it as they might without asking god s blessing or
continued obstinately deaf to all proposals of settlingto business and frequently expostulated
hurt and that she wondered how i could think of any such thing afterthe discourse i had had
mother refused to move it to my father yet i heard afterwardsthat she reported all the discourse to him and that
ould nothave so much hand in my destruction and i should never have it to saythat my mother was willing when
of any such thing afterthe discourse i had had with my father and such kind and tenderexpressions as she knew
according to my father s desire but alas a few dayswore it all off and in short to prevent any
t going to sea andmy inclination to this led me so strongly against the will nay thecommands
to assist in my recovery i observed in this last part of his discourse which was truly prophetic though i suppose
ned to be something fatal inthat propensity of nature tending directly to the life of misery whichwas to befall me
ne third son of the family and not bred to any trade my headbegan to be filled very early with rambling thoughts
chapter i start in lifei was born in the year 1632 in the city of york of a good family though not of that country
ung man said he dependupon it if you do not go back wherever you go you will meet withnothing but disasters and disappointments till your
me how i did and telling his father who iwas and how i had come this voyage only for a trial in order to gofurther abroad
me it appeared his tone was altered and looking very melancholy andshaking his head he asked me how i did and telling
ose of making an elopement at that time but i say being there and one of my companions being about to sail tolondon in
ompting me to go with them with thecommon allurement of seafaring men that it should cost me nothing for mypassage i consulted neither
and expostulated very warmlywith me upon this subject he asked me what reasons more than a merewandering inclination i had for leaving
l lockhart and was killed at the battle near dunkirk against thespaniards what became of my second brother i never knew any more thanmy

Figure 5. Word Tree for “father” from Chapter 1 of Robinson Crusoe word of interest. It shows how the word of interest is used as the second word of three-word phrases (for example, ___ *of* ___). The relative frequency of words is again indicated by font size, while the relative frequency of each three word phrase is indicated by the thickness of its respective arrow. That arrow represents the second word of the phrase, the central word of interest. An example of this type of visualization appears in Figure 6, where the central word of interest is *"of"*.

Figure 6 shows that the phrases "course *of* life", "station *of* life", "state *of* life", as well as "part *of* mankind" and "ashamed *of* the" occur more often than other three-word phrases, such as "thoughts *of* it." Such information could be useful for literary analysis, author comparison, or a variety of other tasks.

However, it is worth noting that increased visual complexity does not always result in a commensurate increase in communicative power. The busyness of the Phrase Net visualization requires one to think longer about its meaning, whereas the information conveyed by earlier visualizations was more transparent. This is a good example of how information actually can be obscured when attempting to cover too much information with a single visualization.

Summary of text visualization algorithms. When analyzing collaborative note-taking data, there are a number of aspects of the data that we may wish to consider. We may be interested in topic-specific vocabulary, in which case a simple word frequency table would provide us with information regarding the most commonly used words in the body of notes.

For other reasons, we may be interested in word usage. To help us understand how people are using words and phrases, we could employ concordance lines, word trees, or phrase nets.

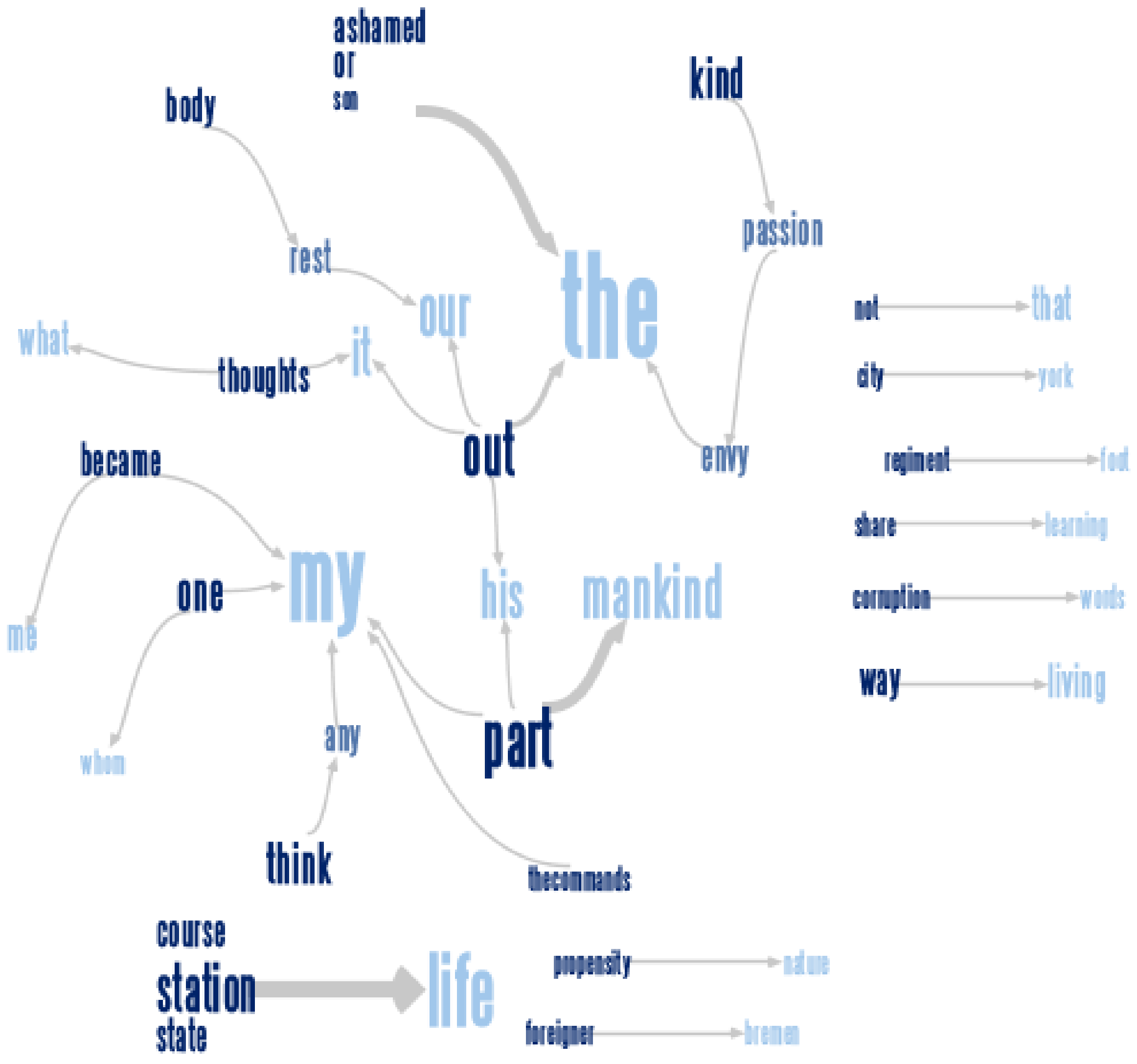


Figure 6. Phrase Net for * of * from Chapter 1 of Robinson Crusoe

However, the basic purpose of collaborative note-taking is to share *information* and to enrich our understanding of the topic of interest based on the ideas we receive from other people. Thus, when we are participating in a collaborative note-taking activity, we are more likely to benefit from those visualizations of the shared notes that convey information about *ideas*, rather than the detailed minutia of the *language* used to express those ideas.

While each of the aforementioned textual visualizations helps us look at details of language used, they each strip individual words and phrases of their context and, consequently, do little to communicate the prominence and significance of ideas expressed. In the remaining sections of this project report, I will describe how I created a new textual visualization algorithm that can in a clear and straightforward manner convey the most significant and frequent ideas expressed in a collection of shared notes.

Method and Findings

From the perspective of the Grammar of Graphics theory discussed earlier, there are infinite possible visualizations for a given data set. Unfortunately, this fact does little to settle the questions of optimality that could arise when discussing visualizations. Additionally, information visualization is an emerging science. Practitioners and academics in this field are still striving to establish a solid foundation of theory, and many diverse opinions exist regarding "best practices."

Consequently, the optimal method of development for this algorithm was unknown in advance. In general, I followed what may be loosely termed a modular and iterative approach to development. The final algorithm emerged in an evolutionary manner as the result of exploration. Amid discussion with colleagues, I refined various aspects of the algorithm numerous times in reply to the feedback I received and the problems that manifested themselves along the way.

Therefore, in the sections that follow, I have chosen not to belabor any explanation of the development methodology itself, but rather to give a clear explanation of the problems encountered during development and the methods and rationale that I used to solve them. Thus, the method and findings are overlapping and interlocking, and are consequently presented as one in this section.

Genesis of Algorithm

The idea for the following visualization algorithm originated from a data exploration issue in which a colleague and I found ourselves with a large dataset, wondering what we could learn from it. The dataset comprised all public tweets with the *#ldsconf* hash tag that were made by Twitter users tuning in to broadcasts of the April 2010 General Conference of the Church of Jesus Christ of Latter-day Saints (2010a).

The collection contained a number of different types of tweets. Among them was a smattering of good-natured nonsense and chatter that is typical to online communications among friends and acquaintances. However, a great deal of these tweets directly referenced the talks given by speakers at the conference. In fact, many tweets contained (semi-)verbatim quotations from the conference talks.

In this context, it is reasonable to assume that the tweets belonging to a specific Twitter user represent ideas and sayings from the talks that resonated with that individual. By recording these ideas and quotes online, the user was engaged in the act of note-taking. By identifying these tweets with the *#ldsconf* hash tag, the user shared these notes with interested persons.

Effectively, the Twitter users tuned in to General Conference were engaged in a form of massively distributed collaborative note-taking.

Having before us a digital record of this collaborative note-taking activity, we began to ask a number of exploratory questions in order to gain a sense of the types of information that could be gleaned from this data set. The one question that figured most prominently in this project was essentially, *Since many of these tweets contain direct quotes, can we find out which phrases were quoted most frequently?* By answering this question, we hoped to identify those tweet-sized statements and phrases—ideas from the conference talks—that were salient to large numbers of people.

For reasons described in the earlier "Information Visualization" section, we felt that some of the best answers to this question could be found through visualization of the tweets themselves. We needed a visualization that would appropriately identify frequently quoted material without removing those quotes from their context. The visualization that we ultimately envisioned comprised the full text of the talk (in order to preserve context) with varying amounts of highlighting commensurately applied to each word or phrase according to the number of times the word or phrase was quoted.

Accordingly, I began developing an algorithm to produce this visualization of quoted material from the *#ldsconf* tweet data that we had obtained.

Acquisition of a More Robust Data Set

After becoming casually acquainted with the original April 2010 tweet data, we found it insufficient for our purposes. Each tweet, as delivered through the Twitter system, has a large number of data fields that describe various attributes of the tweet (e.g., timestamp, the sending user, the receiving user, their respective time zones, etc.) (Twitter, 2012). However, the tweet data from the April 2010 conference, as delivered to us by a third party, had been stripped of most of these fields. We were, unsurprisingly, displeased with the prospect of proceeding with far less data than could otherwise be available.

Additionally, the third-party nature of the data left us unaware of precisely how the data had been collected. We could not with confidence answer even basic questions pertaining to the completeness and representativeness of the tweet data.

We ultimately decided to acquire a new data set of General Conference tweets. This decision seemed appropriate for several reasons. First, the LDS Church holds a General Conference every six months, and at the time here described the next conference (October 2010) was less than two weeks in the future (LDS, 2010b). It was therefore very convenient for us to collect our own data set of tweets from the October 2010 conference. By doing so, we had full control over which tweets were collected and which were filtered out. We also had access to all of the data fields from each tweet, instead of the restricted subset from the April 2010 data set.

Second, we had advanced knowledge of the hash tag—*#ldsconf*—that Twitter users used to label their conference-related tweets. Without knowing the hashtag in advance, it would have been much more difficult to obtain the tweets from the conference.

Third, the community of Twitter users using the *#ldsconf* hash tag had been steadily growing. We expected the October 2010 data set to be substantially larger than the April 2010 set. This additional data would improve the quality of our visualizations, increase the likelihood of discovering trends and patterns in the data, and strengthen any conclusions that might ultimately be drawn.

Downloading the new data set. With this in consideration, I began to write the code that would allow us to collect the *#ldsconf* tweets for the October 2010 General Conference. I ultimately decided to write the code for this part of the project in Python (2012a). Python is a freely-available programming language with a large selection of user-contributed software libraries. One of these libraries, *tweetstream* (Python, 2012b), proved particularly useful for our purposes. This library facilitates access to Twitter's Streaming API (Twitter, 2012b), a service that allows automated programs, such as that which was under construction at the time, to download tweets in real-time based on specific search terms.

Using this Python script, I recorded every *#ldsconf* tweet returned from the Streaming API between Friday, October 1, 2010 and Monday, October 11, 2010—a period of approximately 250 hours. While the conference itself convened only Saturday, October 2 and Sunday, October 3, I ran the script for an extra week in order to catch any post-conference tweets that might arise. In the end, the extended collection period did little to supplement the size of the data set; the vast majority of the 26,479 collected *#ldsconf* tweets occurred on Saturday and Sunday.

The Streaming API returned the tweet data in JSON (Crockford, 2012) format. In the interest of making our data set more accessible and amenable to human analysis, I created another Python script to import the tweet data from the JSON format into a MySQL relational database (Oracle, 2012). Doing so opened our data to SQL queries. This, by itself, gave us the ability to quickly filter and aggregate data on a whim, greatly increasing the efficiency of future analysis.

Obtaining transcripts of the talks. In addition to the tweet data, we also required the transcripts of the 35 talks given during the conference. I waited for several days after the conference for the LDS Church to place the transcripts online in HTML format. I downloaded the talk transcripts (in HTML format) and created a Perl script to eliminate the HTML markup from the downloaded files. The result of this script was a plain-text format better suited for text indexing. This simplified format included only the title of the talk, the speaker's name, a short quote from the talk, and the text of the talk itself. (See Appendix A for an example.)

Finally, I used Apache Lucene (Apache Foundation, 2012) to create an index of the plain-text talk files. There are many text indexing software packages available, and it is likely that any one of them would have sufficed for our purposes. However, Lucene was the most attractive option for several reasons: first, Lucene is freely available due to its open-source license; second, it has been in active development since March 2000, is quite stable, has an established support community, and has a large amount of available documentation; third, Lucene is written for Java, a well-supported general-purpose programming language that is suitable for many diverse types of projects; fourth, I was already familiar with Lucene, having used it in previous projects. For these reasons, Lucene seemed the most likely candidate for fulfilling the text indexing requirements of this project.

