Links - A Literature Visualisation and Analysis Tool

by

Eoin Nolan

Supervisor: Professor Mike Brady

Dissertation

Presented to the
University of Dublin, Trinity College
in fulfilment
of the requirements
for the Degree of

Master in Computer Science

University of Dublin, Trinity College

22nd May 2014

Declaration

I, Eoin Nolan, declare that the following dissertation, except where otherwise stated, is entirely my own work; that it has not previously been submitted as an exercise for a degree, either in Trinity College Dublin, or in any other University; and that the library may lend or copy it or any part thereof on request.

Eoin Nolan

22nd May 2014

Acknowledgements

I would like to thank Professor Mike Brady for his excellent advice, anecdotes, and support throughout the year. The guidance he offered was an invaluable help in completing this dissertation. I would also like to thank Clare Hayes-Brady for providing the initial idea for the dissertation and for taking time out of her schedule to contribute valuable feedback from the perspective of the user.

Eoin Nolan

Links - A Literature Visualisation and Analysis Tool

Eoin Nolan

University of Dublin, Trinity College

Supervisor: Professor Mike Brady

The aim of this dissertation is to demonstrate the creation of a literature visualisation and analysis tool that can augment the study of literature, while still remaining open and intuitive in its presentation of features and data. The tool is designed to work on an iPad in an effort to tap into the growing mobile market.

The central motivation of the tool, named *Links*, is to enable knowledge discovery by gathering interesting quantitative data about the text and visualising it to the user in such a way that it highlights thought-provoking sections. From exploration of the data, the user should gain a deeper understanding of the underlying relationships, themes, and writing styles in the text, which they may then use in their own studies.

The technical design and implementation outlined in this dissertation explains how the data metrics needed to make this possible may be gathered. Data is gathered through the use of natural language processing, in which grammatical tagging is used to apply lexical categories to each word.

A focus is placed on analysing word frequency, relationships, and sentiment within the text. Additionally, the design places a particular emphasis on creating a user-friendly interface that enables exploration, filtering, and annotation of the underlying data, by following recommendations from previous research. The resulting design is compared and contrasted to similar analysis tools to evaluate the performance of its design.

A qualitative evaluation of the tool shows that the data produced by *Links* meets the aim of supporting knowledge discovery within literature. In particular, the data metrics that are gathered can reliably highlight the most important relationships in a text, and provide and overview of the sentiment as the piece progresses.

Contents

List of Figures

List of Tables

List of Algorithms

1 | Introduction

The advent of the computer fundamentally changed the way in which we in interact with data. Problems of scale, work that would have taken decades to complete, could now be done overnight at the press of a button. This power opened new avenues of research within the arts disciplines, merging computing and humanities to form a brand new field of research; the digital humanities.

The digital humanities are vast areas of research covering countless different topics and tools. Indeed there are numerous examples of this research and the interesting data it can produce being carried out within *Trinity College* itself [I]. This dissertation focuses on the area of literature analysis. Literature, and indeed all natural language, is a complex mesh of characters, imagery, emotion, and sentiment. Analysis of literature is an in depth process in which one must navigate between hidden meaning and relationships in the text. Depending on the scale of the analysis, this is a substantial amount of work in which the outcome is very much tied to the opinions and thoughts of the reader.

The motivation behind this dissertation is to create a tool that that can leverage natural language processing and data visualisation techniques to make the process of knowledge discovery easier during literature analysis.

The tool, named *Links*, focuses on four main metrics of analysis: frequency and distribution of words, word relationships throughout the text, difference analysis, and sentiment analysis. It is believed that by analysing these metrics, the user will gain the ability to easily find interesting areas of the text, such as highly positive paragraphs or sections where two major characters appear together. With this ability to emphasise important sections of the text, the user may supplement their traditional research methods by browsing pertinent data and annotating their findings.

[I]TCD Digital Humanities `https://www.tcd.ie/trinitylongroomhub/themes/digital-humanities/`

The aim of the tool is not to replace the qualitative insight that the user has about a piece of literature; instead, *Links* provides a new method of large-scale visualisation, analysis, and annotation that would not have been possible before. Data is not limited to single sources; in fact some of the most interesting data is produced when multiple sources are contrasted against one another; for example, to show differences in vocabulary usage between two authors.

Many of the tools designed for use within the digital humanities unfortunately go underutilised because of their complexity. There is an ongoing debate within the field about the usability of the tools available. While many tools focus purely on the quality of the data they return, the lack of attention to user interaction often hampers adoption rates among researchers. Gibbs and Owens carried out a survey of tool creators finding that only 33% ever conducted usability reviews of their tools [1]. They conclude with the finding that only 6% of scholars will use these tools, and recommend that placing an emphasis on cultivating a broader audience should be a larger concern for creators. *Links* focuses on abstracting away this complexity so that the user can operate the tool with little to no training.

The recent change in the industry from physical control based interaction, through mouse and keyboard, to touch based interaction has been heralded as a paradigm shift in how applications are designed with usability in mind. *Links* is designed to run on an Apple iPad and naturally a large importance is placed on user interaction in a touch-enabled context. The majority of current tools are designed for desktop interaction with a traditional mouse and keyboard setup. However, as the usage of both desktop and laptops continues to dwindle – having recently being overtaken in gross sales by tablets [2] – it was felt that the option for similar tools in a mobile context should be explored.

This dissertation is structured as follows: the second chapter provides an overview of currently available text analysis tools and algorithms and establishes the background on which the presented tool builds upon them. The third chapter presents a design and implementation of key components of the tool including algorithms, data structures, user interface design, and visualisation techniques. The fourth chapter analyses the performance of the tool both technically and qualitatively; the qualitative tests focus on the performance of the tool in a knowledge discovery context using literature from project Gutenberg. Chapter five outline recommendations for future iterations of the tool and different areas which may be improved upon. The final chapter, chapter six, concludes the report with a summary of the findings.

2 | Background

The tool presented in this dissertation builds upon the foundations of several different fields of research: *Natural language processing (NLP)* which examines how accurate parsing and analysis of text by a machine can be achieved; *data visualisation* which studies the best way to present complex data sets in a human readable form; and *human computer interaction (HCI)* research which studies how users interact with a system and the ways in which it can be improved.

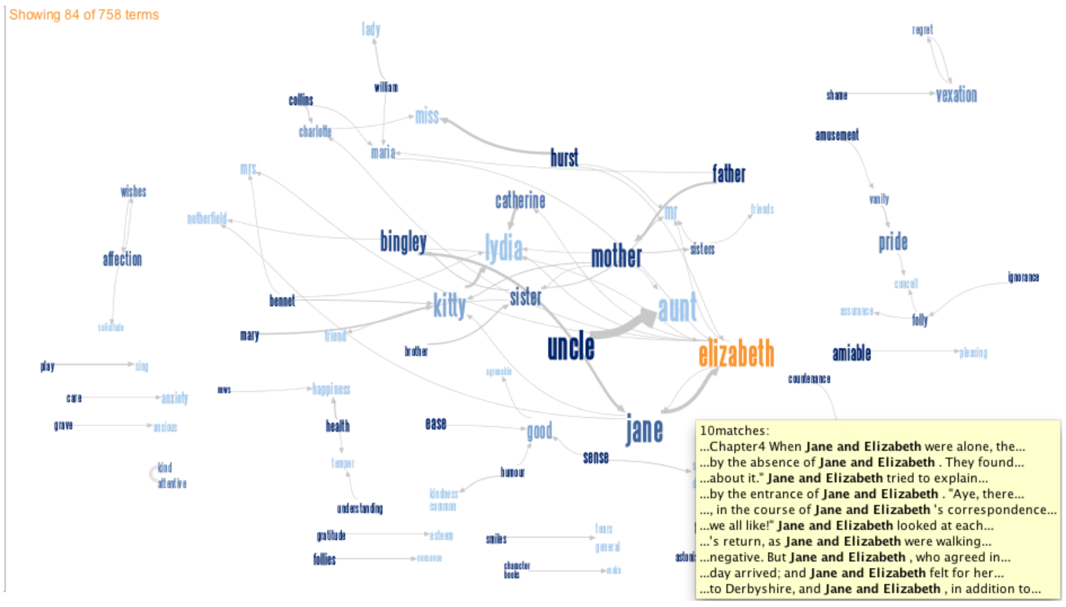
This chapter begins with Section 2.1, which provides an overview of the related tools in the area of text analysis and visualisation. Section 2.2 discusses the algorithms and data sets that are commonly used within natural language processing and text analysis, as well as the previous research in the area. Section 2.3 provides an overview of data presentation techniques that are commonly used in text visualisation tools. Section 2.4 presents a critique and evaluation of the presented items from the previous sections, and discusses their comparable uses within the design and implementation of *Links*.

2.1 | Related Tools

The digital humanities are a vast field of research and with a wide range of studies being pursued, there comes an equally wide range of tools to aid with research. The initial focus of this dissertation was to identify useful tools in the domain of text analysis and examine how they achieved their output, both through parsing algorithms and user interaction techniques. Many of the currently available tools aim to assist with a specific metric of text analysis, such as analysing sentiment or word usage. This section outlines the background and aims of these tools.

2.1.1 | Phrase Net

Phrase Net is a web-based text visualisation tool developed by IBM as part of their Many Eyes project [3] [4]. The tool visualises the pairs of words that are found across a text in a graph format. In this visualisation, the thickness of an edge between words represents the amount of times that pair appears. The tool uses word frequency to determine what to choose as the contents of the graph, limiting what is displayed to the most common fifty words by default. The intent of this tool is to visualise the major phrases in a book by linking common words across pattern phrases. For example, the user may define a pattern '* and *'. This will visualise links between common words where they match that pattern. In the example of *Pride and Prejudice*, the phrase *'Elizabeth and Jane'* is quite common and is therefore highlighted within the graph. Users may then select the edge to see the exact sentence where the pair has occurred.



10matches:
...Chapter4 When **Jane and Elizabeth** were alone, the...
...by the absence of **Jane and Elizabeth** . They found...
...about it." **Jane and Elizabeth** tried to explain...
...by the entrance of **Jane and Elizabeth** . "Aye, there...
..., in the course of **Jane and Elizabeth** 's correspondence...
...we all like!" **Jane and Elizabeth** looked at each...
...'s return, as **Jane and Elizabeth** were walking...
...negative. But **Jane and Elizabeth** , who agreed in...
...day arrived; and **Jane and Elizabeth** felt for her...
...to Derbyshire, and **Jane and Elizabeth** , in addition to...