I added the text of each talk to a Lucene index. This searchable index served two fundamental purposes: first, it allowed us to quickly identify individual talks containing specific words and phrases; second, it provided location information for words and phrases within the talks themselves.

Analysis: Matching Tweets with Talks

Having gathered the requisite data, the next task was to perform a comparison between each tweet and each of the conference talks. This process ultimately allowed me to associate each tweet with the one talk from which it most likely originated. This process rests on two simplifying, underlying assumptions:

(1) Each *#ldsconf* tweet contains quoted material from a conference talk.

(2) Each *#ldsconf* tweet quotes from only one conference talk.

Correctly identifying a one-to-one talk-to-tweet quote relationship proved to be an interesting challenge that was complicated by two facts: first, quotes demonstrated varying degrees of fidelity to the source material. In general, some individuals will attentively craft a thoroughly verbatim quote, while others will throw together a loose paraphrase of the original speaker's intent. Second, individuals may or may not use syntactic annotations to delineate the boundaries between quoted and non-quoted text; it would not be at all surprising to find in a large set of tweets multiple examples of quoted material that lack quotation marks. It is quite a complex problem to create an algorithm, devoid of the human endowment of intellect and contextual linguistic experience, to identify quotes under these circumstances.

While computers are largely inept at making meaningful interpretations of vague data, they excel at making strict comparisons between specific, well-defined data points. Thus, an approach involving comparisons between tweet text and talk text seemed to be potentially profitable for identifying quotes.

The comparison algorithm, however, requires additional complexity in order to be sufficiently robust to identify quotes that are *nearly verbatim*. Due to the likely inconsistencies between the source text and a quote (for example, a slight grammatical change, a misspelled word, or even a misplaced comma), a strict character-by-character comparison of talk text and tweet text would be extremely unlikely to correctly identify a quote in most situations. Consequently, a simple search query of the verbatim tweet text across all talks would be unlikely to find any matches.

The imperfect quality of the quote data forced me to approach the issue from a different perspective. I decided temporarily to lay aside the strict question of *Does tweet X quote from* *talk Y?* in favor of the less stringent (and arguably more useful) question, *How likely is it that tweet X quotes from talk Y?* From this perspective, the focus shifts from the quote in its entirety to the individual parts that make up the quote.

As the basis for our comparison becomes less complex, the likelihood of matching a tweet with a talk increases. Thus, we expect that a computer would be able to find matches between tweet text and talk text, provided that we make the comparison simple enough. We can then combine the results of these smaller comparisons into a numerical measure of the degree of similarity between a quote and a talk. By deriving a similarity metric, we are able to use straightforward and simple mathematical methods to provide an answer for what was originally a very complex question.

In order to perform smaller, simpler comparisons between tweets and talks, the first step was to reduce each quote to a set of constituent parts. The approach I used was to generate an exhaustive set of n-gram phrases from the text of each quote. In this context, an n-gram is a length n sub-sequence of adjacent words from the text of the tweet. For example, if the text of a tweet were *What did you say?*, then the exhaustive set of n-gram phrases for that tweet would be as shown in Table 1.

Table 1

N-grams for the Phrase 'What did you say?'

n = 1	n = 2	n = 3	n = 4
what	what did	what did you	what did you say
did	did you	did you say	--
you	you say	--	--
say	--	--	--

Breaking up the text of a tweet in this way establishes a set of criteria by which to measure the degree of similarity between the tweet and each of the talks; if a talk contains one or more of the n-gram phrases, then it may be said that the tweet has to this extent quoted from that talk.

Therefore, by using the Lucene text index, we can find out which of the tweet's constituent n-gram phrases are contained in each conference talk. By considering the number of matching n-grams between a tweet and a talk, we can begin to make an intelligent guess as to which talk was most likely the source of the tweet's quote. At first glance, it may seem that the most likely match is the talk that contains the greatest number of these n-gram phrases in common with the tweet. However, as we will shortly see, the situation requires a slightly more complex solution.

For example, it is obvious that a match on the 1-gram "the" would not be very significant; "the" is an extremely common word, and it likely appears many times in each of the conference talks. A 2-gram match such as "the purpose" would be slightly more significant. However, a 2-gram match is likewise not as significant as a match on the 5-gram "the purpose of this life." The probability that the n-gram represents quoted material from the talk clearly increases with the length of the n-gram.

Therefore, the certainty with which we may say that a particular tweet quotes from a particular talk is not simply a function of counting the distinct tweet n-grams that occur within the talk. We must weight the n-grams somehow, such that a 1-gram match weighs less heavily toward establishing a quote relationship, while a 5-gram match would weigh more heavily. Further, a 10- or 15-gram match would all but guarantee that the user had intended to quote from that specific talk, and our weighting system should reflect this.

Accordingly, I chose to weight the n-grams on an exponential scale, proportional to the length of the n-gram. Specifically, I set n-gram weight to be n^2. Therefore, a 2-gram match contributes a value of 4 to the overall relationship score between the tweet and a talk, a 5-gram match contributes 25, and so forth.

As a hypothetical example, assume that we are trying to match one tweet *T* to one of three documents, D_A, D_B, or D_C. After breaking *T* into its constituent n-grams, and searching for each of these n-grams in the three documents, we discover the matches presented in Table 2. After summing the weighted n-gram scores for each of the matches, we select the document with the greatest relationship score for tweet *T*. In this example, D_C earned a score (118) higher than either D_A or D_B (26 and 37, respectively). We would therefore postulate that tweet *T* contained material quoted from D_C.

I processed each of the 26,479 *#ldsconf* tweets in this manner by searching for their respective n-grams in each of the 35 general conference talks. The final result was the one-to-one mapping of postulated tweet-to-talk quote relationships that I had originally sought. Given this information, I was able to separate the tweets into 35 separate groups that would form the basis for the highlighting to be applied to the text of each talk.

Transformation of Language Data into Highlighting Data

Having established a relationship between each talk and a set of tweets, the next step was to create a visualization for each talk based on its associated set of tweets. This visualization uses the metaphor of "highlighting" a document. In other words, the final visualization will appear to be a copy of the conference talk collaboratively highlighted by the entire community. The remainder of this section describes the process of creating the visualization for one

Table 2

Illustrative Example Analysis of Matching Procedure on Hypothetical Data

Document	N-gram Match Length (n)	Weighted Match Score (n^2)
Document A (D_A)		
Match 1	1	1
Match 2	3	9
Match 3	4	16
Final Score for D_A		26
Document B (D_B)		
Match 1	1	1
Match 2	6	36
Final Score for D_B		37
Document C (D_C)		
Match 1	1	1
Match 2	1	1
Match 3	10	100
Match 4	4	16
Final Score for D_C		118

Note: Final Score for each document is the sum of the Weighted Match Score column.

conference talk, given a set of tweets. Ultimately, I performed this process 35 times (once per conference talk) to generate a complete set of visualizations for the conference data.

Conversion of tweet text into highlight magnitude values. To begin, let us consider the final visualization for each talk as the result of merging two separate types of data: text and highlighting. The text data comprises a sequence of words—each word of the talk, in order. The highlighting data is a sequence of numerical values, one for each word in the talk. These numbers determine the intensity of the highlighting each word in the talk will receive in the visualization. A large value in the highlighting data will correspond to a greater intensity of highlighting for its corresponding word. We will next discuss how these numerical highlighting values are calculated.

Initially, each word has a highlighting value of 0 (corresponding to no highlighting). By applying the process that we will hereafter describe to each word in each talk, we increased each word's highlighting value according to the quality and frequency of the quotes contained in the tweets.

Earlier, when seeking to postulate a tweet-talk quote relationship for each tweet, I segmented the text of each tweet into its constituent n-grams as a means of dealing with the inconsistent quality of the quote data. In a similar manner, I segmented the text of each of this talk's tweets into n-grams to facilitate matching them with phrases used in the text of the talk. The locations of these matches determine where to increase the highlighting intensity values. Each n-gram of a particular tweet could occur 0 or more times within the text of the talk.

I used Lucene to search for each n-gram within the text of the talk, resulting in a list of indexed locations for all occurrences of that n-gram within the talk. For each occurrence/location, I increased the corresponding numbers in the highlight data by the match score (n^2). The final value for each number in the highlight data is the sum of the match scores for all matches encompassing the word corresponding to that particular highlight value. Thus, each value in the highlight data is a result both of the frequency with which the corresponding word matched a tweet n-gram and of the quality of the quote (as expressed by the length of the matching n-grams).

Conversion of highlight magnitude values into colors. Having derived the highlighting intensity values for each word in the text of the talk, the next step was to convert these values into actual RGB color values. RGB is the color model in use on most modern display devices. The RGB color model defines a color as the additive combination of three component color values—red, green, and blue. A triplet of numbers corresponding to amounts of red, green, and blue can represent any color in the RGB color space. These numbers are limited to the range [0 - 255]. Mixing various amounts of the three component colors gives rise to the vast array of colors that are possible on RGB display devices: there are 16,777,216 unique color representations in the 24-bit RGB color space.

To begin the process of converting our highlighting intensity values into actual RGB values, I first needed to choose a base highlighting color. Because yellow markers are often used for highlighting text, I initially chose yellow as the highlighting color. However, after several revisions of the visualization, I found that lighter shades of blue are more visible on an LCD monitor when compared to lighter shades of yellow. Thus, I chose to use blue as the highlighting color.

In order to convert the highlighting intensity values for the talk into shades of blue, I first normalized the values by rescaling them to the range [0 - 1], according to the formula $\text{value}_{[0 - 1]}$

= [value – MIN(value)]/[MAX(value) – MIN(value)]. Here, MIN(value) is the minimum highlighting intensity value in this talk (usually 0), and MAX(value) is the maximum highlighting intensity value in this talk. This normalization is advantageous for several reasons. First, it ensures that all values in the highlight data will be between 0 and 1, inclusive. This sets definite, known boundaries to the otherwise unbounded highlight data and also enables consistent, meaningful comparisons among the highlight intensity values of separate talks.

Secondly, by rescaling our highlighting intensity values to the range [0 - 1], they become a convenient multiplier that we can use to create a gradient of colors from white to blue. If blue (R=0, G=0, B=255) is our highlighting color, then white (R=255, G=255, B=255) is the color we would like to use to represent the absence of highlighting. Therefore, we would like our highlighting colors to be in a range from white to blue. If a word has a corresponding normalized highlighting intensity value of 0, it should receive no highlighting (white). If the corresponding normalized intensity value is 1, the word should receive full highlighting (bright blue). All other values in the [0 - 1] range correspond to lighter and darker shades of blue.

The key to producing this gradient is to hold the Blue component of the color constant at 255 (producing a bright blue) and add increasing amounts of Red and Green to generate less intense shades of blue. I calculated the Red and Green component values according to the formula Red = Green = 255 * (1 - Normalized Highlighting Intensity Value).

I derived highlighting colors from intensity values for each word in the talk, according to the preceding method. Having established a highlighting strength for each word in the talk, the final step in creating visualization was to weave the text data and highlighting color data into the final visualization format.

Visualization: Superimposing Highlights onto Original HTML

The final step in creating the visualization was to reconstruct the talk in HTML format, based on the original HTML file and the derived highlight colors. The HTML structure primarily consists of <div> and <span> tags. The <div> tags structure the document into paragraphs (as in the original HTML file), and the <span> tags provide hooks for CSS code to add the highlighting to each word. A line from the final HTML file may look like Figure 7.

```
<div>
  <span style=”color: rgb(255, 255, 0)”>The</span> <span style=”color:
rgb(255, 255, 120)”>word</span> …
</div>
```

Figure 7. Example of Highlighted HTML

This modified HTML file constitutes the final visualization of the talk and can be viewed in any standards-compliant web browser. See Appendix B for an example of a fully highlighted conference talk.

Evaluation of the Final Visualization

In order to obtain a basic evaluation of the visualization output, several individuals were invited to participate in an online survey. The survey consisted of a single, highlighted conference talk, followed by a number of questions intended to evaluate the potential usefulness of the visualization. These questions included eight Likert-type questions and six free-response questions. Thirteen people completed the survey. Overall, they found the tool helpful, as evidenced by the survey results. In this section, I present each of the survey questions and briefly discuss participant data, followed by a summary of the resulting suggestions for improving the visualization.

Survey Questions

Question 1: Overall, I found the visualization (tweet-based highlighting of conference talk text) intuitive and easy to understand. It appears that the conventions used in the visualization were straightforward and easily understood. Four respondents (31%) Strongly Agreed with this statement and nine respondents (69%) Agreed. No respondents selected Neutral, Disagree, or Strongly Disagree for this item.

Question 2: The visualization helped me identify interesting excerpts from the talk that I might not have otherwise discovered. The results for this item suggest that the visualization is helpful for bringing potentially interesting quotes to the attention of the reader. Three respondents (23%) Strongly Agreed with this statement, six (46%) Agreed, and four respondents (31%) were Neutral. No respondents selected Disagree of Strongly Disagree for this item.

Question 3: Being able to access this visualization would improve my learning from General Conference. The results for this item were interesting because it showed a greater spread of opinions. The majority agreed with this statement, but almost one third of respondents were either neutral or in disagreement. One respondent (8%) Strongly Agreed with this statement and eight (62%) Agreed. Two respondents (15%) were Neutral. One respondent (8%) Disagreed and one (8%) Strongly Disagreed.

Question 4: If this were available to me, I would use this tool and visualization in studying conference talks. Again, the majority of respondents were in agreement with this statement. Two (15%) Strongly Agreed with this statement, and nine (69%) Agreed. One respondent (8%) was Neutral, and one (8%) Strongly Disagreed. No respondents marked Disagree for this item.

Question 5: If I were taught how (if necessary), I would participate in tweeting thoughts during Conference to create visualizations such as this. The results of this item were surprising. While the majority of respondents indicated in question 4 that they would use this tool, the majority of respondents also disinclined to participate in the creation of the visualizations. One respondent (8%) Strongly Agreed with this statement, and three (23%) Agreed. Five (38%) were Neutral, one (8%) Disagreed, and three respondents (23%) Strongly Disagreed.