Figure 2.1: A Phrase Nets graph displaying pairs found in *Pride and Prejdudice*

2.1.2 | Easy Text Classification with Machine Learning

Easy Text Classification with Machine Learning, or simply 'etcML', is a web based text analysis tool developed by computer scientists at Stanford University, which was released in 2013 [5]. The tool allows users to upload text and run analysis using built-in classifiers or user-generated classifiers. These classifiers will determine, for example, if an article has positive or negative semantics . Previous studies using the tool have analysed political bias against the Obama administration [6] and measured global sentiment about popular events by analysing trending Twitter feeds. EtcML places an emphasis on user-friendly interaction to differentiate it with tools that have traditionally been considered quite difficult to use. Richard Socher, lead developer on the tool described this design focus:

> "We wanted to make standard machine learning techniques available to people and researchers who may not be able to program." [5]

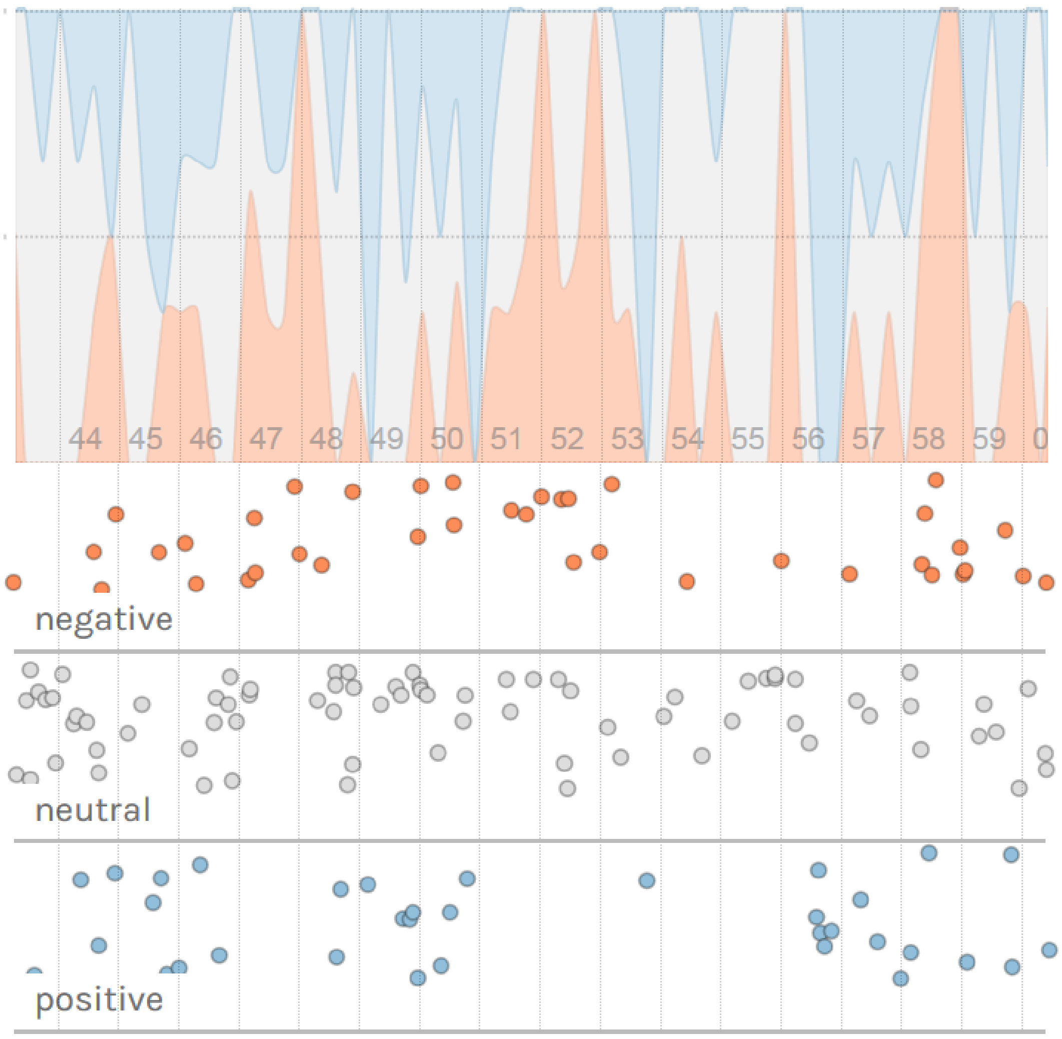


Figure 2.2: An etcML sentiment chart measuring sentiment for a popular social media topic.

2.1.3 | Google's N-Gram Viewer

Google provides an N-Gram viewer that uses data gathered from their vast collection of books spanning nearly two hundred years [1]. An N-Gram is a contiguous sequence of words within a text. They can be used to measure the probability of certain words appearing together. Previous research has used this method to generate probability maps to allow accurate parsing of text from sources like the Wall Street Journal [7]. With this data the user may see a distinct shift in an author's vocabulary, or they may measure how a specific word or phrase has changed in meaning over time. Such data may prove to be a useful asset for users attempting to identify certain shifts or changes within a text. The following example is taken from the N-Gram viewer; it shows the changing frequency in the usage of the word 'black' over two hundred years. Its usage roughly matched that of other colours until usage jumped in the 1960s to match the usage of the word 'white'. This jump coincides with the beginning of the American civil rights movements and is an example of the change in meaning that can be discovered by frequency distribution analysis of words or phrases [8].

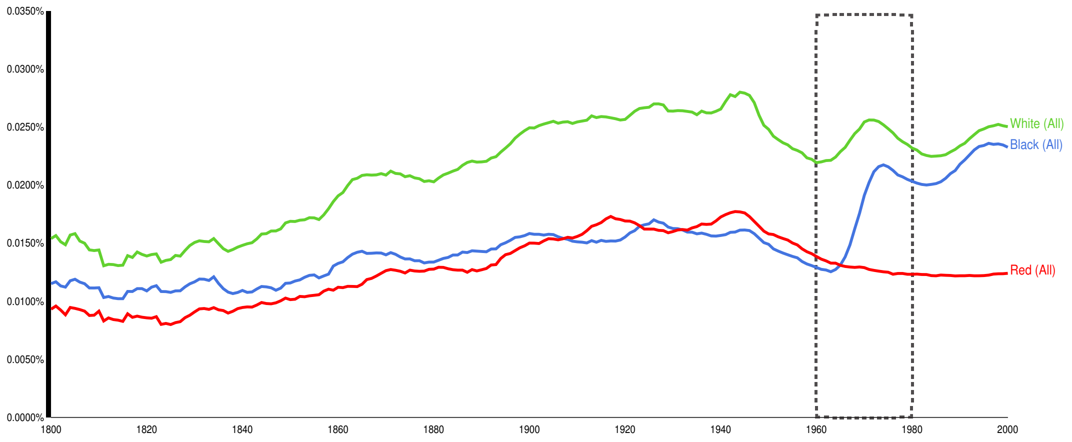


Figure 2.3: A N-Gram graph showing the doubling frequency of the word 'Black'.

2.1.4 | Java Graphical Authorship Attribution Program

Stylometry is a field of study that examines the linguistic style used in pieces of literature to determine the author of the piece. The frequency and distribution of function words – words that have little lexical meaning but express grammatical relationships – are examined across a set of texts to look for patterns. Tools for this purpose are widely available; one

[1]Googled N-Gram Viewer : `https://books.google.com/ngrams`

such example is JGAAP [II] (Java Graphical Authorship Attribution Program) which uses quantitative analysis techniques to attribute a text to predefined authors. This technology has been used to uncover unknown authors of literature pieces; most recently the novel *'The Cuckoo's Calling'* was revealed to have been written by J.K. Rowling, by researchers from Duquesne University, having been published under a pseudonym [9].

2.1.5 | Phraseology

Phraseology [III] is a commercial text editor for the iPad that provides the functionality to perform text tagging of your own content. From a user interaction perspective, Phraseology focuses on providing a user-friendly text analysis tool specifically designed to operate on the hardware provided by an iPad. The tool allows users to tag their speech usage and generate statistics on their word usage. It also allows for word stemming: showing the user the stem, or base, of any given word and its usage throughout their work. Phraseology is aimed towards aiding writers rather than literature researchers as is shown through its focus on writing style analysis tools.

2.1.6 | TopicNets

TopicNets is a visual analysis tool that analyses and assigns a topic to each book contained in a corpus [10]. The tool allows users to browse complex graphs listing academic theses from many different disciplines so that they may look for specific topics, overlaps, and connected works. It was developed as part of the WiGi research project run by the computer science department at the University of California, Santa Barbara. The focus of this project was to develop novel interactive interfaces for exploration of data, and to make large-scale data analysis available to users over a web interface. TopicNets presents a method of displaying complex graphs to users that are still interactive in real time despite their considerable size. This allows for exploration of the data rather than a traditional search and lookup interaction.

[II]JGAAP : `http://evllabs.com/jgaap/w/index.php`
[III]Phraseology : `http://agiletortoise.com/phraseology/`

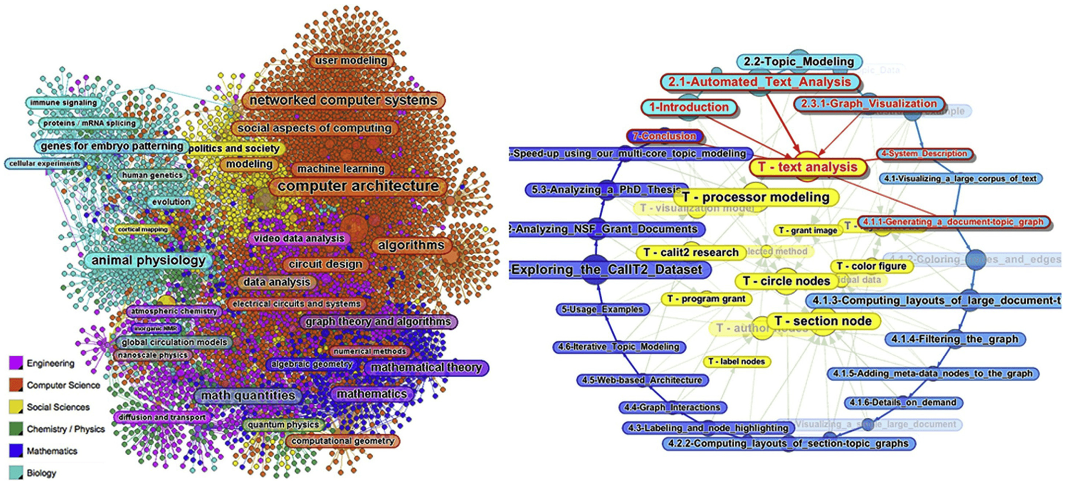


Figure 2.4: An example of dense and sparse TopicNets graphs

2.2 | Analysis Algorithms, Methods, and Data Sets

At its core, *Links* relies heavily on NLP to dissect texts to be able to provide useful feedback to the user about their contents. As such, there was a need to analyse the field of text parsing algorithms to determine which methods would provide the best trade-off between computation time and output accuracy. To complement these algorithms, additional text analysis data sets were examined; a data set in this context is a precompiled set of data that supplements the results of text parsing by allowing the tool to add additional metrics to words such as sentiment or emotion.

2.2.1 | Grammatical Tagging

Grammatical tagging, or part-of-speech tagging, is the process of identifying the lexical category of each word in piece of text [11]. Tagging relies on both sentence context and term definition to correctly apply a grammatical tag to each word. There are a variety of text parsing methods that have been developed for use in NLP. Due to the complexity and ambiguity of natural language, parsing is a particularly difficult problem to solve and indeed has not yet been solved completely [12]. The classic example of the difficulties presented by parsing natural language is given using the following quote:

> “Time flies like and arrow; fruit flies like a banana”.

In this case it is clear to see that correct tagging of words into their lexical category, noun, verb etc. is problematic because of the ambiguity of some words; *'Flies'* could refer to the insect or to the verb. Context is particularly important when parsing such sentences; central to these algorithms are classifiers that identify the surrounding words. There are two distinct groups of algorithms which perform tagging. The first group relies on a rule based approach that attempts to achieve accurate classification through supervised learning. The second group makes use of probabilistic models, such as the Bayesian approach, which analyses word co-occurrence frequencies to select the most likely lexical category. Accuracy of the output depends on the complexity of the classification. Keeping the number of lexical categories to a minimum reduces the likelihood of incorrect tagging occurring but affects the quality of the output.

2.2.2 | Word Frequency and Distribution

Having parsed the input text and classified each word, a clear place to start with the analysis is to examine the vocabulary the author uses in their work. A lower mean frequency of words used within the piece should indicate a greater vocabulary used by the author. Words with higher frequency could indicate that the author is using certain writing styles or themes [13]. Mean frequency is defined as being the number of occurrences of a word in comparison to the total word tokens in the piece. The overall usefulness of solely using frequency as an analysis metric in this manner has been questioned.

Baayen analysed the practicality of using word frequencies as a consistent measure of the vocabulary and writing style present in the piece [14]. Baayen's findings show that in natural language, the mean word frequency does not converge as a sample size increases. In reality, mean frequency will increase as the size of the corpus increases even with samples in the size of tens of millions of words. Baayen finds that natural language cannot be modelled as a random distribution of words. With the example of *Alice in Wonderland*, he shows that key words such as character names are topic sensitive and are tightly distributed in distinct groups throughout the book. The distribution of these key words becomes a much more interesting metric than simply frequency alone.

2.2.3 | WordNet

WordNet is an open source lexical database of the English language that was developed at Princeton University and is used in many major natural language applications and research projects [15]. The database maps the relationship between words by grouping them into synonymously related sets. In a text analysis tool this database can be used to further identify interesting points in the piece by mapping synonym relationships with frequently occurring words; one may similarly map antonyms for a given word to find contrasting sections in the piece.

2.2.4 | Sentiment Analysis

Sentiment analysis is the study of extracting the polarity or opinions of a piece of text [16]. The aim is to measure how positive or negative the sentiment of the particular piece is. This is particularly useful when analysing highly opinionated pieces such as media articles or reviews. The common method of extracting sentiment is to use a tailored dataset that assigns a polarity ranking to single words for a given context. Such data sets are available for movie reviews, sports coverage, political opinion pieces etc. This is known as the *'bag-of-words'* approach to sentiment analysis.

SenticNet is a tool developed by MIT that is used for concept level sentiment analysis of text. The main aim of SenticNet is to make the conceptual and affective information conveyed by natural language more easily accessible to machines. Similar to etcML, SenticNet can be used to measure user sentiment over data sets. Unlike etcML however, SenticNet provides open source parsing tools and a sentiment database. This parser represents the cutting edge of NLP research. It is more complex than simply applying numeric values to each word in a sentence; it searches for additional contextual information such as negations in the input. *'Happy'* would rank as positive but *'not happy'* would incorporate the negation into the result and give a negative ranking. Furthermore, it incorporates the *'bag-of-concepts'* model of parsing text in which the parser intelligently identifies the current concept to determine what the sentiment is [12]. For example, the word *'cloud'* may be negative in concept of *'weather'* but positive in the concept of *'computing'*. This type of parsing is envisioned to be the medium-long term future of NLP.

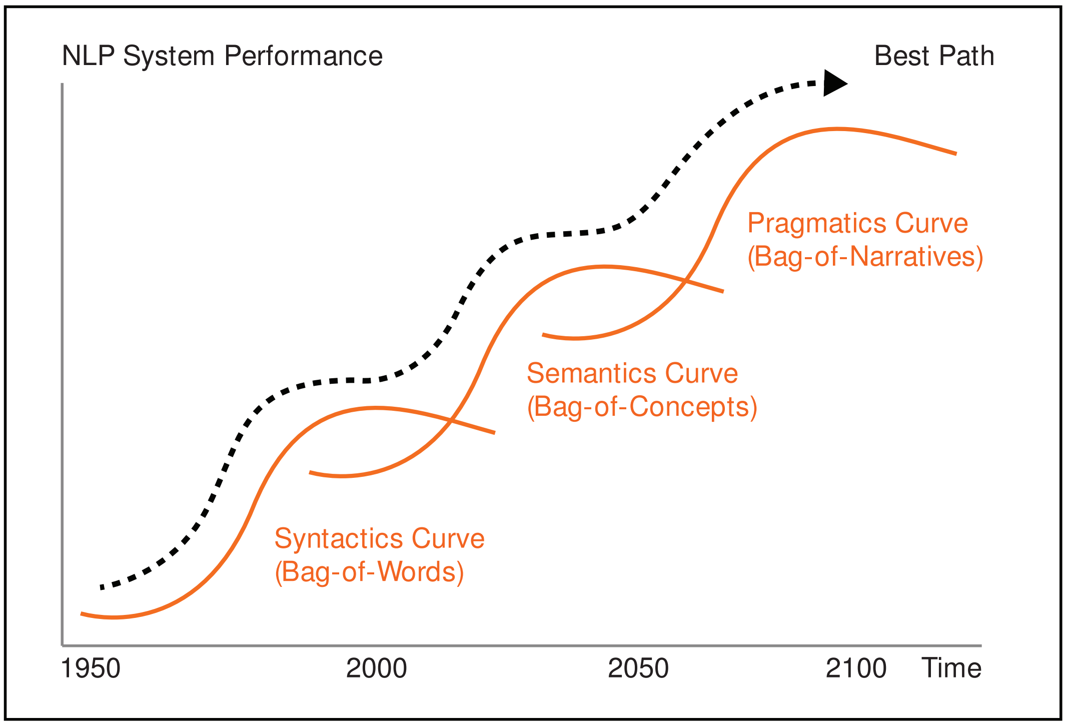


Figure 2.5: Envisioned evolution of NLP research through three different eras [12].

2.3 | Data Visualisation Techniques

Having parsed the input and generated useful results data, the final requirement is to present that data to the user in an intuitive and concise manner. With the advent of the touch interface there was a fundamental shift in how users interacted with computers, specifically, interaction with data sets, charts, and graphs became more ‘hands-on’ with natural panning and zooming now available to the user for the first time. As a data intensive tool, the focus of finding user-friendly visualisation techniques for *Links* was paramount. General visualisation techniques were assessed for the purpose of making data exploration as simple and natural as possible.

2.3.1 | Frequency & Distribution Visualisations

Visualising the frequency of words for users is an important step to assisting with the discovery of interesting data. Particularly common words may hint at the style and tone being used by an author as mentioned in Section 2.2.2. Visualisation of such data may enable the user to quickly analyse multiple books, contrasting two authors and their differing approach to writing. 'Word Clouds' are an extremely popular method of visualising the frequency of words in a piece of text. In a word cloud, the size of the word is directly proportional to the frequency in which it occurs in a text, making it easy for a user to identify the most common words at a glance. They have been commonly used on web pages as a means of quickly highlighting the key tag-words used within the page to allow users to view more content related to the topic. Word Clouds have several different styles of presentation, both aesthetic and functional (Figure 2.6). The aesthetic example gives an overview of the content but does not lend itself to exploration; placement of words is not constrained and is unpredictable. The functional example demonstrates the use of colour to signify importance as well as displaying the most common word in a prominent central position.

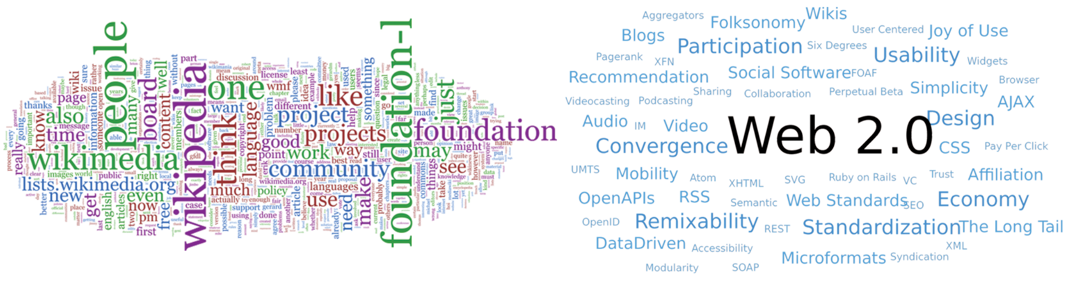


Figure 2.6: Two word clouds. Left: An aesthetic design. Right: A functional design.

2.3.2 | Graph Visualisation

Clearly visualising the frequently occurring word pairs present in a text would allow the user to identify complex relationships within the piece. Providing a method to explore the data is an inherently challenging task. For a large book, the graph produced is extremely dense with each node having a large set of edges. It is difficult to present this type of data in two dimensional space while retaining the user's ability to consume information and discover interesting data points; the sheer amount of data may result in an 'information overload'. With thousands nodes to be displayed, one must choose some method of prioritising which data is actually shown, and how this process can be controlled by the user.

The semantic web is one of the largest graph structures assembled and much research has been undertaken to determine optimal ways of uncovering this data to the common user.

Dadzie and Rowe examined this field with the intent of finding a visualisation technique that would allow non-technical audiences to *"obtain a good understanding of the semantic web's structure, and therefore implicitly compose queries, identify links between resources and intuitively discover new pieces of information"* [17]. Their suggestions focus heavily on providing an experience that allows for the generation of data overviews and presentation of query options to users in a simple format. Like the TopicNets approach, they show that colour and detail restriction can achieve comprehensible graph visualisations.