Question 6: On the previous page, I read the entire talk, word for word. The majority of respondents skimmed the talk rather than read it thoroughly. One respondent (8%) Strongly Agreed, and three (23%) Agreed. Two respondents (15%) were Neutral, while five (38%) Disagreed, and two (15%) Strongly Disagreed.

Question 7: When reading in general, I prefer to skim the material rather than to read the piece in its entirety. This item revealed an interesting split among respondents. Five respondents (38%) Strongly Agreed, while four (31%) were Neutral and four (31%) Disagreed. No respondents selected Agree or Strongly Disagree.

Question 8: The highlighting in the conference talk encouraged me to skim more than I normally would. While question 7 revealed a distinction between those who prefer to skim read and those who prefer to read thoroughly, the majority of respondents affirmed that they felt the visualization encouraged them to skim through the talk more than they normally would. Four respondents (31%) Strongly Agreed, and six (46%) Agreed. Two respondents (15%) were Neutral, while one (8%) Disagreed. No respondents Strongly Disagreed with this statement.

Question 9: What questions came to mind (about the data, the visualization, or anything else) as you were using this visualization? The majority of responses to this question centered on the use of highlighting. Two respondents wondered how the relative highlighting magnitudes mapped to actual number of people who tweeted a particular quote. (This question was raised again by a respondent in question 11.) Two others indicated their dislike for blue as the highlighting color, citing difficulty in reading the dark blue segments. Other responses indicated uncertainty about why some seemingly unimportant parts of the talk were highlighted while other more important parts received no highlighting at all.

Question 10: Was there any way in which the visualization was not *helpful, or was confusing?* Three respondents indicated that in many instances, certain parts of the talk are highlighted in awkward ways. For example, one noted that sometimes "the edges of the highlighting were not semantically appropriate, as it goes beyond the edge of a thought or sentence and picks up the first word or two of the next." Two other respondents mentioned that small words and phrases, including names, received dark highlighting and noted that these highlighted segments seemed meaningless.

Question 11: How could the visualization of the tweet data be improved? The majority of respondents focused their responses on the use of color. Two respondents indicated that they would like to see multiple colors used, either to indicate the user that tweeted the quote or to distinguish between different ideas in the text. Two respondents indicated their dislike of the color blue, stating that it was difficult to read at times. Interestingly, one respondent supported the decision to highlight in blue, saying that "it is easy on the eyes."

Question 12: Would you personally use this visualization tool to study General Conference? All respondents indicated (with varying degrees of certainty) that they either would use the visualization for personal study of General Conference or that they could find other uses for it. For example, three respondents indicated that this tool would be very useful in preparing to teach others a lesson based on the talk. Of those who indicated they would use it for personal study, four respondents suggested using the visualization in conjunction with the unmarked version of the talk. They each stated that the visualization could be useful either as a pre-reading or a review tool in addition to a thorough reading of the non-highlighted talk.

Question 13: How could this visualization tool be useful in studying other topics? Do you have any examples? The clear benefit of this visualization as expressed by many of the respondents is its ability to help you skim through the most important parts of a large text. Seven out of thirteen respondents suggested that this method of collaborative highlighting could be applied to textbooks, essays, journal articles, or any other reading item that one might encounter in a university course. Three respondents suggested that it could be used in university classes, especially large lecture-hall type classes with large numbers of students. Two other respondents noted that this could be useful for any situation in which a transcript is produced and the speaker is looking for feedback. Such situations could include (as already stated) classroom lectures or even political speeches.

Question 14: Do you have any other comments in general? As a broad “catch-all” type of question, this item elicited a wide variety of responses, of which there were two particularly noteworthy observations from the respondents. One respondent noted that while this visualization makes clear what the majority of one’s peers thought was important, it might actually detract the learner from specific details that are important for improving his or her own understanding. Another respondent noted that this type of visualization (based on tweets) is probably better suited to the younger population. Older generations, who are unfamiliar with tweeting and social media in general, would likely be less inclined to participate in the production of such a visualization.

Summary of Survey Results

The complete set of survey responses provided me with insightful feedback that could help improve the next version of this visualization. Among these ideas, the following items stand out as major improvements:

- In order to help the reader understand how many people quoted a specific phrase, provide a key describing how highlighting intensity corresponds to frequency.
- In order to accommodate individuals with varying color preferences, allow users to choose the highlight color that works best for them.
- In order to decrease visual clutter, enhance the algorithm to avoid highlighting seemingly disconnected bits of information.

Discussion of the Final Visualization

In this study I set out to create a centralized visualization from the notes generated during a large-scale distributed collaborative note-taking exercise. I chose to base this project on data collected from Twitter during the October 2010 General Conference of the Church of Jesus Christ of Latter-day Saints (hashtag: *#ldsconf*). During the data collection period, over 2,500 users produced over 26,000 tweets, sharing their observations, comments, notes, and favorite quotes with everyone else monitoring the *#ldsconf* hashtag. These Twitter users were located all over the world, making this collaborative note-taking exercise a truly large-scale, distributed endeavor. I used the method described above to convert the tweets into 35 distinct visualizations, one for each conference talk.

Implications and Applications

These visualizations support collaborative note-taking efforts by ameliorating some of the effects of information overload. Collaborative note-taking among 2,500 collaborators would be unthinkable under most circumstances, and to consume 26,000 notes would typically exceed the time and attention constraints of most collaborators. However, these visualizations are theoretically unlimited in the number of collaborators and the number of notes they can support. Furthermore, an increase in the number of notes (i.e., tweets) actually increases the representativeness of such a visualization. It may be possible that an even more well-tweeted event, such as a presidential speech, might reach a point of diminishing returns wherein the sheer volume of tweets reduces the overall utility of this visualization approach, warranting further research on its boundaries and limitations.

These visualizations illuminate the most frequently used words and phrases in the *#ldsconf* tweets collected during the conference. In general, words and phrases that were most frequently quoted by Twitter users are shown with a more intense blue highlighting. Other text visualizations, such as Wordle, also show the relative frequency of words in a given body of text. However, the visualization algorithm described in this study has an advantage over other text visualizations in that it preserves the original context of the quoted words and phrases as a unifying framework with which to condense the information represented in the individual notes. This context also clearly aids the eventual consumer of the notes in obtaining a more complete understanding of the notes.

Limitations of the Current Algorithm

While the visualization algorithm generally appears to perform as expected, it suffers from two limitations in its present form. First, because the structure of the visualization is defined by the text of the talk from which people were quoting, the algorithm makes no allowance for text that is not contained within the talk. In other words, if all 2,500 Twitter users made an identical tweet, but the text of that tweet (or sub-components thereof) was not found in the text of a talk, the final visualization would not give any indication that the expression in question was tweeted 2,500 times. In this way, it is possible for the visualizations to omit certain words or expressions that many people tweeted.

The second major limitation stands opposite to the first. Because the weighting of each phrase is based on the length of the phrase, it is possible for one individual to tweet a verbatim quote of a lengthy passage from the text of the talk and skew the highlighting scale. If the quote were long enough, its weight would trump the weights of almost all other quotes, resulting in a very dark highlighting for this one quote (tweeted by only one individual) and very weak highlighting for other quotes which, although shorter in length, were quoted by many more individuals.

Suggested Improvements

Future versions of the visualization algorithm need to address the issues outlined in the "Limitations" section. Additionally, the algorithm and visualizations could be improved in several other ways. First, the algorithm could be optimized for speed. Currently, the algorithm searches for all n-gram phrase matches. However, if a 5-gram match is found, then its constituent 4-, 3-, 2-, and 1- grams logically must also be found. If the algorithm searched for matches, starting with highest order n-grams first, we could eliminate the need to search for many of the smaller n-gram phrases, thus significantly improving the speed and efficiency of the algorithm.

A second improvement that would greatly improve the usability of the visualization would be to enable dynamic highlighting scale modification by means of a user interface component such as a slider widget. This slider would give the user the ability to choose the desired amount of highlighting, thus eliminating any excess highlighting due to frequently occurring, though less important, words and phrases.

Suggestions for Future Research

There are many ways in which the research undertaken in this project can be expanded and improved. One major limitation of the algorithm, described above, is the inability to visualize words and phrases that were used by many of the collaborators but not found in the text of the source document. One way in which this limitation might be overcome is to combine this visualization method with another visualization method, such as IBM's WordTree visualization. Such a setup would allow the user to both visualize tweets quoting from the source text and catch frequently used words and phrases that were not used in the source text.

While the automated approach associates tweets with phrases within conference addresses, there is currently no way to gain access to the tweets used by the system to highlight those phrases. In another version of the system, it should be possible to click on a highlighted word or phrase and see the tweets that the system associated with that word or phrase. In addition to providing a quick way to check the accuracy of the system, this would also provide individuals who want to dive deeper into a certain phrase with access to the raw data related to the phrase. This will further expand the context-preserving nature of the system.

The approach could be expanded and used together with tools that track public sentiment or opinion. Combining these approaches to analyze political speeches would be particularly interesting. Political speeches commonly have transcripts published and commonly generate a large number of tweets. The capability to quickly and easily visualize public response to political addresses could be a key piece of infrastructure for reinvigorating democracy in an information age.

This project centered on the collaborative note-taking activities of participants in the LDS General Conference. However, this method could be applied to any situation in which a transcript for the event is eventually published. For example, in a classroom lecture, the teacher could record the lecture using free voice-recording software and a cheap microphone. That audio recording could be fed through inexpensive automatic transcription software, such as Dragon Naturally Speaking, and turned into a transcript of the talk. If the students in the class recorded their notes digitally in small pieces (similar to Twitter), then these notes and the transcript of the lecture could be fed into the visualization algorithm, producing a visualization of class notes for every class period.

While all of the previous examples require the existence of a transcript, it may also be possible to apply this algorithm to situations in which no transcript exists. However, it would still be necessary to provide some body of text to be used for the structure of the visualization. For example, one study might involve collecting tweets from the Twitter Stream regarding a topic like *#economics*. Then, all articles on economics from Wikipedia might be downloaded to be used as the structure text. By applying the algorithm to this data and selecting those snippets from the Wikipedia article text, one may still be able to produce a contextual visualization that shows what people are saying about that specific topic.

Conclusion

Each day, an incredible amount of information is produced in the form of tweets, blog posts, forum discussions, emails, and a host of other formats. Much of this information is produced by people with specific interests, with the intent of sharing information with others.

This massively distributed form of collaborative note-taking represents an untapped reservoir of valuable information. Hopefully the technique described in this project, together with others it may inspire, will help individuals around the world derive greater value from the ever-growing amount of information available online.

References

Apache Foundation. (2012). Apache Lucene. Retrieved from http://lucene.apache.org/java/docs/index.html

Berque, D., & Bonebright, T. (2004). Using pen-based computers across the computer science curriculum. *SIGCSE ’04 Proceedings of the 35th SIGCSE Technical Symposium on Computer Science Education* (pp. 61 - 65). New York, NY: ACM.

Blair, A. (2008). Student manuscripts and the textbook. In E. Campi, S. De Angelis, A. Goeing, & A. Grafton (Eds.), *Scholarly knowledge: Textbooks in early modern Europe* (pp. 39-73). Geneva: Droz.

Chiu, C. H., Chen, C. H., Wu, C. Y., & Chen, S. W. (2010). Elementary school students’ attitudes toward applying wikis or blogs for collaborative note-taking activities. In Z. W. Abas, I. Jung, & J. Luca (Eds.), *Proceedings of Global Learn Asia Pacific 2010* (pp. 298-302). Chesapeake, VA: AACE.

Crockford, D. (2012). JSON. Retrieved from http://json.org/

Defoe, D. (1919). *The life and adventures of Robinson Crusoe*. London: Seeley, Service & Co.

Denoue, L., Singh, G., & Das, A. (2004). Taking notes on PDAs with shared text input. In L. Cantoni & C. McLoughlin (Eds.), *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications 2004* (pp. 2553-2560). Chesapeake, VA: AACE.

Feinberg, J. (2011). Wordle - beautiful word clouds. Retrieved January 10, 2012, from http://www.wordle.net

Hearst, M. A., & Rosner, D. (2008). Tag clouds: Data analysis tool or social signaller? *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)* (p. 160). IEEE.

Hunston, S. (2002). *Corpora in applied linguistics*. (M. H. Long & J. C. Richards, Eds.). Cambridge: Cambridge University Press.

International Business Machines. (2012a). Many eyes: Word tree. Retrieved from http://www-958.ibm.com/software/data/cognos/manyeyes/page/Word_Tree.html

International Business Machines. (2012b). Many eyes: Phrase net. Retrieved from http://www-958.ibm.com/software/data/cognos/manyeyes/page/Phrase_Net.html

Kosara, R. (2010). The year of infovis theory. Retrieved from http://eagereyes.org/blog/2010/the-year-of-infovis-theory

Manning, C. D., & Schütze, H. (1999). *Foundations of statistical natural language processing*. Cambridge, MA: The MIT Press.

Oracle. (2012). MySQL : The world's most popular open source database. Retrieved from http://www.mysql.com/

Python. (2012a). Python programming language – Official website. Retrieved from http://python.org/

Python. (2012b). tweetstream 0.3.1 : Python package index. Retrieved from http://pypi.python.org/pypi/tweetstream/0.3.1

Singh, G., Denoue, L., & Das, A. (2005). Collaborative note taking using PDAs. *Journal of Information Science and Engineering*, *21*, 835-848.

The Church of Jesus Christ of Latter-day Saints. (2010a). April 2010 General Conference. Retrieved from http://lds.org/general-conference/sessions/2010/04?lang=eng

The Church of Jesus Christ of Latter-day Saints. (2010b). October 2010 General Conference. Retrieved from http://lds.org/general-conference/sessions/2010/10?lang=eng

Twitter. (2012a). Twitter developers. Retrieved from https://dev.twitter.com/

Twitter. (2012b). Streaming API - Twitter developers. Retrieved from https://dev.twitter.com/docs/streaming-api

Ware, C. (2004). *Information visualization: Perception for design* (2nd ed.). San Francisco: Morgan Kaufmann.

Wickham, H. (2009). *ggplot2: Elegant graphics for data analysis*. (R. Gentleman, K. Hornik, & G. Parmigiani, Eds.). New York: Springer.