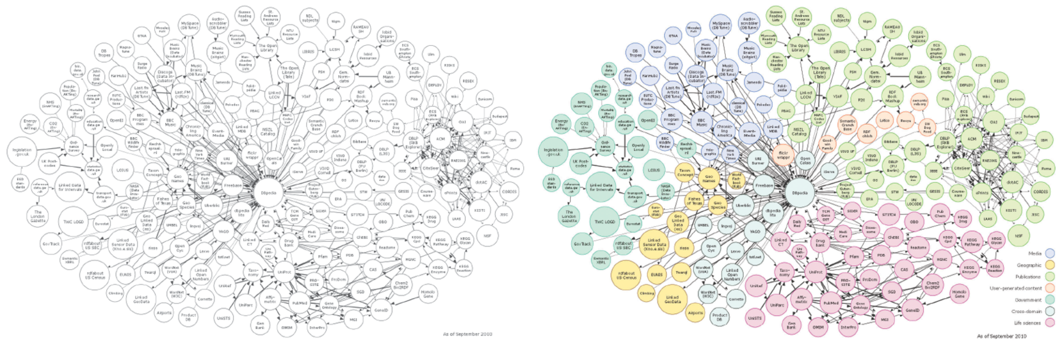


Figure 2.7: A dense graph showing the improvement made in legibility through use of colour categorisation [17]

IBM's Many Eyes project experimented with various different visualisation styles for data, with a specific focus on providing an interactive experience for the user [4]. One particularly relevant visualisation method provided by this tool is the 'Word Tree'. A Word Tree, as its name suggests, visualises a large tree of the possible sentences that occur within a piece of text which begin with a chosen root word. Essentially, it visualises the common *trie* data structure with entire words rather than single characters. While a tree structure is slightly less complex than a graph, as nodes do not link back to their parents or siblings, the visualisation style shown by IBM demonstrates that extremely large data structures can be presented in clear manner by representing each additional layer of the structure as a separate list.

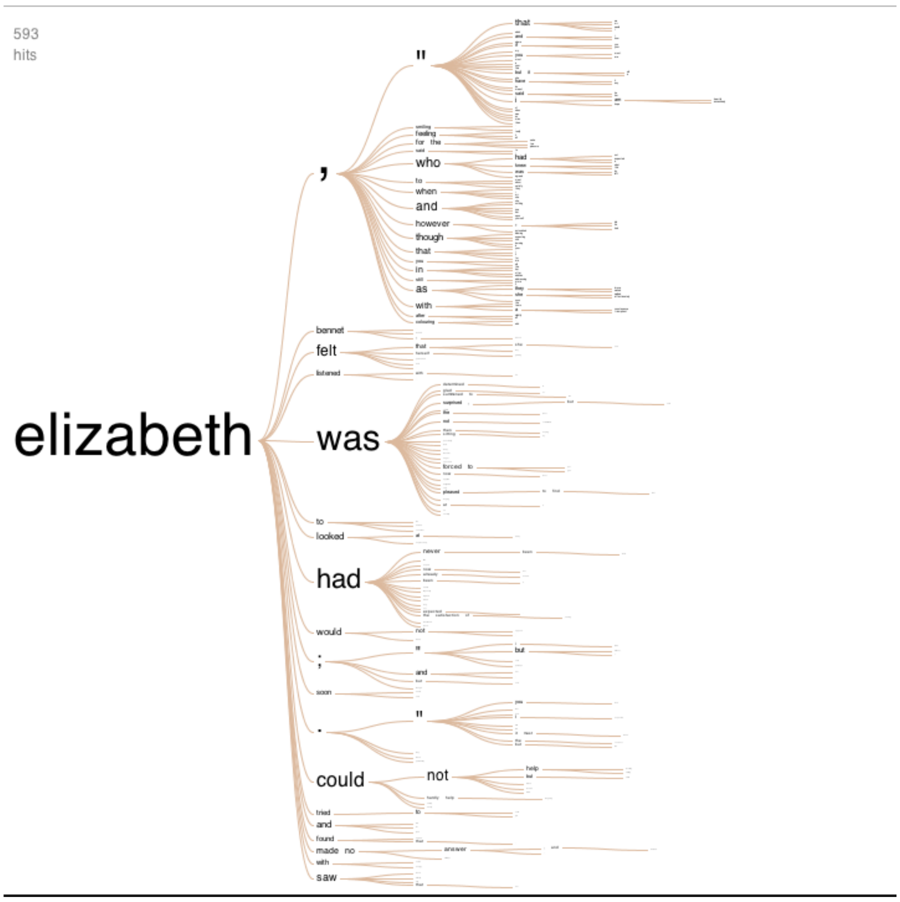


Figure 2.8: An example of the Word Tree visualisation for *Pride and Prejudice*

2.4 | Evaluation of Related Work

Having presented the related tools, algorithms, analysis methods, and visualisation techniques; this section discusses their relation to this dissertation, and how the design of the tool aims to build upon this work. Related work is evaluated and potential shortfalls of previous approaches are critiqued with a focus placed on how *Links* will avoid them.

2.4.1 | Evaluation of Related Tools

Instead of focusing on one specific area of analysis, *Links* brings together these various different methods so that it can provide the user with a wide range of analysis options through which they may further their studies. The tools presented in Section 2.1 demonstrate the scope of analysis options that are available. This section discusses the relationship between *Links* and these tools, and how their designs were relevant to the implementation of the tool.

Phrase Net is the most closely related tool to *Links* described in this chapter. As with *Links*, the tool has a strong focus on the visualisation of text, specifically in word relationships. Unlike *Links*, however, the tool places less emphasis on knowledge discovery. This is evident in how user interaction with the data is handled; users must explicitly search for pairs within the text by defining their own patterns. This design choice means that highly frequent word pairs may go unseen by the user simply because their template does not match. Essentially, the tool moves the burden of quality data generation onto the user.

The tool does not provide in-depth data filtering functionality which restricts the usefulness of the data that is being presented. The design choices made during the implementation of *Links* were motivated by the goal of creating a tool that would offer a more interactive data presentation design than the Phrase Net approach. The relatively poor data generation seen in the output of Phrase Net also highlights the need for higher quality relationship detection. Due to the similarities of the two tools, both the technical design and visual design choices made during the implementation of *Links* will be compared and contrasted to those of Phrase Net in Chapter 3, with a focus on how the shortcomings of this approach have been addressed.

EtcML presents a good case study in creating simple user interaction with complex analysis tools. Their approach to calculating and visualising the sentiment of social media data sets is an example of how this process may be achieved by similar tools. The visualisations produced by etcML (Figure 2.2) show how sentiment charts can be used to easily present the user with an immediate overview of the sentiment of a data set, and also how knowledge discovery may be introduced by linking sentiment points back to the underlying data. This approach to visualisation and exploration influenced the design of the sentiment analysis feature of *Links*.

Phraseology demonstrates that a user-friendly text analysis tool is possible to implement on the limited hardware power provided by mobile devices, and that intuitive interaction with such a tool is achievable through the exclusive use of a touch interface. As a commercial tool, it provides evidence that there is a demand for such analysis tools on the market.

Finally, though focusing on an unrelated area to *Links*, the work done by TopicNets offers a range of approaches for effectively visualising large data sets. Their techniques allow for the presentation of large graphs that remain navigable by the user, serving to aid with knowledge discovery within the data. Their use of multiple detail levels, colour coding, and data filtering reduce the graph to a comprehensible size. *Links* builds its own visualisation, and exploration tools upon some of the work done by TopicNets.

2.4.2 | Evaluation of the Presented Algorithms, Data Sets, and Methods

As a tool developed for use on an iPad, the choice of implementation languages was restricted to Objective-C; it was therefore important to identify the best approach for performing the tagging process described in Section 2.2.1 given frameworks and libraries available for this language. Speed and accuracy were the most important factors of this decision; selecting a powerful text tagging algorithm that would take an extremely long time to execute on the limited hardware power provided by the iPad was not acceptable. Apple developed a word tagging library as part of their core ObjectiveC frameworks, `NSLinguisticTagger`[IV]. This tagger is a counterpart to *Siri*, Apple's voice search tool; both technologies were made available with iOS5. The underlying classification methods of `NSLingusticTagger` are proprietary so the author cannot comment on their advantages or trade-offs. The tagger allows for a wide variety of classification types, including named entity recognition, whereby names and organisations are classed differently to regular nouns, and stemming in which the root of a word is returned. The text parsing speed of the tagger is extremely quick. This may be due in part to the access Apple has given the tagger to the lower level functionality of the operating system; this level of access would be unavailable to custom parsing tools as they would be limited to operation within the iOS application sandbox. With proven speed, accuracy, and usage in industry, `NSLinguisticTagger` was a natural choice for use within *Links*.

The results found by Baayen (Section 2.2.2) demonstrates that the distribution of words presents a higher quality data metric than simply measuring frequency. Indeed, the data shown by Google's N-Gram viewer (Section 2.1.3) illustrates the usefulness of measuring word distribution in a real world context. JGAAP (Section 2.1.4) similarly shows the merits of collecting this data for literature analysis, specifically for the purposes of identifying unique attributes of the text. The quality and usefulness these metrics has been clearly demonstrated by these tools; therefore, *Links* collects and retains this information for each book processed to provide the user with similar analysis functionality.

[IV]`NSLinguisticTager:` `https://developer.apple.com/library/ios/documentation/cocoa/reference/NSLinguisticTagger_Class/`

The merits of sentiment analysis have been shown by etcML, and the work done with SenticNet (Section 3.3.3) show how this may be achieved technically. Due to the computational complexity of the type of parsing SenticNet employs, the *'bag-of-concepts'* model, the approach would be too computationally intensive to be realistically implemented on an iPad. Nevertheless, sentiment analysis may instead be provided through the use of generalised sentiment datasets. AFINN [18] and SentiWordNet [19] are examples of such datasets. AFINN is a human generated data set that gives a manually assigned polarity ranking to nearly 2,500 English words and phrases. AFINN was designed to be useful in a variety of texts as opposed to specific items like film reviews, and this makes it extremely useful for a generalised tool like *Links* where the input is not guaranteed to be part of any one particular context. SentiWordNet is a contrast to AFINN in that it is automatically generated using classifiers. These classifiers leverage the synonym information available in WordNet to generate a much larger set of polarising words than AFINN; nearly 30,000. The accuracy of the output may not be as high as a human generated dataset but the added coverage may provide better average results.

2.4.3 | Evaluation of Data Visualisation Techniques

Section 2.3.1 outlines the ability of word clouds to intuitively convey the contents of a text in a visual fashion. As visualisation is a cornerstone of *Links*, this type of design could prove useful in providing an overview of the text to the user. The tool builds upon the functional word cloud approach by applying some layout constraints that make knowledge discovery a more intuitive process for the user.

For graph visualisation, the guidelines presented in Section 2.3.2 by Dadzie *et al*, and the methods utilised by TopicNets, present a useful staring point for interaction design; however, they were created with the assumption that the processing power of a desktop computer would be available. Without having that advantage, *Links* must also attempt to adhere to the guidelines while implementing visualisation techniques that remain computationally viable. Large scale presentation of graphs such as those seen in TopicNets is likely to be unattainable on a mobile device; therefore a focus on adapting the techniques to a mobile context is pursued in the design of *Links*.