Wilkinson, L. (2005). *The grammar of graphics*. (J. Chambers, D. Hand, & W. Haerdle, Eds.) (2nd ed.). London: Springer.

Appendix A

Example of Plain Text Conference Talk Prepared for Indexing

Pride and the Priesthood

President Dieter F. Uchtdorf Second Counselor in the First Presidency

Pride is a switch that turns off priesthood power. Humility is a switch that turns it on.

My dear brothers, thank you for assembling all around the world for this priesthood session of general conference. Your presence shows your commitment to stand, wherever you are, with your brothers who bear the holy priesthood and serve and honor your Lord and Redeemer, Jesus Christ.

Often we mark the span of our lives by events that leave imprints on our minds and hearts. There are many such events in my life, one of which happened in 1989 when I heard a timeless sermon by President Ezra Taft Benson, "Beware of Pride." In the introduction it was noted that this topic had been weighing heavily on President Benson's soul for some time.

I have felt a similar burden during the past months. The promptings of the Holy Spirit have urged me to add my voice as another witness to President Benson's message delivered 21 years ago.

Every mortal has at least a casual if not intimate relationship with the sin of pride. No one has avoided it; few overcome it. When I told my wife that this would be the topic of my talk, she smiled and said, "It is so good that you talk about things you know so much about."

Other Meanings of Pride

I also remember one interesting side effect of President Benson's influential talk. For a while it almost became taboo among Church members to say that they were "proud" of their children or their country or that they took "pride" in their work. The very word pride seemed to become an outcast in our vocabulary.

In the scriptures we find plenty of examples of good and righteous people who rejoice in righteousness and at the same time glory in the goodness of God. Our Heavenly Father Himself introduced His Beloved Son with the words "in whom I am well pleased."

Alma gloried in the thought that he might "be an instrument in the hands of God." The Apostle Paul gloried in the faithfulness of members of the Church. The great missionary Ammon gloried in the success he and his brothers had experienced as missionaries.

I believe there is a difference between being proud of certain things and being prideful. I am proud of many things. I am proud of my wife. I am proud of our children and grandchildren.

I am proud of the youth of the Church, and I rejoice in their goodness. I am proud of you, my dear and faithful brethren. I am proud to stand shoulder to shoulder with you as a bearer of the holy priesthood of God.

Pride Is the Sin of Self-Elevation

So what is the difference between this kind of feeling and the pride that President Benson called "the universal sin"? Pride is sinful, as President Benson so memorably taught, because it breeds hatred or hostility and places us in opposition to God and our fellowmen. At its core, pride is a sin of comparison, for though it usually begins with "Look how wonderful I am and what great things I have done," it always seems to end with "Therefore, I am better than you."

When our hearts are filled with pride, we commit a grave sin, for we violate the two great commandments. Instead of worshipping God and loving our neighbor, we reveal the real object of our worship and love--the image we see in the mirror.

Pride is the great sin of self-elevation. It is for so many a personal Rameumptom, a holy stand that justifies envy, greed, and vanity. In a sense, pride is the original sin, for before the foundations of this earth, pride felled Lucifer, a son of the morning "who was in authority in the presence of God." If pride can corrupt one as capable and promising as this, should we not examine our own souls as well?

Pride Has Many Faces

Pride is a deadly cancer. It is a gateway sin that leads to a host of other human weaknesses. In fact, it could be said that every other sin is, in essence, a manifestation of pride.

This sin has many faces. It leads some to revel in their own perceived self-worth, accomplishments, talents, wealth, or position. They count these blessings as evidence of being "chosen," "superior," or "more righteous" than others. This is the sin of "Thank God I am more special than you." At its core is the desire to be admired or envied. It is the sin of self-glorification.

For others, pride turns to envy: they look bitterly at those who have better positions, more talents, or greater possessions than they do. They seek to hurt, diminish, and tear down others in a misguided and unworthy attempt at self-elevation. When those they envy stumble or suffer, they secretly cheer.

The Laboratory of Sports

Perhaps there is no better laboratory to observe the sin of pride than the world of sports. I have always loved participating in and attending sporting events. But I confess there are times when the lack of civility in sports is embarrassing. How is it that normally kind and compassionate human beings can be so intolerant and filled with hatred toward an opposing team and its fans?

I have watched sports fans vilify and demonize their rivals. They look for any flaw and magnify it. They justify their hatred with broad generalizations and apply them to everyone associated with the other team. When ill fortune afflicts their rival, they rejoice.

Brethren, unfortunately we see today too often the same kind of attitude and behavior spill over into the public discourse of politics, ethnicity, and religion.

My dear brethren of the priesthood, my beloved fellow disciples of the gentle Christ, should we not hold ourselves to a higher standard? As priesthood bearers, we must realize that all of God's children wear the same jersey. Our team is the brotherhood of man. This mortal life is our playing field. Our goal is to learn to love God and to extend that same love toward our fellowman. We are here to live according to His law and establish the kingdom of God. We are here to build, uplift, treat fairly, and encourage all of Heavenly Father's children.

We Must Not Inhale

When I was called as a General Authority, I was blessed to be tutored by many of the senior Brethren in the Church. One day I had the opportunity to drive President James E. Faust to a stake conference. During the hours we spent in the car, President Faust took the time to teach me some important principles about my assignment. He explained also how gracious the members of the Church are, especially to General Authorities. He said, "They will treat you very kindly. They will say nice things about you." He laughed a little and then said, "Dieter, be thankful for this. But don't you ever inhale it."

That is a good lesson for us all, brethren, in any calling or life situation. We can be grateful for our health, wealth, possessions, or positions, but when we begin to inhale it--when we become obsessed with our status; when we focus on our own importance, power, or reputation; when we dwell upon our public image and believe our own press clippings--that's when the trouble begins; that's when pride begins to corrupt.

There are plenty of warnings about pride in the scriptures: "Only by pride cometh contention: but with the well advised is wisdom."

The Apostle Peter warned that "God resisteth the proud, and giveth grace to the humble." Mormon explained, "None is acceptable before God, save the meek and lowly in heart." And by design, the Lord chooses "the weak things of the world to confound the things which are mighty." The Lord does this to show that His hand is in His work, lest we "trust in the arm of flesh."

We are servants of our Lord and Savior, Jesus Christ. We are not given the priesthood so that we can take our bows and bask in praise. We are here to roll up our sleeves and go to work. We are enlisted in no ordinary task. We are called to prepare the world for the coming of our Lord and Savior, Jesus Christ. We seek not our own honor but give praise and glory to God. We know that the contribution we can make by ourselves is small; nevertheless, as we exercise the power of the priesthood in righteousness, God can cause a great and marvelous work to come forth through our efforts. We must learn, as Moses did, that "man is nothing" by himself but that "with God all things are possible."

Jesus Christ Is the Perfect Example of Humility

In this, as in all things, Jesus Christ is our perfect example. Whereas Lucifer tried to change the Father's plan of salvation and obtain honor for himself, the Savior said, "Father, thy will be done, and the glory be thine forever." Despite His magnificent abilities and accomplishments, the Savior was always meek and humble.

Brethren, we hold "the Holy Priesthood, after the Order of the Son of God." It is the power God has granted to men on earth to act for Him. In order to exercise His power, we must strive to be like the Savior. This means that in all things we seek to do the will of the Father, just as the Savior did. It means that we give all glory to the Father, just as the Savior did. It means that we lose ourselves in the service of others, just as the Savior did.

Pride is a switch that turns off priesthood power. Humility is a switch that turns it on.

Be Humble and Full of Love

So how do we conquer this sin of pride that is so prevalent and so damaging? How do we become more humble?

It is almost impossible to be lifted up in pride when our hearts are filled with charity. "No one can assist in this work except he shall be humble and full of love." When we see the world around us through the lens of the pure love of Christ, we begin to understand humility.

Some suppose that humility is about beating ourselves up. Humility does not mean convincing ourselves that we are worthless, meaningless, or of little value. Nor does it mean denying or withholding the talents God has given us. We don't discover humility by thinking less of ourselves; we discover humility by thinking less about ourselves. It comes as we go about our work with an attitude of serving God and our fellowman.

Humility directs our attention and love toward others and to Heavenly Father's purposes. Pride does the opposite. Pride draws its energy and strength from the deep wells of selfishness. The moment we stop obsessing with ourselves and lose ourselves in service, our pride diminishes and begins to die.

My dear brethren, there are so many people in need whom we could be thinking about instead of ourselves. And please don't ever forget your own family, your own wife. There are so many ways we could be serving. We have no time to become absorbed in ourselves.

I once owned a pen that I loved to use during my career as an airline captain. By simply turning the shaft, I could choose one of four colors. The pen did not complain when I wanted to use red ink instead of blue. It did not say to me, "I would rather not write after 10:00 p.m., in heavy fog, or at high altitudes." The pen did not say, "Use me only for important documents, not for the daily mundane tasks." With greatest reliability it performed every task I needed, no matter how important or insignificant. It was always ready to serve.

In a similar way we are tools in the hands of God. When our heart is in the right place, we do not complain that our assigned task is unworthy of our abilities. We gladly serve wherever we are asked. When we do this, the Lord can use us in ways beyond our understanding to accomplish His work.

Let me conclude with words from President Ezra Taft Benson's inspired message of 21 years ago:

"Pride is the great stumbling block to Zion.

"We must cleanse the inner vessel by conquering pride. . . .

"We must yield 'to the enticings of the Holy Spirit,' put off the prideful 'natural man,' become 'a saint through the atonement of Christ the Lord,' and become 'as a child, submissive, meek, humble.' . . .

"God will have a humble people. . . . 'Blessed are they who humble themselves without being compelled to be humble.' . . .

"Let us choose to be humble. We can do it. I know we can."

My beloved brethren, let us follow the example of our Savior and reach out to serve rather than seeking the praise and honor of men. It is my prayer that we will recognize and root out unrighteous pride in our hearts and that we will replace it with "righteousness, godliness, faith, love, patience, [and] meekness." In the sacred name of Jesus Christ, amen.

Appendix B

Example of Highlighted Conference Talk

Pride and the Priesthood

President Dieter F. Uchtdorf Second Counselor in the First Presidency

Pride is a switch that turns off priesthood power. Humility is a switch that turns it on.

My dear brothers, thank you for assembling all around the world for this priesthood session of general conference. Your presence shows your commitment to stand, wherever you are, with your brothers who bear the holy priesthood and serve and honor your Lord and Redeemer, Jesus Christ.

Often we mark the span of our lives by events that leave imprints on our minds and hearts. There are many such events in my life, one of which happened in 1989 when I heard a timeless sermon by President Ezra Taft Benson, "Beware of Pride." In the introduction it was noted that this topic had been weighing heavily on President Benson's soul for some time.

I have felt a similar burden during the past months. The promptings of the Holy Spirit have urged me to add my voice as another witness to President Benson's message delivered 21 years ago.

Every mortal has at least a casual if not intimate relationship with the sin of pride. No one has avoided it; few overcome it. When I told my wife that this would be the topic of my talk, she smiled and said, "It is so good that you talk about things you know so much about."

Other Meanings of Pride

I also remember one interesting side effect of President Benson's influential talk. For a while it almost became taboo among Church members to say that they were "proud" of their children or their country or that they took "pride" in their work. The very word pride seemed to become an outcast in our vocabulary.

In the scriptures we find plenty of examples of good and righteous people who rejoice in righteousness and at the same time glory in the goodness of God. Our Heavenly Father Himself introduced His Beloved Son with the words "in whom I am well pleased."

Alma gloried in the thought that he might "be an instrument in the hands of God." The Apostle Paul gloried in the faithfulness of members of the Church. The great missionary Ammon gloried in the success he and his brothers had experienced as missionaries.

I believe there is a difference between being proud of certain things and being prideful. I am proud of many things. I am proud of my wife. I am proud of our children and grandchildren.

I am proud of the youth of the Church, and I rejoice in their goodness. I am proud of you, my dear and faithful brethren. I am proud to stand shoulder to shoulder with you as a bearer of the holy priesthood of God.

Pride Is the Sin of Self-Elevation

So what is the difference between this kind of feeling and the pride that President Benson called "the universal sin"? Pride is sinful, as President Benson so memorably taught, because it breeds hatred or hostility and places us in opposition to God and our fellowmen. At its core, pride is a sin of comparison, for though it usually begins with "Look how wonderful I am and what great things I have done," it always seems to end with "Therefore, I am better than you."

When our hearts are filled with pride, we commit a grave sin, for we violate the two great commandments. Instead of worshipping God and loving our neighbor, we reveal the real object of our worship and love--the image we see in the mirror.

Pride is the great sin of self-elevation. It is for so many a personal Rameumptom, a holy stand that justifies envy, greed, and vanity. In a sense, pride is the original sin, for before the foundations of this earth, pride felled Lucifer, a son of the morning "who was in authority in the presence of God." If pride can corrupt one as capable and promising as this, should we not examine our own souls as well?

Pride Has Many Faces

Pride is a deadly cancer. It is a gateway sin that leads to a host of other human weaknesses. In fact, it could be said that every other sin is, in essence, a manifestation of pride.

This sin has many faces. It leads some to revel in their own perceived self-worth, accomplishments, talents, wealth, or position. They count these blessings as evidence of being "chosen," "superior," or "more righteous" than others. This is the sin of "Thank God I am more special than you." At its core is the desire to be admired or envied. It is the sin of self-glorification.

For others, pride turns to envy: they look bitterly at those who have better positions, more talents, or greater possessions than they do. They seek to hurt, diminish, and tear down others in a misguided and unworthy attempt at self-elevation. When those they envy stumble or suffer, they secretly cheer.

The Laboratory of Sports

Perhaps there is no better laboratory to observe the sin of pride than the world of sports. I have always loved participating in and attending sporting events. But I confess there are times when the lack of civility in sports is embarrassing. How is it that normally kind and compassionate human beings can be so intolerant and filled with hatred toward an opposing team and its fans?

I have watched sports fans vilify and demonize their rivals. They look for any flaw and magnify it. They justify their hatred with broad generalizations and apply them to everyone associated with the other team. When ill fortune afflicts their rival, they rejoice.

Brethren, unfortunately we see today too often the same kind of attitude and behavior spill over into the public discourse of politics, ethnicity, and religion.

My dear brethren of the priesthood, my beloved fellow disciples of the gentle Christ, should we not hold ourselves to a higher standard? As priesthood bearers, we must realize that all of God's children wear the same jersey. Our team is the brotherhood of man. This mortal life is our playing field. Our goal is to learn to love God and to extend that same love toward our fellowman. We are here to live according to His law and establish the kingdom of God. We are here to build, uplift, treat fairly, and encourage all of Heavenly Father's children.

We Must Not Inhale

When I was called as a General Authority, I was blessed to be tutored by many of the senior Brethren in the Church. One day I had the opportunity to drive President James E. Faust to a stake conference. During the hours we spent in the car, President Faust took the time to teach me some important principles about my assignment. He explained also how gracious the members of the Church are, especially to General Authorities. He said, "They will treat you very kindly. They will say nice things about you." He laughed a little and then said, "Dieter, be thankful for this. But don't you ever inhale it."

That is a good lesson for us all, brethren, in any calling or life situation. We can be grateful for our health, wealth, possessions, or positions, but when we begin to inhale it--when we become obsessed with our status; when we focus on our own importance, power, or reputation; when we dwell upon our public image and believe our own press clippings--that's when the trouble begins; that's when pride begins to corrupt.

There are plenty of warnings about pride in the scriptures: "Only by pride cometh contention: but with the well advised is wisdom."

The Apostle Peter warned that "God resisteth the proud, and giveth grace to the humble." Mormon explained, "None is acceptable before God, save the meek and lowly in heart." And by design, the Lord chooses "the weak things of the world to confound the things which are mighty." The Lord does this to show that His hand is in His work, lest we "trust in the arm of flesh."

We are servants of our Lord and Savior, Jesus Christ. We are not given the priesthood so that we can take our bows and bask in praise. We are here to roll up our sleeves and go to work. We are enlisted in no ordinary task. We are called to prepare the world for the coming of our Lord and Savior, Jesus Christ. We seek not our own honor but give praise and glory to God. We know that the contribution we can make by ourselves is small; nevertheless, as we exercise the power of the priesthood in righteousness, God can cause a great and marvelous work to come forth through our efforts. We must learn, as Moses did, that "man is nothing" by himself but that "with God all things are possible."

Jesus Christ Is the Perfect Example of Humility

In this, as in all things, Jesus Christ is our perfect example. Whereas Lucifer tried to change the Father's plan of salvation and obtain honor for himself, the Savior said, "Father, thy will be done, and the glory be thine forever." Despite His magnificent abilities and accomplishments, the Savior was always meek and humble.

Brethren, we hold "the Holy Priesthood, after the Order of the Son of God." It is the power God has granted to men on earth to act for Him. In order to exercise His power, we must strive to be like the Savior. This means that in all things we seek to do the will of the Father, just as the Savior did. It means that we give all glory to the Father, just as the Savior did. It means that we lose ourselves in the service of others, just as the Savior did.

Pride is a switch that turns off priesthood power. Humility is a switch that turns it on.

Be Humble and Full of Love

So how do we conquer this sin of pride that is so prevalent and so damaging? How do we become more humble?

It is almost impossible to be lifted up in pride when our hearts are filled with charity. "No one can assist in this work except he shall be humble and full of love." When we see the world around us through the lens of the pure love of Christ, we begin to understand humility.

Some suppose that humility is about beating ourselves up. Humility does not mean convincing ourselves that we are worthless, meaningless, or of little value. Nor does it mean denying or withholding the talents God has given us. We don't discover humility by thinking less of ourselves; we discover humility by thinking less about ourselves. It comes as we go about our work with an attitude of serving God and our fellowman.

Humility directs our attention and love toward others and to Heavenly Father's purposes. Pride does the opposite. Pride draws its energy and strength from the deep wells of selfishness. The moment we stop obsessing with ourselves and lose ourselves in service, our pride diminishes and begins to die.

My dear brethren, there are so many people in need whom we could be thinking about instead of ourselves. And please don't ever forget your own family, your own wife. There are so many ways we could be serving. We have no time to become absorbed in ourselves.

I once owned a pen that I loved to use during my career as an airline captain. By simply turning the shaft, I could choose one of four colors. The pen did not complain when I wanted to use red ink instead of blue. It did not say to me, "I would rather not write after 10:00 p.m., in heavy fog, or at high altitudes." The pen did not say, "Use me only for important documents, not for the daily mundane tasks." With greatest reliability it performed every task I needed, no matter how important or insignificant. It was always ready to serve.

In a similar way we are tools in the hands of God. When our heart is in the right place, we do not complain that our assigned task is unworthy of our abilities. We gladly serve wherever we are asked. When we do this, the Lord can use us in ways beyond our understanding to accomplish His work.

Let me conclude with words from President Ezra Taft Benson's inspired message of 21 years ago:

"Pride is the great stumbling block to Zion.

"We must cleanse the inner vessel by conquering pride. . . .

"We must yield 'to the enticings of the Holy Spirit,' put off the prideful 'natural man,' become 'a saint through the atonement of Christ the Lord,' and become 'as a child, submissive, meek, humble.' . . .

"God will have a humble people. . . . 'Blessed are they who humble themselves without being compelled to be humble.' . . .

"Let us choose to be humble. We can do it. I know we can."

My beloved brethren, let us follow the example of our Savior and reach out to serve rather than seeking the praise and honor of men. It is my prayer that we will recognize and root out unrighteous pride in our hearts and that we will replace it with "righteousness, godliness, faith, love, patience, [and] meekness." In the sacred name of Jesus Christ, amen.

Appendix C

Source Code for the Visualization Algorithm

The following source code can be downloaded from:

https://github.com/ajmagnifico/Arpodwot-Twheatmap

org/arpodwot/heatmaps/documents/input/DocumentListFile.java 1/35

```
package org.arpodwot.heatmaps.documents.input;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.LinkedList;

public class DocumentListFile {
    private String _filePath;
    private int _fileCount;
    private int _currentPosition;

    private LinkedList<String> _list;

    public DocumentListFile(String filePath) throws IOException {
        _currentPosition = 0;
        _list = new LinkedList<String>();
        _filePath = filePath;

        // read in the notes from the file
        FileReader fr = new FileReader(_filePath);
        BufferedReader br = new BufferedReader(fr);

        String file;
        while((file = br.readLine()) != null){
            _list.add(file);
        }
        br.close();
        fr.close();

        _fileCount = _list.size();
    }

    public String[] getFiles() {
        return (String[])_list.toArray();
    }

    public void reset() {
        _currentPosition = 0;
    }

    public boolean hasNext() {
        if (_currentPosition >= _list.size())
            return false;

        return true;
    }

    public String nextFile() {
        if (hasNext()){
            String file = _list.get(_currentPosition);
            _currentPosition++;
            return file;
        }

        return null;
    }

    public int size(){
        return _fileCount;
    }
}
```

org/arpodwot/heatmaps/documents/input/InputDocumentCollection.java 2/35

```
package org.arpodwot.heatmaps.documents.input;

import java.io.IOException;
import java.util.ArrayList;


public class InputDocumentCollection extends ArrayList<InputDocument> {
    private static final long serialVersionUID = 1L;

    public InputDocumentCollection(String filePath) throws IOException {
        DocumentListFile docList = new DocumentListFile(filePath);
        int nextId = 0;
        while (docList.hasNext()){
            InputDocument d = new SimpleTextDocument(docList.nextFile());
            d.setId(nextId);
            this.add(d);
            nextId++;
        }
    }
}
```

org/arpodwot/heatmaps/documents/input/InputDocument.java 3/35

```
package org.arpodwot.heatmaps.documents.input;
```

```java
import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.index.TermPositionVector;

public interface InputDocument {
    public int getId();
    public void setId(int id);

    public String getFilePath();
    public void setFilePath(String filePath);

    public String getDirPath();
    public void setDirPath(String dirPath);

    public String getFileName();
    public void setFileName(String fileName);

    public String getText();
    public void setText(String text);

    public TermPositionVector getTermPositionVector();
    public void setTermPositionVector(TermPositionVector tpv);

    public double[] getRawHighlightData();
    public void setRawHighlightData(double[] highlightData);

    public Analyzer getDocumentClassAnalyzer();
}
```

org/arpodwot/heatmaps/documents/input/SimpleTextDocument.java 4/35

```java
package org.arpodwot.heatmaps.documents.input;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.IOException;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.SimpleAnalyzer;
import org.apache.lucene.index.TermPositionVector;

public class SimpleTextDocument implements InputDocument {
    public static Analyzer DEFAULT_ANALYZER = new SimpleAnalyzer();

    @Override
    public Analyzer getDocumentClassAnalyzer(){
        return DEFAULT_ANALYZER;
    }

    private int _id;
    private String _filePath;
    private String _dirPath;
    private String _fileName;
    private String _text;

    private TermPositionVector _termPositionVector;
    private double[] _rawHighlightData;

    public SimpleTextDocument(){
        // return a blank document
    }

    public SimpleTextDocument(String filePath) throws IOException {
        _filePath = filePath;
        File f = new File(_filePath);
        _dirPath = f.getParent();
        _fileName = f.getName();

        FileReader fr = new FileReader(_filePath);
        BufferedReader br = new BufferedReader(fr);

        StringBuilder builder = new StringBuilder();
        String line;
        while ((line = br.readLine()) != null){
            if (line.trim().length() == 0) continue;
            builder.append(line+"\n");
        }

        br.close();
        fr.close();

        _text = builder.toString();
    }

    @Override
    public String getText() {
        return _text;
    }
    @Override
    public void setText(String text){
        _text = text;
    }

    @Override
    public int getId() {
        return _id;
    }
    @Override
    public void setId(int id) {
        _id = id;
    }

    @Override
    public String getFilePath(){
        return _filePath;
    }
    @Override
    public void setFilePath(String filePath){
        _filePath = filePath;
    }

    @Override
    public String getDirPath(){
        return _dirPath;
    }
    @Override
    public void setDirPath(String dirPath){
        _dirPath = dirPath;
    }

    @Override
    public String getFileName(){
        return _fileName;
    }
    @Override
    public void setFileName(String fileName){
        _fileName = fileName;
    }

    @Override
    public TermPositionVector getTermPositionVector(){
        return _termPositionVector;
    }
    @Override
    public void setTermPositionVector(TermPositionVector tpv){
        _termPositionVector = tpv;
    }

    @Override
    public double[] getRawHighlightData(){
        return _rawHighlightData;
    }
    @Override
    public void setRawHighlightData(double[] highlightData){
        _rawHighlightData = highlightData;
    }
}
```

org/arpodwot/heatmaps/documents/output/HighlightedHTMLDocumentGenerator.java

5/35

```
package org.arpodwot.heatmaps.documents.output;

import java.io.FileWriter;
import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.arpodwot.heatmaps.highlighting.color.BlueHighlighter;
import org.arpodwot.heatmaps.highlighting.color.HighlightColorCSSGenerator;
import org.arpodwot.heatmaps.highlighting.color.YellowHighlighter;


public class HighlightedHTMLDocumentGenerator implements
OutputDocumentGenerator {

    @Override
    public void writeToFile (
            double[] highlightData,
            String originalText,
            String filePath)
                throws Exception
    {
        String htmlFilePath = filePath+".html";
        Pattern linePattern = Pattern.compile("^.+$", Pattern.MULTILINE);
        Matcher lineMatcher = linePattern.matcher(originalText);
        Pattern tokenPattern = Pattern.compile("[a-z]+",
Pattern.CASE_INSENSITIVE);
        Matcher tokenMatcher = tokenPattern.matcher(originalText);

        HighlightColorCSSGenerator cssGen = new BlueHighlighter(0.1, 0.9);
        StringBuilder outputText = new StringBuilder();
        int currentToken = 0;

        // take it one line at a time
        int start = 0;
        while (lineMatcher.find(start)){
            int lineStart = lineMatcher.start();
            int lineEnd = lineMatcher.end();

            // find all tokens in this line
            ArrayList<Integer> tokenStarts = new ArrayList<Integer>();
            int tmpStart = lineStart;
            while (tmpStart < lineEnd){
                if (tokenMatcher.find(tmpStart) && tokenMatcher.start() <
lineEnd){
                    tokenStarts.add(tokenMatcher.start());
                    tmpStart = tokenMatcher.end();
                } else {
                    break;
                }
            }

            // add this line to the output
            outputText.append("<div class=\"paragraph\">");
            for (int i = 0; i < tokenStarts.size(); i++){
                int subSeqStart;
                int subSeqEnd;

                if (i == 0)
                    subSeqStart = lineStart;
                else
                    subSeqStart = tokenStarts.get(i);
                if (i == tokenStarts.size() - 1)
                    subSeqEnd = lineEnd;
                else
                    subSeqEnd = tokenStarts.get(i+1);

                // get the highlighting color
                String highlightCSS =
cssGen.generateHighlightCSS(highlightData[currentToken]);

                outputText.append("<span style=\""+highlightCSS+"\">");
                outputText.append(originalText.substring(subSeqStart,
subSeqEnd));
                outputText.append("</span>");
                currentToken++;
            }
            outputText.append("</div>\n");

            // move to the next line
            start = lineMatcher.end();
        }

        // write out the file
        FileWriter fw = new FileWriter(htmlFilePath);
        fw.write(outputText.toString());
        fw.close();
    }
}
```

org/arpodwot/heatmaps/documents/output/OutputDocumentGenerator.java 6/35

```
package org.arpodwot.heatmaps.documents.output;

public interface OutputDocumentGenerator {
    public void writeToFile(double[] highlightData, String originalText,
String filePath) throws Exception;
}
```

org/arpodwot/heatmaps/documents/SearchableDocumentCollection.java 7/35

```
package org.arpodwot.heatmaps.documents;

import java.io.IOException;

import org.apache.lucene.queryParser.ParseException;
import org.arpodwot.heatmaps.documents.input.InputDocument;

public interface SearchableDocumentCollection {
    public int[] searchDocuments(String queryText) throws
        IOException,
        ParseException;
    public InputDocument getDocumentById(int id) throws IOException;
}
```