The approach taken by IBM's Word Trees presents a good starting point for moving these visualisations to a mobile context. Though Word Trees are designed to visualise phrase usage, the approach may also be suitable for visualising relationships between nodes in a graph. Given the screen space limitations that a tool running on an iPad must operate within, this approach provides a suitable presentation style that also functions well within a touch context. Compared to the Phrase Nets approach (Figure 2.1), where the number of words displayed is restricted and edges are drawn between each visible node, the Word Tree approach allows for arbitrarily large lists. This feature benefits data exploration and knowledge discovery.

2.5 | Conclusion

This chapter discussed the work that has been done in the field of text visualisation and analysis, and its relevance to this dissertation. The background presented here strongly influenced the design and implementation of *Links*. The following chapter discusses specific design and implementation details, relating this work back to the information presented in this chapter by comparing and contrasting the work done to the approaches presented here.

3 | Design & Implementation

As an application designed to run on an iPad, the implementation was developed through the use of Objective-C and experimentation with additional Apple frameworks such as Quartz and Core Data. The choice of technologies and frameworks available on Apple devices influenced design decisions throughout the tool's development. This chapter explains how the tool was designed, the reasoning behind the design decisions that were made, and how they affected the outcome and effectiveness of the tools.

Section 3.1 discusses the advantages and drawbacks of developing the tool to run locally on an iPad versus a design which connects with a central server. Section 3.2 details the data structure design choices, what the requirements were and how it was implemented. It also compares the memory usage and speed of two separate approaches: in-memory and on-disk. Section 3.3 discusses the implementation of the parsing algorithms required to generate the data. It includes an examination of speed and memory performance, as well as trade-offs made. Section 3.4 outlines the design of the data queries within the tool, including the user requirements that drove their implementation, and their technical implementation. Section 3.5 describes the design of the annotation system within *Links*. The chapter concludes with Section 3.6, which describes the design of the interface with a particular focus on usability and data exploration.

3.1 | Local versus Remote Data Processing

The first design decision was to choose a data processing architecture on which to build the application. Data processing in this context refers to the three distinct operations: parsing and tagging of input text, creation of the required data structures, and analysis of the text's sentiment. The two choices were to process the data locally or to transfer data to a central server for processing and then querying that repository later. Processing data remotely has the advantage of being able to provide computationally powerful hardware, thus increasing the speed of both text parsing and data queries. A central server would also allow for the parsing algorithms and data query algorithms to be updated as needed rather than having to push out application updates. However, if one considers the possibility of multiple users simultaneously accessing the service, then a scaling problem is presented; processing in a central location produces a bottleneck as all users must wait for their requests to be scheduled and handled, as well as having to wait for data to be returned over the network. Scaling this design to handle large amounts of users is a difficult architecture problem with the solution simply being to increase the amount of available servers. While that is an acceptable solution, it is one that comes at a higher operational cost.

The alternative local processing implementation does not suffer from this scaling problem. Though the processing time is longer due to the slower hardware available on the devices, it remains constant regardless of the amount of users. Similarly, users do not require an internet connection and data queries do not need to be transmitted, saving both time and bandwidth.

With these considerations it was decided that *Links* use a local processing approach. As an academic tool with no focus on monetary profit, reliance on a central server was considered to be too much of a hindrance despite potential speed advantages. Operational and maintenance costs associated with running a central server would need to be financed by a benefactor; removing this requirement removes the associated costs. Future implementations of such a tool may examine the possibility of using a local server, potentially one that is operated by a university department, as a means to increase speed while not suffering from the drawbacks of a centralised architecture.

3.2 | Designing a Data Structure for Text Relationships

Choosing a central data structure for the tool was arguably the most important design decision made during the implementation. The performance of the application is directly linked with the performance of the data structure that the users manipulate and query. The algorithms within the application, and the type of data that needed to be returned from queries, dictated the requirements of the data structure. This section discusses these requirements and how the data structure was designed in response the potential queries a user would perform, and the trade-offs between time complexity and space complexity.

3.2.1 | Data Structure Requirements

As with any data intensive application, the requirements of the underlying data structures are driven by the type of operations that will be carried out. Trade-offs between time complexity and size complexity need to be fine tuned to create the most efficient data structure possible. As an application running on relatively weak mobile hardware, namely an iPad, *Links* prioritises the time complexity of data operations over space complexity. Data storage is reasonably plentiful on these devices in comparison to CPU power. By increasing space complexity of the underlying data structure, it was possible to decrease time complexity to a point whereby queries could be carried out in an acceptable amount of time, normally instantaneously. This has the drawback of making initial processing slower as more data needs to be produced, however, offsetting computation time into the processing phase produces a superior user experience.

With speed being a key requirement, the choice of underlying data structures was made to prioritise constant time operations. The tool operates on tens of thousands of unique words and an order of magnitude more relationships between words. To enable queries such as word lookup and comparison of word usage across multiple data sets, lookup operations need to be handled in constant time. This allows the application to quickly check if a certain word or relationship exists without the need to search the entire data set, i.e. $\mathrm{O}(1)$ lookup time versus $\mathrm{O}(n)$ lookup time, where n is the size of the database. Storage of these words needs to be contained within a structure that allows for this requirement.

The calculation of relationships between words at parsing time greatly increases the space complexity of the data structure, but decreases the time required for later queries. A graph structure naturally lends itself to this kind data; each word becomes a node in the graph and the edges between nodes represent the relationships that have been found within a text. Relationships need to be weighted to measure their relevance in a text, therefore a weighted graph structure is the most suitable design for this task.

3.2.2 | An In-Memory Object Based Approach

Having chosen a graph data structure, the overall design could then be implemented with an object based approach; that is, each node and edge of the graph has a corresponding in-memory object representing it. In-memory approaches allow for extremely fast lookup functionality as no data needs to be retrieved from disk at any point. This additional speed-up is particularly useful for the computationally weak devices the tool is designed to run on.

A `Node` object consists of several pieces of data: the word it represents, the lexical category it belongs to, a list of edges to other nodes, the frequency of occurrence of the word within the text, and the positions in the text where the word appears. The edges are stored within an `EdgeList` object, this is simply an extension of the `NSMutableSet` class that allows for additional operations to be built into the object such as selecting edges that link to specific lexical categories, or selecting edges that have met a certain minimum weight. The `Edge` object consists of a pointer to both left and right `Node` objects and has a weight associated with it.

Unlike traditional graph data structures where nodes are discovered through searching algorithms such as Breath-First Search or Depth-First Search, the graph used within *Links* required constant time lookup of each node and edge as described previously. This is achieved by storing each object within a central hash-table. Nodes are placed in the table using a key that is unique to the node. This key is generated through the following function:

```
Node Key = hash(word ++ lexical category)
```

Including the lexical category in key creation is extremely important as it allows the user to distinguish between the different meanings of words as they are used within the text, e.g. ‘flies’ as a noun and ‘flies’ as a verb. Similarly, edges are keyed and stored within a hash table. Given two nodes A and B, the key for the edge between them is found as follows:

```
Edge Key = hash( ( MIN(A.key, B.key) << 32) | MAX(A.key, B.key) )
```

This method of edge keying has the advantage of being commutative whereas simply appending the two keys and hashing would produce different keys depending on the order of the nodes. Therefore the same edge can be found between two nodes, A and B, regardless of the order they are parsed in.

When a graph is successfully constructed it is archived to disk so that reconstruction of the graph does not need to occur every time the application begins. This is achieved through use of the `NSKeyedArchiver` class. This archiver can intelligently store object graphs to disk by tracking the objects that have already been saved, and ignoring those that have been previously encountered; therefore, cyclic graphs such as those found within *Links* do not become stuck in an infinite loop. However, the archiver is subject to stack memory constraints. Continuously recursing on an object's children can cause a stack overflow.

This was a problem in the initial implementation of *Links* whereby saving extremely large graphs would crash the application. By recursing on every Edge object and saving the nodes and their children, there could easily be thousands of stack frames created. To solve this problem the Edge object is set up to no longer recurse on its related nodes; instead, it saves the two keys associated with these nodes and returns from the archival function immediately. When reloading a graph from memory, the nodes are loaded first followed by the edges. When an Edge object is loaded it reconstructs the graph by looking up the two Node objects it forms an edge between by using the previously saved keys. The pointers to these nodes are then saved and the keys are discarded.

3.2.3 | Additional Application Data

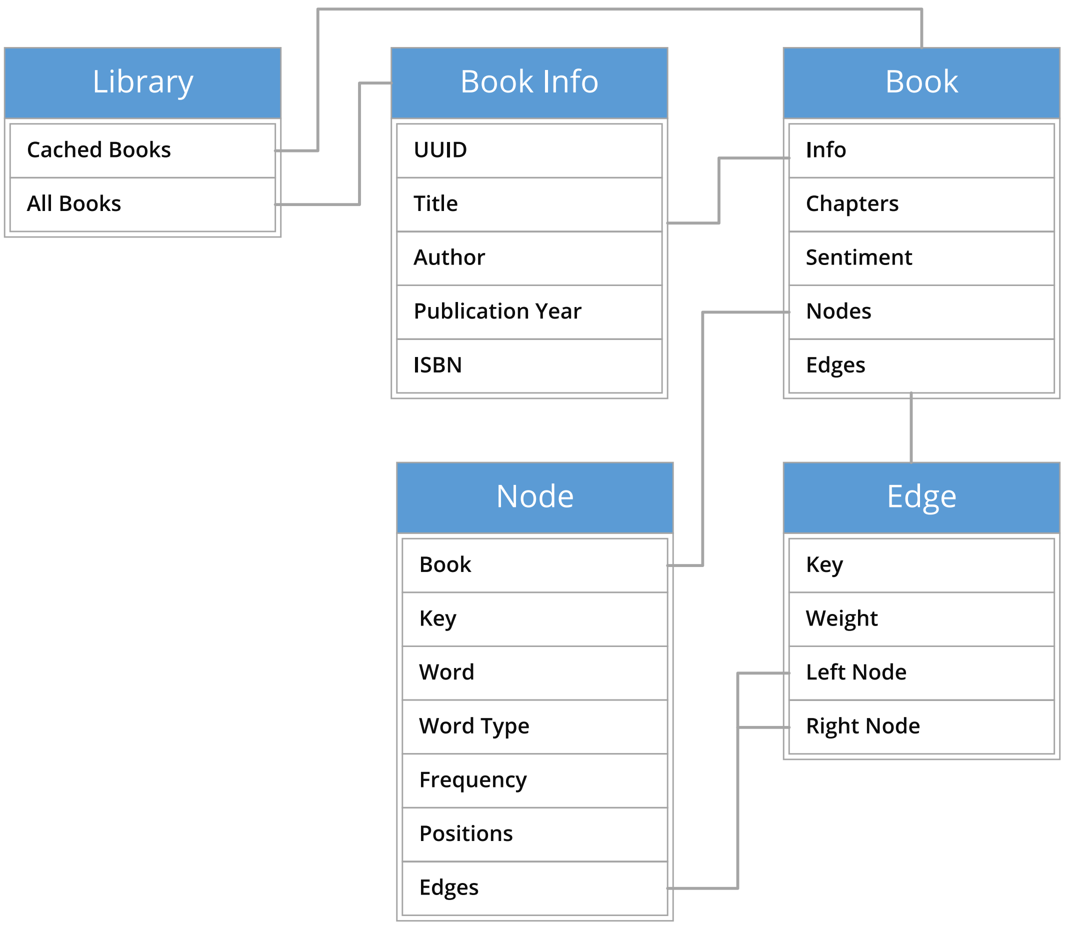


Figure 3.1: An overview of the data model used within *Links*

The root object of the application is the `Library` object, a singleton class that may be accessed by any part of the application. This object contains a cache of books that have already been loaded from disk, which allows the application to prevent needless reloading any time a book is accessed more than once. Accompanying the cache is a list of all books in the library; this is a list of `BookInfo` objects. A `BookInfo` object contains all the necessary information required to present details of the book to the user in the user interface without having to load the entire object graph it relates to. This information includes the title, author, ISBN, descriptions, and a UUID. The UUID is used for finding the corresponding graph on disk, e.g. `<UUID.archive>` will load the related book. A UUID was used for this purpose rather than the title or ISBN as these would restrict the user to only being able to load a single copy of a given novel, or create collisions if the titles of two novels were the same.