org/arpodwot/heatmaps/highlighting/color/BlueHighlighter.java 8/35

```
package org.arpodwot.heatmaps.highlighting.color;

public class BlueHighlighter implements HighlightColorCSSGenerator {
    private double _minPercent;
    private double _maxPercent;

    public BlueHighlighter(){
        _minPercent = 0.0;
        _maxPercent = 1.0;
    }

    public BlueHighlighter(double minPercent, double maxPercent){
        _minPercent = minPercent;
        _maxPercent = maxPercent;
    }


    @Override
    public String generateHighlightCSS(double highlightValue) {
        double min255 = 255 * _minPercent;
        double max255 = 255 * _maxPercent;
        int blueValue = 255;

        if (highlightValue < 0.01)
            return "background-color:rgb(255,255,255);";

        int redGreenValue = (int)Math.floor(max255 - ((highlightValue *
(max255-min255))+min255));

        return "background-
color:rgb("+redGreenValue+","+redGreenValue+","+blueValue+");";
    }

}
```

org/arpodwot/heatmaps/highlighting/color/HighlightColorCSSGenerator.java 9/35

```
package org.arpodwot.heatmaps.highlighting.color;

public interface HighlightColorCSSGenerator {
    public String generateHighlightCSS(double highlightValue);
}
```

org/arpodwot/heatmaps/highlighting/color/YellowHighlighter.java 10/35

```
package org.arpodwot.heatmaps.highlighting.color;

public class YellowHighlighter implements HighlightColorCSSGenerator {
    private double _minPercent;

    public YellowHighlighter(){
        _minPercent = 0.0;
    }
    public YellowHighlighter(double minPercent){
        _minPercent = minPercent;
    }

    @Override
    public String generateHighlightCSS(double highlightValue) {
        double min255 = 255 * _minPercent;
        int redValue = 255;
        int greenValue = 255;

        if (highlightValue < 0.01)
            return "background-color:rgb(255,255,255);";

        int blueValue = (int)Math.floor(255 - ((highlightValue * (255-
min255))+min255));

        return "background-
color:rgb("+redValue+","+greenValue+","+blueValue+");";
    }
}
```

org/arpodwot/heatmaps/highlighting/DocumentHighlighter.java 11/35

```
package org.arpodwot.heatmaps.highlighting;

import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.notes.Note;

public interface DocumentHighlighter {
    public void highlightDocument(InputDocument doc, Note[] notes);
}
```

org/arpodwot/heatmaps/highlighting/HighlightedNote.java 12/35

```
package org.arpodwot.heatmaps.highlighting;

import java.util.ArrayList;

public class HighlightedNote extends ArrayList<HighlightedPhrase> {
    private static final long serialVersionUID = 1L;

    private double _minProbability = Double.MAX_VALUE;
    private double _maxProbability = Double.MIN_VALUE;
    private int _maxLength = Integer.MIN_VALUE;

    public HighlightedNote(int initialCapacity){
        super(initialCapacity);
    }

    @Override
    public boolean add(HighlightedPhrase hp){
        boolean result = super.add(hp);
        int length = hp.getLength();
        double prob = hp.getProbability();
        if (prob < _minProbability)
            _minProbability = prob;
        if (prob > _maxProbability)
            _maxProbability = prob;
        if (length > _maxLength)
            _maxLength = length;

        return result;
    }

    public double getMinProbability(){
        return _minProbability;
    }
    public double getMaxProbability(){
        return _maxProbability;
    }
    public int getMaxLength(){
        return _maxLength;
    }
}
```

---

org/arpodwot/heatmaps/highlighting/HighlightedPhrase.java 13/35

```
package org.arpodwot.heatmaps.highlighting;

import java.util.ArrayList;
import java.util.HashSet;

public class HighlightedPhrase {
    private int _length;
    private String[] _phrase;
    private double _probability;
    private int _matchCount;

    private HashSet<Integer> _docPositions;

    public HighlightedPhrase(){
        _docPositions = new HashSet<Integer>();
    }

    public int getLength(){
        return _length;
    }
    public void setLength(int len){
        _length = len;
    }

    public double getProbability(){
        return _probability;
    }
    public void setProbability(double prob){
        _probability = prob;
    }
    public int getMatchCount(){
        return _matchCount;
    }

    public String[] getPhrase(){
        return _phrase;
    }
    public void setPhrase(String[] phrase){
        _phrase = phrase;
    }

    public void addDocPositions(int[] positions){
        _matchCount++;
        ArrayList<Integer> tmpIndices = new
ArrayList<Integer>(positions.length);
        for (int i = 0; i < positions.length; i++){
            tmpIndices.add(positions[i]);
        }
        _docPositions.addAll(tmpIndices);
    }

    public Integer[] getDocPositions(){
        return _docPositions.toArray(new Integer[_docPositions.size()]);
    }
}
```

org/arpodwot/heatmaps/highlighting/ProbabilityGenerator.java 14/35

```
package org.arpodwot.heatmaps.highlighting;

import java.util.HashMap;

import org.apache.lucene.index.TermPositionVector;

public class ProbabilityGenerator {
    private HashMap<String, Double> _probabilityMap = new HashMap<String,
Double>();

    public ProbabilityGenerator(TermPositionVector index){
        String[] terms = index.getTerms();
        int[] freqs = index.getTermFrequencies();
        double sum = 0;
        // get the sum
        for (int i = 0; i < freqs.length; i++){
            sum += freqs[i];
        }

        // compute probabilities
        for (int i = 0; i < freqs.length; i++){
            _probabilityMap.put(terms[i], new Double(freqs[i] / sum));
        }
    }

    public double getPhraseProbability(String[] phrase){
        double prod = 1;
        for (int i = 0; i < phrase.length; i++){
            prod *= _probabilityMap.get(phrase[i]);
        }
        return prod;
    }
```

---

org/arpodwot/heatmaps/highlighting/SimpleDocumentHighlighter.java 15/35

```
package org.arpodwot.heatmaps.highlighting;

import java.util.ArrayList;

import org.apache.lucene.index.TermPositionVector;
import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.indexing.ngrams.SimpleAnalyzerNGramGenerator;
import org.arpodwot.heatmaps.notes.Note;
import org.arpodwot.heatmaps.util.Stopwords;

public class SimpleDocumentHighlighter implements DocumentHighlighter {
    @Override
    public void highlightDocument(InputDocument doc, Note[] notes) {
        TermPositionVector tpv = doc.getTermPositionVector();

        ArrayList<HighlightedNote> allHighlightedNotes =
                new ArrayList<HighlightedNote>(notes.length);

        // go find all of the matches
        for (Note n : notes){
            allHighlightedNotes.add(getNoteHighlighting(n, tpv));
        }

        Stopwords stops = new Stopwords();

        // initial values are 0
        double[] docHighlightValues = new
double[sum(tpv.getTermFrequencies())];

        // now apply their respective highlight values
        for (HighlightedNote note : allHighlightedNotes){
            double[] noteHighlightValues = new
double[docHighlightValues.length];
            double noteMinProb = note.getMinProbability();
            double noteMaxProb = note.getMaxProbability();
            int noteMaxLength = note.getMaxLength();
            double probRange = noteMaxProb - noteMinProb;
            for (HighlightedPhrase phrase : note){
                int phraseLength = phrase.getLength();
                if (phraseLength < 4)
                    continue;
                if (stops.areAllStopWords(phrase.getPhrase()))
                    continue;
                if (phrase.getLength() < 5 &&
stops.getStopwordPercentage(phrase.getPhrase()) >= 0.5)
                    continue;

                double normPhraseProb = ( probRange == 0.0 ? 1 :
(phrase.getProbability() - noteMinProb)/probRange );
                double phraseQuoteyness = 1 - normPhraseProb;
                phraseQuoteyness = phraseQuoteyness * (phraseLength /
noteMaxLength);
                if (phraseQuoteyness == 0)
                    continue;
                for (int position : phrase.getDocPositions()){
                    if (phraseQuoteyness > noteHighlightValues[position])
                    noteHighlightValues[position] = phraseQuoteyness;
                }
            }

            // add note highlights to doc highlights
            for (int i = 0; i < noteHighlightValues.length; i++){
                docHighlightValues[i] += noteHighlightValues[i];
            }
        }

        // put this highlight data into the InputDocument object
        doc.setRawHighlightData(docHighlightValues);
    }

    private HighlightedNote getNoteHighlighting(Note n, TermPositionVector
tpv){
        SimpleAnalyzerNGramGenerator ngrams =
SimpleAnalyzerNGramGenerator.createGenerator(n);
        HighlightedNote highlights = new
HighlightedNote(ngrams.getTotalCount());
        ProbabilityGenerator prob = new ProbabilityGenerator(tpv);

        // look for phrase matches, add them to the HP
        while(ngrams.hasNext()){
            HighlightedPhrase hp = new HighlightedPhrase();
            String[] ngram = ngrams.nextNGramArray();
            hp.setLength(ngram.length);
            hp.setPhrase(ngram);
            int rarestTermIndex = -1;
            int rarestTermFreq = Integer.MAX_VALUE;

            int[][] termPosVectors = new int[ngram.length][];
            for (int i = 0; i < ngram.length; i++){
                termPosVectors[i] =
tpv.getTermPositions(tpv.indexOf(ngram[i]));
                if (termPosVectors[i].length < rarestTermFreq){
                    rarestTermFreq = termPosVectors[i].length;
                    rarestTermIndex = i;
                }
            }

            // all of the positions that the rarest word in the ngram occurs
            int[] rarestTermPosVector = termPosVectors[rarestTermIndex];

            // how far we've gotten through the position array for each term
            int[] seekMarkers = new int[ngram.length];

            // go through each of the positions for the rarest word in the
ngram
            for (int i = 0; i < rarestTermPosVector.length; i++){
                boolean match = true; // each one is a potential match

                // this is where our searching is anchored
                int anchorPos = rarestTermPosVector[i];

                // this is where everything else should be
                int[] proposedMatch = new int[ngram.length];
                for (int j = 0; j < proposedMatch.length; j++){
                    proposedMatch[j] = anchorPos + (j - rarestTermIndex);
                }

                // for each of the other words in the ngram, see if they
                //    fall in the right place
                for (int j = 0; j < termPosVectors.length; j++){
                    if (j == rarestTermIndex) continue;

                    // the position array of this word that we're checking
                    int[] jthTermPositions = termPosVectors[j];

                    // the position that we want to find for this word, given
the

                    //   current value of p
                    int findPos = proposedMatch[j];

                    // look through this position array until we either find
                    //   the position we're looking for (findPos) or we
exceed it
                    for (; seekMarkers[j] < jthTermPositions.length;
seekMarkers[j]++){
                        // if we found the position we're looking for
                        if (jthTermPositions[seekMarkers[j]] == findPos)
                            break;
                        // if the current check position has exceeded the
position
                        //   we're looking for
                        if (jthTermPositions[seekMarkers[j]] > findPos){
                            match = false;
                            break;
                        }

                        // if we ran out of positions to check before finding
                        //   the correct one
                        if (seekMarkers[j] == jthTermPositions.length - 1){
                            match = false;
                            break;
                        }
                    }

                    if (!match)
                        break;
                }

                if (match){
                    hp.setProbability(prob.getPhraseProbability(ngram));
                    hp.addDocPositions(proposedMatch);
                }
            }

            if (hp.getMatchCount() > 0)
                highlights.add(hp);
        }

        highlights.trimToSize();
        return highlights;
    }

    private int sum(int[] x){
        int total = 0;
        for (int i = 0; i < x.length; i++){
            total += x[i];
        }
        return total;
    }
}
```

org/arpodwot/heatmaps/indexing/MultiDocumentIndexBuilder.java 16/35

```
package org.arpodwot.heatmaps.indexing;

import java.io.File;
import java.io.IOException;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.SimpleAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.document.Field;
import org.apache.lucene.document.NumericField;
import org.apache.lucene.index.IndexWriter;
import org.apache.lucene.store.FSDirectory;
import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.documents.input.InputDocumentCollection;

public class MultiDocumentIndexBuilder {
    public static IndexWriter.MaxFieldLength DEFAULT_MAX_FIELD_LENGTH =
IndexWriter.MaxFieldLength.UNLIMITED;

    public static void buildMultiDocumentIndex(String indexPath,
InputDocumentCollection docs) throws IOException {
        InputDocument[] docArray = new InputDocument[docs.size()];
        buildMultiDocumentIndex(indexPath, docs.toArray(docArray));
    }

    public static void buildMultiDocumentIndex(String indexPath,
InputDocument[] docs) throws IOException {
        String _indexPath = indexPath;

        // set various index settings
        Analyzer _analyzer = new SimpleAnalyzer();
        IndexWriter.MaxFieldLength _maxFieldLength =
DEFAULT_MAX_FIELD_LENGTH;

        // initialize the document template
        Document _doc = new Document();
        NumericField _idField = new NumericField("id", Field.Store.YES,
true);
        Field _filePathField = new Field("filePath", "Default",
Field.Store.YES, Field.Index.NO, Field.TermVector.NO);
        Field _dirPathField = new Field("dirPath", "Default",
Field.Store.YES, Field.Index.NO, Field.TermVector.NO);
        Field _fileNameField = new Field("fileName", "Default",
Field.Store.YES, Field.Index.ANALYZED_NO_NORMS,
Field.TermVector.WITH_POSITIONS_OFFSETS);
        Field _textField = new Field("text", "Default", Field.Store.YES,
Field.Index.ANALYZED_NO_NORMS, Field.TermVector.WITH_POSITIONS_OFFSETS);
        _doc.add(_idField);
        _doc.add(_filePathField);
        _doc.add(_dirPathField);
        _doc.add(_fileNameField);
        _doc.add(_textField);

        // initialize the IndexWriter
        File f = new File(_indexPath);
        FSDirectory dir = FSDirectory.open(f);
        IndexWriter writer = new IndexWriter(dir, _analyzer, true,
_maxFieldLength);

        // write each file to the index
        for (InputDocument doc : docs){
            _idField.setIntValue(doc.getId());
            _filePathField.setValue(doc.getFilePath());
            _dirPathField.setValue(doc.getDirPath());
            _fileNameField.setValue(doc.getFileName());
            _textField.setValue(doc.getText());

            writer.addDocument(_doc);
        }

        // close the IndexWriter
        writer.commit();
        writer.optimize();
        writer.close();
    }
}
```