Each book is saved to disk contained within a `Book` object. This object contains the nodes and edges of the graph in hash-tables as previously described. Coupled with the graph is a list of chapters and their positions within the text, along with a measurement of the sentiment of the piece.

3.2.4 | An On-Disk Approach using Core Data

Despite the speed advantages provided by an in-memory approach, it presents an upper limit on the amount of text that may be analysed, as a finite amount of data structures may be loaded into memory at any one time. Single pieces of text are unlikely to present a memory problem as, for example, a 100,000-word novel requires only around 10MB of space. However, when analysing an entire corpus consisting of many large novels it is possible that storing the data entirely within memory may not be feasible. With this possibility clearly limiting *Links*, experimentation with the use of an on-disk data store was carried out.

Core Data is the Apple framework designed for this purpose. The framework allows for the creation of *'managed object contexts'* which assigns the responsibility of object management to the system; this includes moving objects from being stored in an SQL database on disk, to being cached in memory. The retrieval of these objects from disk is transparent to the developer, and objects contained in a managed context can be periodically flushed to disk to prevent a large memory overhead. Similarly, data is only read from disk when it is required; queries can be run over the entire database and only the relevant data will be stored in memory on completion in what is called the persistent store [20]. In comparison to the in-memory approach, the Core Data method may also completely use available memory, as it has the option to simply reset the context and empty memory. This advantage means it can theoretically work with much larger data sets.

Modifying the application to make use of the Core Data framework involved creating an adapted data model for the graph structure. Graph data structures do not naturally lend themselves to the relational model that Core Data is built upon so some changes were made. These changes revolve around ensuring that each `Node` or `Edge` is linked to a specific `Book` by creating a new relationship between them, and that `Node` positions – where the word occurs within a piece of text– are saved to a separate table. With an object-based approach these relationships are explicit; positions are stored within the `Node` objects themselves, and nodes and edges are stored within a parent `Book` object. References to nodes and edges are cached in memory to prevent repeated lookup from the persistent store.

The performance of the standard in-memory implementation was compared against the Core Data implementation. Both implementations used the same parsing and graph construction algorithms; only the data storage was different between the two. Despite being designed for data access, the speed performance of the Core Data implementation was poor in comparison to the in-memory approach. While it did allow for lower run time memory usage, its memory usage during the parsing stage was nearly five times greater. This greater usage may be due to the fact that objects created during the process are not being autoreleased by the persistent store. Disabling autorelease on the in-memory implementation creates equally high memory usage; this suggests that temporary objects created during parsing are not being released until much later in the process. It may also be attributed to the extra data needed to fit the graph data structure into a relational form; for example, each edge table row contains a reference to a Book object which introduces an additional overhead. More concerning was the lookup performance offered by Core Data. During graph creation the parser will produce pairs of nodes that need to be linked. An edge may exist between these nodes already so the algorithm must determine if it is incrementing the weight of an existing edge or creating a new one. This is handled using a lookup; fetch requests used by Core Data were exponentially slower than the in-memory design and access time increased with the size of the database. Timing performance is a crucial factor in the choice of data storage and because of the poor results shown by Core Data it became impractical to use. Shown in Table 3.1 are the average timings taken during testing.

	Words Parsed	**Memory Usage**	**Creation Time**	**Edge Lookup**
Core Data	150,000	108MB	17 seconds	6.7ms
	300,000	240MB	58 seconds	16ms
In-Memory	150,000	24MB	12 seconds	0.0004ms
	300,000	44MB	26 seconds	0.0004ms

Table 3.1: Memory and speed results of the implemented data storage methods.

3.2.5 | Optimisations

The original implementation of the graph structure included an array in each `Edge` object to store the locations of where the relationship appears in the text; this was intended to make the process of locating relationships – where two words appear in close proximity within the text – more efficient. The extra memory overhead this created was too costly to be retained; each `NSMutableArray` has a 250-byte minimum memory cost to allow for insertions and internal variables. In a large book with an excess of 100,000 edges, storing positions created the need to have roughly 25MB of extra data in memory. This could account for nearly half of the overall size of the data structure. By omitting the edge locations and instead computing them when required based on the node locations, the need to store this data is removed.

Each `Node` object contains a list of location tuples detailing the character offset of the word in the text and the actual index of the word within the text. Two nodes share an edge if their index is within a specified adjacency limit. Edges can be found in $O(n)$ time by taking advantage of the fact that node locations are guaranteed to be sorted in order.

Algorithm 1: Locating Edge Positions

```
Result: Given two nodes, A and B, algorithm returns an array of edge positions
Matched = 0
Skipped = 0
for i = 0, i < Node A Locations do
    for j = Matched + Skipped, k = 0, j < Node B Locations and k < Adjacency Limit do
        IndexA = Node A Location[i]
        IndexB = Node B Location[j]
        // Check if the two indexes are outside the adjacency limit
        if IndexA > IndexB + Adjacency Limit then
            Skipped ++
            continue
        else if IndexA < IndexB - Adjacency Limit then
            break
        // When inside the limit the two indexes form an adjacent pair
        else
            Add Location to Edge Positions
            Matched ++
            k ++
        end
    end
end
return Edge Positions
```

3.3 Parsing Texts and Generating Data Structures

The initial task that must be carried out when loading a piece of text into the application is parsing. This process converts input from a plain text file into the previously described graph structure on which meaningful queries may be carried out. Parsing consists of three distinct operations: tagging, graph construction, and semantic analysis. This is the most processor intensive task that is carried out at any point during the execution of the application, and therefore the speed at which it completes is paramount to the usability of the tool. Parsing is only required once per book as the data produced during this stage may simply be read from disk during subsequent executions.

3.3.1 | Parsing With `NSLinguisticTagger`

Apple's `NSLinguisticTagger` provides the backbone to the text parsing algorithm used within *Links*. The `NSLinguisticTagger` handles the process of determining the lexical category of each word in a piece of text. Tagging allows the tool to separate words into distinct categories so that their usage may be measured throughout the piece. The tagger caters for most lexical classes present in English grammar; however, during the parsing process *Links* strips the words that are not considered to provide any useful information to the user, as shown in Table 3.2.

Retained		**Discarded**	
Nouns	Adverbs	Pronouns	Conjunctions
Adjectives	Adverbs	Prepositions	Determiners
Classifiers	Idioms	Particles	Numbers
		Interjections	Others

Table 3.2: Retained and discarded lexical categories.

The rationale behind removing certain lexical classes is twofold. Firstly, the words that are used from certain categories, such as pronouns *(he, she, their etc)* and conjunctions *(and, but, so etc)* are unlikely to significantly change from text to text; therefore their inclusion would add little additional information to the user's insight about a piece, while significantly diluting the information being presented to them. *Links* does not attempt to perform coreferencing: the process of linking pronouns with the noun representation. Secondly, storing these extra nodes within the graph would drastically increase the memory overhead required. Each extra word encountered means there would be an additional `Node` object as well as an extra set of `Edge` objects to go with it.

In general, these discarded classes make up a significant amount of the overall word usage within English text. By stripping them before the graph creation process, there is a non-trivial amount of memory which can be saved. Biber *et al* studied the usage of lexical classes in different writing styles [21]. Their results show that by removing the lexical classes in question, the total memory savings available range from 38.5% to 57%. Furthermore, the graph construction time is reduced linearly as the amount of nodes decreases, which results in a 40% speed increase in graph creation, and a 9% speed increase to the overall parsing algorithm. The linear complexity of parsing is shown in Figure 3.4.

Conversation							
Adverb	*Adjective*	*Verb*	*Noun*	*Pronoun*	*Preposition*	*Determiner*	*Other*
50	25	210	150	165	55	45	300
Academic							
Adverb	*Adjective*	*Verb*	*Noun*	*Pronoun*	*Preposition*	*Determiner*	*Other*
30	100	165	300	40	150	100	95

Table 3.3: Average lexical category usage (in thousands) per million words [21]

As each word is tagged, it is placed in a `WordToken` object; this object contains the word itself, a reference to the lexical class, a character offset into the text and a word count. The character offset is required to allow the user to return to the exact location of the given word in the text. The word count is a reference to the order in which the word was processed by the parser, ignoring white space and discarded words; e.g. *'Elizabeth is the twentieth word in the text'*. Character offsets cannot be used for this purpose, as there is no guarantee that the distance between two offsets indicates a edge location, regardless of how close they are. This is due to stripped words, punctuation, and white space creating unknown distances between each word in a pair. A simple ascending integer removes this ambiguity and allows for later computation of pair locations within the text. Once created, each `WordToken` is placed into an array and this is passed to the graph construction algorithm when parsing is complete.

3.3.2 | Memory Leaks in `NSLinguisticTagger`

During the development of the parsing algorithm there was an unusually high memory overhead which could not be attributed to any object allocations that were included within the code. From further analysis of the code it was discovered that `NSLinguisticTagger` was leaking memory during its execution. Removing any additional functionality, i.e. `WordToken` creation, and simply running the tagger over a piece of text could prove this theory. With no allocations in the code there should be little overhead from the tagger; however, the tagger would still acquire memory during its execution. The program would not release this memory even after parsing had completed. If the user were to load a large piece of text into *Links*, this leak would cause memory warnings and eventually cause the application to be terminated by the operating system.