---

org/arpodwot/heatmaps/indexing/MultiDocumentIndexSearcher.java 17/35

```
package org.arpodwot.heatmaps.indexing;

import java.io.File;
import java.io.IOException;
import java.util.HashMap;

import org.apache.lucene.analysis.Analyzer;
import org.apache.lucene.analysis.SimpleAnalyzer;
import org.apache.lucene.document.Document;
import org.apache.lucene.index.IndexReader;
import org.apache.lucene.index.TermPositionVector;
import org.apache.lucene.queryParser.ParseException;
import org.apache.lucene.queryParser.QueryParser;
import org.apache.lucene.search.IndexSearcher;
import org.apache.lucene.search.Query;
import org.apache.lucene.search.NumericRangeQuery;
import org.apache.lucene.search.TopDocs;
import org.apache.lucene.store.FSDirectory;
import org.apache.lucene.util.Version;
import org.arpodwot.heatmaps.documents.SearchableDocumentCollection;
import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.documents.input.SimpleTextDocument;

public class MultiDocumentIndexSearcher implements
    SearchableDocumentCollection
{
    public static Analyzer DEFAULT_ANALYZER = new SimpleAnalyzer();
    public static QueryParser DEFAULT_PARSER = new
QueryParser(Version.LUCENE_30, "text", DEFAULT_ANALYZER);

    private HashMap<Integer, InputDocument> _documentCache;

    private String _indexPath;
    private IndexReader _reader;
    private IndexSearcher _searcher;

    private QueryParser _parser;
    public MultiDocumentIndexSearcher(String indexPath) throws IOException {
        // initialize things
        _parser = DEFAULT_PARSER;
        _indexPath = indexPath;
        _documentCache = new HashMap<Integer, InputDocument>();

        openIndex();
    }

    private void openIndex() throws IOException {
        File f = new File(_indexPath);
        FSDirectory dir = FSDirectory.open(f);
        _searcher = new IndexSearcher(dir);
        _reader = _searcher.getIndexReader();
    }

    public void close() throws IOException {
        _searcher.close();
        _reader.close();

        _searcher = null;
        _reader = null;
    }

    @Override
    public int[] searchDocuments(String queryText) throws
        IOException,
        ParseException
    {
        if (_reader == null || _searcher == null)
            throw new IllegalStateException("This instance has already been
closed and is no longer usable.");

        Query query = _parser.parse(queryText);
        TopDocs hits = _searcher.search(query, _reader.numDocs());

        int[] ids = new int[hits.scoreDocs.length];
        for (int i = 0; i < hits.scoreDocs.length; i++){
            Document d = _reader.document(hits.scoreDocs[i].doc);
            ids[i] = Integer.parseInt(d.get("id"));
        }

        return ids;
    }

    @Override
    public InputDocument getDocumentById(int id) throws IOException {
        Integer idObj = new Integer(id);

        // see if we've already retrieved it from the index
        if (_documentCache.containsKey(idObj)){
            return _documentCache.get(idObj);
        } else {
            // if not, go get it!
            InputDocument retDoc = new SimpleTextDocument();

            // retrieve the document from the index
            Query q = NumericRangeQuery.newIntRange("id", idObj, idObj, true,
true);

            TopDocs hits = _searcher.search(q, 1);

            if (hits.totalHits == 0)
                return null; // couldn't find this id in the index!

            int internalDocId = hits.scoreDocs[0].doc;
            Document doc = _reader.document(internalDocId);

            retDoc.setId(Integer.parseInt(doc.get("id")));
            retDoc.setFilePath(doc.get("filePath"));
            retDoc.setDirPath(doc.get("dirPath"));
            retDoc.setFileName(doc.get("fileName"));
            retDoc.setText(doc.get("text"));

retDoc.setTermPositionVector((TermPositionVector)_reader.getTermFreqVector(in
ternalDocId, "text"));

            // add it to the cache
            _documentCache.put(idObj, retDoc);

            return retDoc;
        }
    }
}
```

---

org/arpodwot/heatmaps/indexing/ngrams/NGramGenerator.java 18/35

```
package org.arpodwot.heatmaps.indexing.ngrams;

import java.util.List;

public class NGramGenerator implements NGramQueryGenerator {
    private String[] _textBits;
    private int _bitCount;

    private int _previousNGramLength;
    private int _currentLength;
    private int _currentIndex;

    private int _totalCount;

    public static NGramGenerator createGenerator(List<String> textBits){
        String[] tb = new String[textBits.size()];
        return new NGramGenerator(textBits.toArray(tb));
    }

    public static NGramGenerator createGenerator(String[] textBits){
        return new NGramGenerator(textBits);
```

```
}

protected NGramGenerator(String[] textBits){
    _textBits = textBits;
    _bitCount = _textBits.length;
    _totalCount = ((_bitCount * _bitCount) + _bitCount) / 2;
    _currentLength = 1;
    _currentIndex = 0;
}

@Override
public String nextNGramString() {
    if (hasNext()){
        // jump to the next length if we need to
        if (_currentIndex + _currentLength > _bitCount){
            _currentIndex = 0;
            _currentLength++;
        }
        _previousNGramLength = _currentLength;

        StringBuilder ngram = new StringBuilder("\"");
        for (int i = 0; i < _currentLength; i++){
            if (i > 0)
                ngram.append(' ');

            ngram.append(_textBits[_currentIndex + i]);
        }
        ngram.append('"');

        _currentIndex++;

        return ngram.toString();
    }

    return null;
}

public String[] nextNGramArray(){
    if (hasNext()){
        // jump to the next length if we need to
        if (_currentIndex + _currentLength > _bitCount){
            _currentIndex = 0;
            _currentLength++;
        }
        _previousNGramLength = _currentLength;

        String[] ngram = new String[_currentLength];
        for (int i = 0; i < _currentLength; i++){
            ngram[i] = _textBits[_currentIndex + i];
        }

        _currentIndex++;

        return ngram;
    }
```

```
        return null;
    }

    @Override
    public boolean hasNext(){
        if (_currentLength > _bitCount)
            return false;

        if (_currentLength == _bitCount & _currentIndex + _currentLength >
_bitCount)
            return false;

        return true;
    }

    @Override
    public void reset() {
        _currentIndex = 0;
        _currentLength = 1;
    }
    public void reset(int index, int length){
        _currentIndex = index;
        _currentLength = length;
    }

    @Override
    public int getPreviousNGramLength(){
        return _previousNGramLength;
    }

    @Override
    public int getTotalCount(){
        return _totalCount;
    }
}
```

org/arpodwot/heatmaps/indexing/ngrams/NGramQueryGenerator.java 19/35

```
package org.arpodwot.heatmaps.indexing.ngrams;

public interface NGramQueryGenerator {
    public int getTotalCount();
    public int getPreviousNGramLength();
    public String nextNGramString();
    public String[] nextNGramArray();
    public boolean hasNext();
    public void reset();
}
```

org/arpodwot/heatmaps/indexing/ngrams/SimpleAnalyzerNGramGenerator.java 20/35

```java
package org.arpodwot.heatmaps.indexing.ngrams;

import java.util.ArrayList;
import java.util.regex.Matcher;
import java.util.regex.Pattern;

import org.arpodwot.heatmaps.notes.Note;

public class SimpleAnalyzerNGramGenerator extends NGramGenerator {
    public static Pattern tokenPattern = Pattern.compile("[a-z]+");

    public static SimpleAnalyzerNGramGenerator createGenerator(Note n){
        return createGenerator(n.getText());
    }

    public static SimpleAnalyzerNGramGenerator createGenerator(String text){
        //String[] textBits = text.trim().toLowerCase().split("[^a-z]+");
        // using n.getText().trim().toLowerCase().split("[^a-z]+")
        //   yields an empty string in position 0 ("") if the note text
        //   begins with a non a-z character.
        //   This creates more tokens than actually exist, and causes
problems
        //   with token counts in other places in the code.
        // Ergo, any code using the .split() command should instead
        //   aggregate tokens with the regular expressions as shown below.

        String cleanText = text.trim().toLowerCase();
        Matcher m = tokenPattern.matcher(cleanText);
        ArrayList<String> tokens = new ArrayList<String>();
        while(m.find()){
            tokens.add(m.group());
        }
        String[] textBits = tokens.toArray(new String[tokens.size()]);
        return new SimpleAnalyzerNGramGenerator(textBits);
    }

    protected SimpleAnalyzerNGramGenerator(String[] textBits){
        super(textBits);
    }
}
```

---

org/arpodwot/heatmaps/matching/DocumentScoreAggregator.java 21/35

```java
package org.arpodwot.heatmaps.matching;

import java.util.HashMap;

import org.apache.lucene.search.ScoreDoc;

public class DocumentScoreAggregator {
    @SuppressWarnings("unused")
    private static final long serialVersionUID = 1L;

    private HashMap<Integer, Double> _scoreMap = new HashMap<Integer,
Double>();
    private int _maxScoreDocId;
    private double _maxScore;

    public DocumentScoreAggregator(){
        _maxScoreDocId = -1;
        _maxScore = -1;
    }

    public void incrementDocumentScores(int[] docs, double score){
        for (int i = 0; i < docs.length; i++){
            incrementDocumentScore(docs[i], score);
        }
    }

    public void incrementDocumentScores(ScoreDoc[] documents, double score){
        for (int i = 0; i < documents.length; i++){
            ScoreDoc sd = documents[i];
            incrementDocumentScore(sd.doc, score);
        }
    }

    public double incrementDocumentScore(int docId, double score){
        // docScore is either 0 or the existing score
        double docScore = 0;
        if (_scoreMap.containsKey(docId))
            docScore = _scoreMap.get(docId);

        // increase docScore by the score argument
        docScore += score;

        // store that new score in the map
        _scoreMap.put(docId, docScore);

        // if this is the new maximum
        if (docScore > _maxScore){
            _maxScoreDocId = docId;
            _maxScore = docScore;
        }

        return docScore;
    }

    public int getMaxScoreDocId(){
        return _maxScoreDocId;
    }

    public double getMaxScore(){
        return _maxScore;
    }
}
```

org/arpodwot/heatmaps/matching/MatchesFile.java 22/35

```
package org.arpodwot.heatmaps.matching;

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.indexing.MultiDocumentIndexSearcher;
import org.arpodwot.heatmaps.notes.Note;
import org.arpodwot.heatmaps.notes.NoteCollection;
import org.arpodwot.heatmaps.notes.SimpleNoteFile;

public class MatchesFile {
    public static HashMap<InputDocument, ArrayList<Note>>
getMatchesFromFile(String indexPath, String noteFilePath, String
matchFilePath) throws
        IOException
    {
        System.out.print("Initializing document index ... ");
        MultiDocumentIndexSearcher search =
                new MultiDocumentIndexSearcher(indexPath);
        System.out.println("Done!");

        System.out.print("Loading notes from file ... ");
        NoteCollection notes = new SimpleNoteFile(noteFilePath);
        System.out.println("Done!");


        HashMap<InputDocument, ArrayList<Note>> matches =
                new HashMap<InputDocument, ArrayList<Note>>();

        FileReader fr = new FileReader(matchFilePath);
        BufferedReader br = new BufferedReader(fr);

        System.out.print("Loading in matches from file ... ");
        String line;
        while ((line = br.readLine()) != null){
            if (line.trim().length() == 0) continue;

            String[] ids = line.split("\t");
            int noteId = Integer.parseInt(ids[0]);
            int docId = Integer.parseInt(ids[1]);

            Note n = notes.getNote(noteId);
            InputDocument doc = search.getDocumentById(docId);

            if (!matches.containsKey(doc))
                matches.put(doc, new ArrayList<Note>());

            matches.get(doc).add(n);
        }
        br.close();
        fr.close();

        System.out.println("Done!");

        return matches;
    }
}
```

---

org/arpodwot/heatmaps/matching/MatchScoreCalculation.java 23/35

```
package org.arpodwot.heatmaps.matching;

public interface MatchScoreCalculation {
    public double calculateMatchScore(int ngramLength);
}
```

---

org/arpodwot/heatmaps/matching/NoteDocumentMatcher.java 24/35

```
package org.arpodwot.heatmaps.matching;

import java.io.IOException;

import org.apache.lucene.queryParser.ParseException;
import org.arpodwot.heatmaps.documents.SearchableDocumentCollection;
import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.indexing.ngrams.SimpleAnalyzerNGramGenerator;
import org.arpodwot.heatmaps.notes.Note;

public class NoteDocumentMatcher {
    public static InputDocument findMatchingDocument(Note n,
SearchableDocumentCollection docs) throws
        IOException,
        ParseException
    {
        SimpleAnalyzerNGramGenerator ngrams =
SimpleAnalyzerNGramGenerator.createGenerator(n);

        MatchScoreCalculation calc = new NSquaredMatchScore();
        DocumentScoreAggregator scores = new DocumentScoreAggregator();
        while(ngrams.hasNext()){
            // parse the query and do the search
            int[] hits = docs.searchDocuments(ngrams.nextNGramString());
            scores.incrementDocumentScores(hits,
calc.calculateMatchScore(ngrams.getPreviousNGramLength()));
        }

        return docs.getDocumentById(scores.getMaxScoreDocId());
    }
}
```

---

org/arpodwot/heatmaps/matching/NSquaredMatchScore.java 25/35

```java
package org.arpodwot.heatmaps.matching;

public class NSquaredMatchScore implements MatchScoreCalculation {
    @Override
    public double calculateMatchScore(int ngramLength){
        return Math.pow(ngramLength, 2);
    }
}
```

---

org/arpodwot/heatmaps/notes/NoteCollection.java 26/35

```java
package org.arpodwot.heatmaps.notes;

public interface NoteCollection {
    public Note getNote(int index);
    public Note[] getNotes();
    public void reset();
    public boolean hasNext();
    public Note nextNote();
    public int size();
}
```

---

org/arpodwot/heatmaps/notes/Note.java 27/35

```java
package org.arpodwot.heatmaps.notes;

public class Note {
    private int _id;
    private String _note;

    public Note (int id, String note){
        _id = id;
        _note = note;
    }

    public Note (String note){
        _note = note;
    }

    public int getId(){
        return _id;
    }
    public void setId(int id){
        _id = id;
    }
    public String getText(){
        return _note;
    }
}
```