It was discovered that any new line characters within the text being parsed would cause this memory leak. By removing all newlines from the input text, and carrying out tagging sentence by sentence, this leak was eliminated. This has the unfortunate effect of removing white space that may have been added by the author explicitly. It is unclear what the underlying cause of the leak is, as the implementation at the operating system level is unavailable to examine.

3.3.3 | Sentiment Analysis

Sentiment analysis takes place during the tagging process. The purpose of this analysis is to produce an overall picture of the sentiment of a piece and how it changes as the text progresses. This allows the user to highlight interesting sections where the sentiment may radically change. The analysis calculates the average sentiment of a small text segment; this average is used as a data point on a line chart. Each segment produces the average sentiment for six hundred words. This number was chosen after testing performance against a small number of known texts. It provides a good balance between data point resolution and delivering a high-level overview of sentiment for the text. If the segment section is too small, the data produced will not realistically reflect the actual sentiment intended by the author, as a relatively small amount of strongly positive or negative words can distort the data at that point. If the segment is too large then the data point will tend heavily toward neutrality, and the overall chart will not be able to highlight interesting sections as effectively.

The data sets that determine the polarity value of each word, AFINN [18] and SentiWordNet [19], have differing data representation formats and must be standardised for use within the application. AFINN ranks words with a polarity value between −5 and +5, whereas SentiWordNet ranks words between −1 and +1. Both data sets were parsed from their respective sources and added into an Objective-C dictionary, mapping each word to its polarity value normalised between −1 and +1. This is a process that only needs to be carried out once as the binary data of the dictionary can simply be loaded from disk when needed without additional parsing. There is no need to handle this process within *Links* itself; a separate command line tool was produced to read sentiment data sets and output a normalised sentiment dictionary that could be statically linked within the application.

As each word is parsed by the tagger, its polarity is checked using the sentiment dictionary. If the word is not found then it is implicitly declared neutral and given a polarity value of zero. The sentiment algorithm uses a sliding-window approach to generating segment averages. In this approach, the total sentiment for a sub section of two hundred words is calculated and placed on a stack. When a sub section is calculated, it is pushed onto the stack and the oldest section is removed. The average for three sections is calculated and a data point is saved for that text location. Five sentiment data points are calculated for every thousand words in the text. By calculating averages using overlapping sections in this way, the possibility of having dramatically different data points per segment is reduced and a more realistic picture of overall sentiment is presented. It is unlikely in most cases that the true sentiment of a piece will radically change in less than two hundred words; therefore, the sliding window approach effectively eliminates outlying data points on the line graph it produces. The data points produced are saved to disk for later use and an overall sentiment indicator is calculated for the book, ranking it very positive, positive, neutral, negative, and very negative. 0% to 10% is neutral, 10% to 20% is positive, and 20% to 100%, and the same for the negative direction. These ranges tend towards zero to reflect to how books seem to tend towards neutrality, regardless of their sentiment.

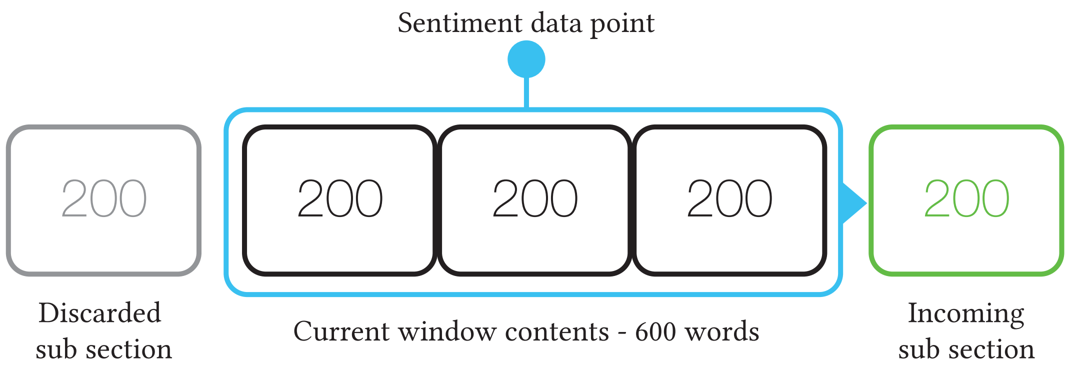


Figure 3.2: Representation of the sentiment analysis sliding window process

3.3.4 | Graph Construction

The graph construction processes essentially takes the complete set of `WordToken` objects from the parser and builds the graph structure that will be used throughout the program for queries. The process begins by iterating through each `WordToken` in the array received from the tagger. The tokens are provided by the tagger in the order that they appear in the text, which allows for word proximity – a measure of the relationship between two words – to be analysed. A `Node` object is created for each token as it is encountered for the first time, setting its initial frequency – or appearance count – to one. When a word is encountered that has been previously processed, the constructor is intelligent enough to fetch the existing node and increment its frequency rather than create a duplicate `Node` object.

After a `Node` is created, the algorithm gathers a set of the adjacent nodes succeeding it. For each of the gathered nodes, the algorithm creates an `Edge` object to represent the link between the node and its adjacent counterpart, that is, each `Edge` represents a relationship between two words that have been found within the text. As recurring word pairs within the text are discovered, the weight of the corresponding `Edge` object is incremented to reflect the regularity of this occurrence. Once processing is completed, the edges with the highest weight represent the strongest relationships within the text.

The adjacency limit governs the distance between words in which a pair is still considered valid; words within a close proximity governed by this limit are said to be related. A small set of adjacent nodes means that the construction process will be faster and the memory overhead will be smaller. A larger set will provide better quality word pairs for the user to explore. Consider the case where two main characters appear together in a novel; a word pair could capture this meeting if their names appeared close enough together. Given the following sentence: *"Alice thanked Bob and left."*, the useful word pair in this case would be `(Alice, Bob)`, however, this would only be encountered if the adjacency count was set to two or greater. This is the default adjacency count in *Links*; it provides good relationship generation between proper nouns and their descriptions, through adjectives and adverbs, without incurring a significant speed decrease during parsing.

Figure 3.3 represents the effect of increasing the adjacency count during processing of *Pride and Prejudice*. One may expect the rate at which edge counts increase to ease off as the adjacency becomes higher, this is because the probability that an edge already exists becomes greater as more pairs are encountered. However, as can be seen, this does not appear to be the case. As Baayen stated about word frequencies not converging as a corpus grows [14], similarly pair frequencies do not seem to converge at any significant rate as the adjacency increases. With no clear point of convergence, the choice of adjacency size becomes a question of data quality as there is no memory advantage to selecting a specific value.

In the case where the adjacency limit is set to a value greater than one, the weighting of the edges may be handled in several different ways. The simplest weighting strategy is to apply a weight of 1 to every encountered pair; this means that words appearing directly beside each other are given the same precedence as words that may be separated by several places. Other weighting strategies may apply a linearly decreasing value to the edges based on distance *(1, 0.9, 0.8 etc)*, or similarly, and exponentially decreasing weight *(1, 0.5, 0.25 etc)*. *Links* uses a static weighting strategy as it provides the clearest information to the user, that is, the final weight directly corresponds to the number of times the pair appears. This was considered to be more useful to the user than potentially having different weights for pairs that appear an equal amount of times.

Algorithm 2: The graph construction process

```
Result: The graph data structure required to represent the text
for i = 0, i < Word Tokens do
    Node CurrentNode = Construct Node (Word Tokens[i])
    for j = i + 1, k = 0, j < Word Tokens and k < Adjacency Limit do
        Node AdjacentNode = Construct Node (Word Tokens[j])
        Weight = Weighting Strategy (k)
        Add Edge Between (CurrentNode, AdjacentNode, Weight)
    end
end
return Graph
```

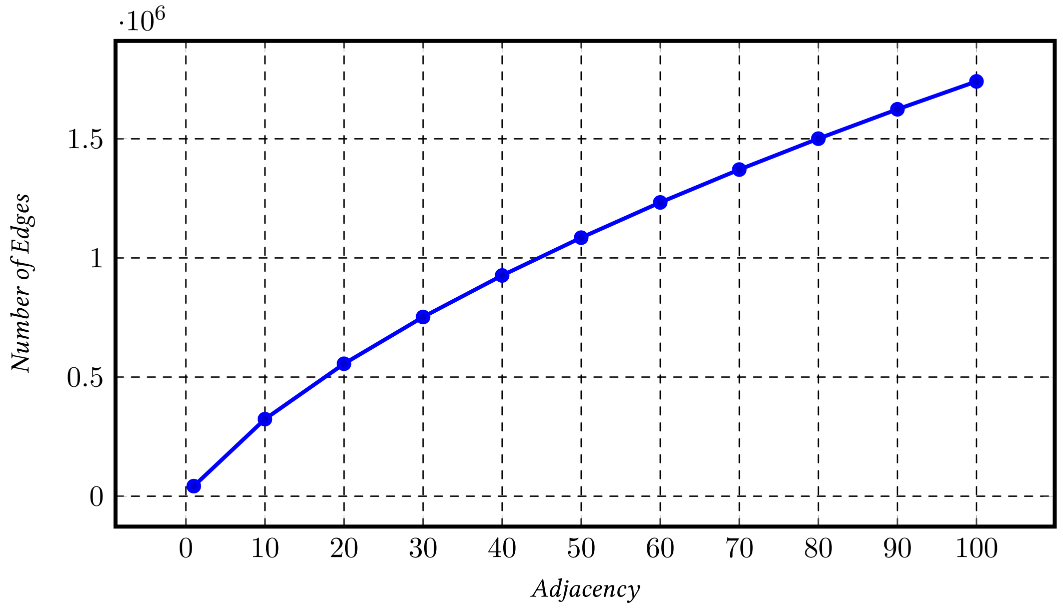


Figure 3.3: Linearly increasing edge count of the graph with increasing adjacency. Realistically, adjacency above ten will produce word pairs that are inconsequentially linked, thus reducing the effectiveness of relationship identification. Adjacency between zero and ten, and indeed to one hundred, shows a linear increase in edge allocations.

In comparison to the design taken by Phrase Net [3], whereby pairs are only generated if they conform to ridged templates, the approach taken by *Links* is more versatile as words are matched by proximity, which therefore allows for common pairs to be more easily detected. To further this benefit, proper nouns are treated separately to other lexical classes during edge calculation; two proper nouns are considered paired if they occur within a slightly larger proximity than any other combination of lexical categories. This has the positive effect of emphasising character and location relationships, while having little effect on the overall memory requirement of the graph.

The benefit of this approach is evident when compared to the Phrase Net approach. When visualising *Pride and Prejudice*, the link between Elizabeth and Mr Darcy is highly pronounced as it should be when one considers their tightly coupled relationship within the text. This same relationship does not register at all with the Phrase Net approach using any of the standard templates, and therefore the user cannot adequately explore where these interactions are happening.