---

org/arpodwot/heatmaps/notes/SimpleNoteFile.java 28/35

```
package org.arpodwot.heatmaps.notes;

import java.util.LinkedList;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;

public class SimpleNoteFile implements NoteCollection {
    private String _filePath;
    private int _noteCount;
    private int _currentPosition;

    private LinkedList<Note> _list;

    public SimpleNoteFile(String filePath) throws IOException {
        _currentPosition = 0;
        _list = new LinkedList<Note>();
        _filePath = filePath;

        // read in the notes from the file
        FileReader fr = new FileReader(_filePath);
        BufferedReader br = new BufferedReader(fr);

        String note;
        while((note = br.readLine()) != null){
            _list.add(new Note(_list.size(), note));
        }
        br.close();
        fr.close();

        _noteCount = _list.size();
    }

    @Override
    public Note getNote(int index) throws IndexOutOfBoundsException {
        return _list.get(index);
    }

    @Override
    public Note[] getNotes() {
        return (Note[])_list.toArray();
    }

    @Override
    public void reset() {
        _currentPosition = 0;
    }

    @Override
    public boolean hasNext() {
        if (_currentPosition >= _list.size())
            return false;

        return true;
    }

    @Override
    public Note nextNote() {
        if (hasNext()){
            Note n = _list.get(_currentPosition);
            _currentPosition++;
            return n;
        }

        return null;
    }

    @Override
    public int size(){
        return _noteCount;
    }
}
```

---

org/arpodwot/heatmaps/util/Stopwords.java 29/35

```
package org.arpodwot.heatmaps.util;

import java.util.HashSet;

public class Stopwords {
    private String[] _stopwords = {
        "a","about","above","after","again",
        "against","all","am","an","and",
        "any","are","aren't","as","at",
        "be","because","been","before","being",
        "below","between","both","but","by", "can",
        "can't","cannot","could","couldn't","did",
        "didn't","do","does","doesn't","doing",
        "don't","down","during","each","few",
        "for","from","further","had","hadn't",
        "has","hasn't","have","haven't","having",
        "he","he'd","he'll","he's","her",
        "here","here's","hers","herself","him",
        "himself","his","how","how's","i",
        "i'd","i'll","i'm","i've","if",
        "in","into","is","isn't","it",
        "it's","its","itself","let's","me",
        "more","most","mustn't","my","myself",
        "no","nor","not","of","off",
        "on","once","only","or","other",
        "ought","our","ours","ourselves","out",
        "over","own","same","shan't","she",
        "she'd","she'll","she's","should","shouldn't",
        "so","some","such","than","that",
        "that's","the","their","theirs","them",
        "themselves","then","there","there's","these",
        "they","they'd","they'll","they're","they've",
        "this","those","through","to","too",
        "under","until","up","very","was",
        "wasn't","we","we'd","we'll","we're",
        "we've","were","weren't","what","what's",
        "when","when's","where","where's","which",
        "while","who","who's","whom","why",
        "why's","with","won't","would","wouldn't",
        "you","you'd","you'll","you're","you've",
        "your","yours","yourself","yourselves"};
    private HashSet<String> _stopwordSet;

    public Stopwords(){
        _stopwordSet = new HashSet<String>();
        for (String w : _stopwords){
            _stopwordSet.add(w);
        }
    }

    public boolean isStopWord(String w){
        return _stopwordSet.contains(w);
    }

    public boolean areAllStopWords(String[] words){
        for (String w : words){
            if (!isStopWord(w)) return false;
        }

        return true;
    }

    public double getStopwordPercentage(String[] words){
        double wordCount = words.length;
        double stopwordCount = 0.0;
        for (String w : words){
            if (isStopWord(w)){
                stopwordCount += 1;
            }
        }

        return stopwordCount / wordCount;
    }
}
```

org/arpodwot/heatmaps/util/Transformation.java 30/35

```
package org.arpodwot.heatmaps.util;

public interface Transformation {
    public double[] transform(double[] highlightData);
}
```

---

org/arpodwot/heatmaps/util/UnitIntervalTransformation.java 31/35

```
package org.arpodwot.heatmaps.util;

public class UnitIntervalTransformation implements Transformation {
    @Override
    public double[] transform(double[] highlightData) {
        double minHighlightValue = Double.MAX_VALUE;
        double maxHighlightValue = Double.MIN_VALUE;

        for (int i = 0; i < highlightData.length; i++){
            if (highlightData[i] < minHighlightValue)
                minHighlightValue = highlightData[i];
            if (highlightData[i] > maxHighlightValue){
                maxHighlightValue = highlightData[i];
            }
        }

        double highlightRange = maxHighlightValue - minHighlightValue;
        for (int i = 0; i < highlightData.length; i++){
            highlightData[i] = (highlightData[i] -
minHighlightValue)/highlightRange;
        }
        return highlightData;
    }
}
```

---

org/arpodwot/heatmaps/workflows/AddDocsToIndex.java 32/35

```
package org.arpodwot.heatmaps.workflows;

import java.io.IOException;

import org.apache.lucene.queryParser.ParseException;
import org.arpodwot.heatmaps.documents.input.InputDocumentCollection;
import org.arpodwot.heatmaps.indexing.MultiDocumentIndexBuilder;

public class AddDocsToIndex {
    public static void main(String[] args) throws
    Exception,
    IOException,
    ParseException
    {
        // args[0] = document filepath list file
        // args[1] = note file
        String docListFilePath = args[0];
        String indexPath = args[1];

        // load in all of the documents
        System.out.print("Loading Documents ... ");
        InputDocumentCollection docs = new
InputDocumentCollection(docListFilePath);
        System.out.println("Done!");

        // put documents into an index
        System.out.print("Building index ... ");
        MultiDocumentIndexBuilder.buildMultiDocumentIndex(indexPath, docs);
        docs = null; // we'll be accessing these via index shortly
        System.out.println("Done!");
    }
}
```

---

org/arpodwot/heatmaps/workflows/CreateVisualization.java 33/35

```
package org.arpodwot.heatmaps.workflows;

import java.io.File;
import java.util.ArrayList;
import java.util.HashMap;

import org.arpodwot.heatmaps.documents.input.InputDocument;
import
org.arpodwot.heatmaps.documents.output.HighlightedHTMLDocumentGenerator;
import org.arpodwot.heatmaps.documents.output.OutputDocumentGenerator;
import org.arpodwot.heatmaps.highlighting.SimpleDocumentHighlighter;
import org.arpodwot.heatmaps.matching.MatchesFile;
import org.arpodwot.heatmaps.notes.Note;
import org.arpodwot.heatmaps.util.Transformation;
import org.arpodwot.heatmaps.util.UnitIntervalTransformation;

public class CreateVisualization {
    public static void main(String[] args) throws Exception {
        String indexPath = args[0];
        String noteFilePath = args[1];
        String matchFilePath = args[2];

        HashMap<InputDocument, ArrayList<Note>> docNoteMap =
                MatchesFile.getMatchesFromFile(indexPath, noteFilePath,
matchFilePath);

        // for each document,
        //   Apply all notes to it, giving raw highlight data
        System.out.println("Calculating highlights for each document ...");
        SimpleDocumentHighlighter highlight = new
SimpleDocumentHighlighter();
        for (InputDocument doc : docNoteMap.keySet()){
            System.out.println("\t"+doc.getFileName());
            Note[] notes = docNoteMap.get(doc).toArray(new Note[0]);
            highlight.highlightDocument(doc, notes);
        }
        System.out.println("\t---Done!");

        // Rescale to 0.0-1.0
        System.out.print("Normalizing highlight values ... ");
        Transformation t = new UnitIntervalTransformation();
        for (InputDocument doc : docNoteMap.keySet()){
            doc.setRawHighlightData(t.transform(doc.getRawHighlightData()));
        }
        System.out.println("Done!");

        //   output to HTML
        System.out.println("Generating HTML output ...");
        OutputDocumentGenerator out = new HighlightedHTMLDocumentGenerator();
        for (InputDocument doc : docNoteMap.keySet()){
            String outFilePath =
doc.getDirPath()+File.separator+"HL_"+doc.getFileName();
            System.out.println("\t"+outFilePath);
            out.writeToFile(doc.getRawHighlightData(), doc.getText(),
outFilePath);
        }
        System.out.println("\t---Done!\n");
        System.out.println("PROCESS COMPLETE");
    }
}
```

org/arpodwot/heatmaps/workflows/MatchNotesToDocs.java 34/35

```
package org.arpodwot.heatmaps.workflows;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import org.apache.lucene.queryParser.ParseException;
import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.indexing.MultiDocumentIndexSearcher;
import org.arpodwot.heatmaps.matching.NoteDocumentMatcher;
import org.arpodwot.heatmaps.notes.Note;
import org.arpodwot.heatmaps.notes.SimpleNoteFile;

public class MatchNotesToDocs {
    public static void main(String[] args) throws
        Exception,
        IOException,
        ParseException
    {
        // args[0] = document index path
        // args[1] = note file
        String indexPath = args[0];
        String noteFilePath = args[1];
        String matchOutputFilePath = args[2];

        // load in the notes
        System.out.print("Loading notes ... ");
        SimpleNoteFile noteCollection = new SimpleNoteFile(noteFilePath);
        System.out.println("Done!");

        // get ready to write the mappings file
        FileWriter fw = new FileWriter(matchOutputFilePath);
        BufferedWriter bw = new BufferedWriter(fw);

        // match the notes to their respective documents
        System.out.println("Matching notes to documents:");
        MultiDocumentIndexSearcher searcher = new
MultiDocumentIndexSearcher(indexPath);
        int noteCount = 0;
        while (noteCollection.hasNext()){
            if (++noteCount % 100 == 0)
                System.out.print("\t"+noteCount+" of
"+noteCollection.size()+"\r");

            Note n = noteCollection.nextNote();
            InputDocument doc = NoteDocumentMatcher.findMatchingDocument(n,
searcher);
            if (doc == null) continue;

            bw.write(n.getId()+"\t"+doc.getId()+"\n");
        }

        bw.close();
        fw.close();

        System.out.println("\t---Done!");
    }
}
```

org/arpodwot/heatmaps/workflows/MultiNotesMultiDocs.java 35/35

```
package org.arpodwot.heatmaps.workflows;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

import org.apache.lucene.queryParser.ParseException;
import org.arpodwot.heatmaps.documents.input.InputDocument;
import org.arpodwot.heatmaps.documents.input.InputDocumentCollection;
import
org.arpodwot.heatmaps.documents.output.HighlightedHTMLDocumentGenerator;
import org.arpodwot.heatmaps.documents.output.OutputDocumentGenerator;
import org.arpodwot.heatmaps.highlighting.SimpleDocumentHighlighter;
import org.arpodwot.heatmaps.indexing.MultiDocumentIndexBuilder;
import org.arpodwot.heatmaps.indexing.MultiDocumentIndexSearcher;
import org.arpodwot.heatmaps.matching.NoteDocumentMatcher;
import org.arpodwot.heatmaps.notes.Note;
import org.arpodwot.heatmaps.notes.SimpleNoteFile;
import org.arpodwot.heatmaps.util.Transformation;
import org.arpodwot.heatmaps.util.UnitIntervalTransformation;

public class MultiNotesMultiDocs {
    public static void main(String[] args) throws
        Exception,
        IOException,
        ParseException
    {
        // args[0] = document list file path
        // args[1] = document index path
        // args[2] = note file
        String docListFilePath = args[0];
        String indexPath = args[1];
        String noteFilePath = args[2];

        // load in all of the documents
        System.out.print("Loading Documents ... ");
        InputDocumentCollection docs = new
InputDocumentCollection(docListFilePath);
        System.out.println("Done!");

        // put documents into an index
        System.out.print("Building index ... ");
        MultiDocumentIndexBuilder.buildMultiDocumentIndex(indexPath, docs);
        docs = null; // we'll be accessing these via index shortly
        System.out.println("Done!");

        // load in the notes
        System.out.print("Loading notes ... ");
        SimpleNoteFile noteCollection = new SimpleNoteFile(noteFilePath);
        System.out.println("Done!");

        // match the notes to their respective documents
        System.out.println("Matching notes to documents:");
        MultiDocumentIndexSearcher searcher = new
MultiDocumentIndexSearcher(indexPath);
        HashMap<InputDocument, ArrayList<Note>> docNoteMap =
                new HashMap<InputDocument, ArrayList<Note>>();
        int noteCount = 0;
        while (noteCollection.hasNext()){
            if (++noteCount % 100 == 0)
                System.out.print("\t"+noteCount+"\r");

            Note n = noteCollection.nextNote();
            InputDocument doc = NoteDocumentMatcher.findMatchingDocument(n,
searcher);
            if (doc == null) continue;

            if (!docNoteMap.containsKey(doc))
```

```
                docNoteMap.put(doc, new ArrayList<Note>());

            docNoteMap.get(doc).add(n);
        }
        System.out.println("\t---Done!");

        // for each document,
        //   Apply all notes to it, giving raw highlight data
        System.out.println("Calculating highlights for each document ...");
        SimpleDocumentHighlighter highlight = new
SimpleDocumentHighlighter();
        for (InputDocument doc : docNoteMap.keySet()){
            System.out.println("\t"+doc.getFileName());
            Note[] notes = docNoteMap.get(doc).toArray(new Note[0]);
            highlight.highlightDocument(doc, notes);
        }
        System.out.println("\t---Done!");

        // Rescale to 0.0-1.0
        System.out.print("Normalizing highlight values ... ");
        Transformation t = new UnitIntervalTransformation();
        for (InputDocument doc : docNoteMap.keySet()){
            doc.setRawHighlightData(t.transform(doc.getRawHighlightData()));
        }
        System.out.println("Done!");

        //   output to HTML
        System.out.println("Generating HTML output ...");
        OutputDocumentGenerator out = new HighlightedHTMLDocumentGenerator();
        for (InputDocument doc : docNoteMap.keySet()){
            String outFilePath =
doc.getDirPath()+File.separator+"HL_"+doc.getFileName();
            System.out.println("\t"+outFilePath);
            out.writeToFile(doc.getRawHighlightData(), doc.getText(),
outFilePath);
        }
        System.out.println("\t---Done!\n");
        System.out.println("PROCESS COMPLETE");
    }
}
```

Generated by GNU Enscript 1.6.5.2.