3.3.5 | Speed and Memory Performance

Parsing and data structure generation is the most CPU intensive process that is carried out within the tool. As the most intensive task, it is also the slowest; it is the only process that requires the user to wait for any significant amount of time. User experience is closely tied to how the user expects an application to perform; if the user has an expectation to quickly load a book and begin exploring the data, a slow parsing process could risk frustrating them and ultimately cause them to stop using the tool.

In an attempt to mitigate this risk, the option of multithreading the parsing process was explored. The graph construction algorithm requires a comparatively short amount of time to complete in the overall process of generating the output data; it is roughly 20% of the required processing time assuming a low adjacency limit. The tagging process consumes the greatest amount of time and presented the greatest opportunity for increasing the overall speed.

Apple's documentation regarding `NSLinguisticTagger` states that any given instance must only be accessed by a single thread; this multithreaded implementation experiments with creating a separate instance per thread. When multithreading the parsing algorithm, the tagging process is split across a set amount of threads. In the case of the iPad, the amount of threads was fixed at two due to current processors in these devices providing two threads of execution. The option to increase this thread amount was retained to future-proof the tool for the likely release of quad core devices.

Each thread is given its own instance of an `NSLinguisticTagger` and a sub section of the overall text required to be parsed; parsing is then carried out as normal until all threads complete and their collective results are combined before beginning graph construction. The multithreaded parser shows a slight increase in performance, averaging 7% quicker than its single threaded counterpart. Removing any non-tagging related functionality from the thread — such as sentiment analysis or `WordToken` creation — causes the speed improvement to disappear. This indicates that multiple instances of the `NSLinguistic-Tagger` cannot be used simultaneously in different threads. It seems likely that the tagger requires exclusive access to a lower level component of the operating system as part of its operation. This is an obvious limitation on the speed of the parsing algorithm and is one that may only be sped up by a multithreaded implementation of the tagger. As a core component of the operating system, this is would have to be implemented by Apple. Despite the limitation, the 7% increase in speed in the other facets of the algorithm is non-trivial and contributes towards a more positive user experience.

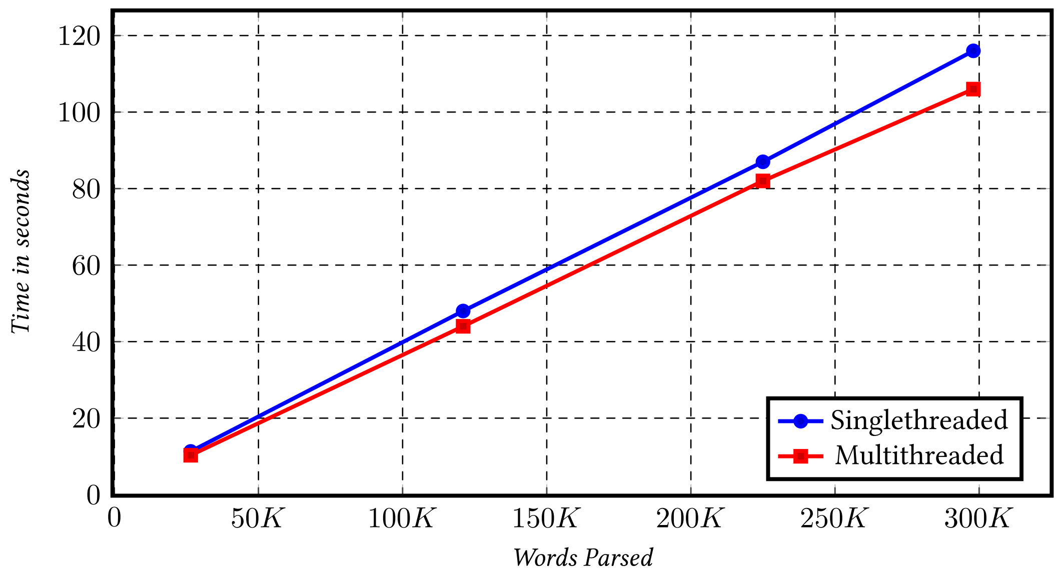


Figure 3.4: Speed Increases of a Multithreaded Parser

3.4 | Query Design

The motivation behind *Links* was to create a tool that that could leverage natural language processing to make the process of knowledge discovery easier during literature analysis. Central to making this possible is by providing the use of queries. Querying is the process of reducing a data set based on specific direction from the user. Queries are a fundamental component of *Links*; they provide the bridge from simply presenting raw data to the user, to enabling knowledge discovery by allowing users to take an entire data set and intelligently reduce it down to a manageable set of results. Queries are often the most difficult component to design within in a data intensive application [17]. Complex queries produce the highest quality set of results as the user may tailor multiple facets of the data to suit their needs. However, with greater complexity comes the added challenge of presenting these query options to the user in a way that is intuitive and easy to use. The downfall of many similar applications, as described by Gibbs and Owens [1], is in their lack of attention to this aspect of their design. Complex queries may provide excellent results, but if the user cannot understand how to make use of them then the quality of the results means little. Keeping with the theme of usability, the design for the query system in *Links* is based on providing the most powerful functionality possible through an interface that has little to no learning curve.

3.4.1 | Identifying User Requirements

Links began development with the intent of offering the user the ability to investigate and discover interesting data in literature. The core set of tasks that were considered important to provide include, but is not limited to:

Word Frequency Analysis Viewing word frequencies gives the user an immediate perspective of the vocabulary an author is using in a piece. This information allows the user to build opinions around the nature of the work, identify thematically important words, or words integrally linked with an issue dealt with in the text. For example, the theme of *family* may lend itself to high usage of the words 'sister', 'mother' etc. With specific words identified as being interesting, their distribution in the text can guide the user towards important sections of the book so that they may collect relevant quotations or contrast how multiple sections differ from one another. Chapters with a high concentration of interesting words highlight to the user where they may find evidence of themes or issues [14] [13].

From a technical perspective, accommodation of this kind of interaction requires queries that can restrict the lexical categories being shown to the user. For example the user may want to identify what adjectives the author is using in their descriptive prose to identify if it is overly positive or negative; filtering out all unnecessary lexical categories is crucial to making this straightforward.

Relationship Identification An important part of analysis is the process of identifying major characters and their relationships to other characters or places in the novel. As with word frequency, the ability to filter lexical category is required. Characters will naturally appear as nouns or proper nouns within the text, therefore by simply removing the non-relevant lexical categories the user will be presented with a set of characters or locations in the novel. Additionally, important characters or locations inherently have a higher frequency of appearance in the text than other nouns and will be presented more prominently in the output as a result. Having identified an important word such as a character, the relationships between this word and others in the text must be presented. These relationships are the word pairs that were generated during the initial parsing. The pairs and their frequency can help highlight relationships between the selected word and others in the book. For example, two characters may be paired; both the frequency of this pair and its distribution throughout the book provide interesting points of analysis for thus user.

Filtering by lexical category is again required for this process. To identify character relationships, the user will need to filter pairs that have the lexical category $proper\ noun \longleftrightarrow proper\ noun$ similarly if the user wishes to identify actions taken by a character they may filter using $proper\ noun \longleftrightarrow verb$ etc.

Contrasting Multiple Books Some of the most interesting data can be gathered by contrasting multiple works. An example of this usage would be to identify the words used within one novel that are of uniquely important to that work. A user may achieve this by comparing a single novel against the rest of the author's bibliography. The words that have an proportionally high frequency of usage may have a special significance to the novel and could allow the user to find interesting examples of their usage. Similarly, a user may want to perform Boolean operations to only show words that are common across a set of books, or to show words from one set of books that do not appear in another set. This kind of analysis allows the user to compare and contrast the vocabulary usage of multiple authors at once. Essentially, it allows the user to perform the previously described operations but at a much larger scale than a single book.

With judicious use of each of the filters: lexical category filters, pair filtering, Boolean filters, proportional filters, and traditional search functionality, the user will be able to create complex queries by layering functionality rather than specifying large complex queries directly.

3.4.2 | Technical Design

Filters are directly tied to a special `BookCollection` object. This object contains a set of all the books being analysed and the filter settings that are currently being applied. The reasoning behind tying the current collection to the current filters was to enable the quick transfer of filtered results between view controllers in the application. This provides the benefit of presenting the same set of results in each view type, that is, if a user performs a filter operation in the one visualisation, the results are automatically updated in any other visualisation of that data. Essentially it avoids the problem of results being out of sync across the application.

A `BookCollection` contains two distinct sets of books: a retain set, and a discard set. The first filter to be applied during a query is the negation filter. When this is applied, nodes appearing in any of the books from the discard set are removed from the final output.

This allows for Boolean filtering in the form $\{node \in Output : node \in RetainSet \wedge \notin DiscardSet\}$. Users may also specify that words must be common across each book in the retain set, this is described in the form $\{node \in Output : node \in Graph\,\forall\,Graph \in RetainSet\}$.

As mentioned in Section 3.2, the data structure was designed to allow $\mathrm{O}\big(1\big)$ lookup of nodes and edges. Leveraging this fact allows for the Boolean filtering algorithms to be completed in $\mathrm{O}\big(n\big)$ time; for each encountered node from books in the retain set, it can be quickly be discovered if the node exists in the discard set, a match causes the node to be discarded. The same process applies for ensuring words are common across each retained book.

If the user has selected the proportional subtraction option then the nodes in the discard set are not simply removed. Instead, the average proportion of all nodes is calculated for the two sets. Proportion is defined as being the total frequency of the words divided by the total size of the set. The resulting set contains the nodes from the retain set with the proportional frequency of nodes from the discard set subtracted from them. For example, if the node *"Algorithm"* has a proportion of 0.02 in the retain set, and a proportion of 0.005 in the discard set, the output node will have a proportion of 0.015. Nodes that have a lower proportion in the retain set than the discard set are completely removed from the output. Words that are used a proportionally equal amount of times in both sets will not be emphasised in the results, and those which differ will naturally rise in prominence in the results.

During the retain/discard operations, there is also the opportunity to filter nodes with unwanted lexical categories. The `BookCollection` has as set of lexical categories that are to be retained. Each node in the graph has its associated lexical category stored as an integer. The process of filtering nodes for output is straightforward; simply do not pass the node to the output if the lexical category is not in the collection's set. Lookup into this set is $\mathrm{O}\big(1\big)$ therefore it does not affect the overall complexity of the filter algorithm. After passing all filter requirements, `Node` objects are copied from their containing data structure to prevent modification of the underlying graph structure.

With the filtered nodes produced, the final step is to sort the output array either ascending or descending based on frequency. This operation uses the standard Objective-C quicksort and completes in $\mathrm{O}\big(n\,log\,n\big)$ time.

3.5 | Designing Annotation Tools

Allowing the user to annotate their findings from the graph marks the beginning of a process that enables transformation of quantitative data into qualitative insight. Annotation is the task of applying user-generated notes and comments to the underlying data structure, and being able to retrieve and review these notes at the user's convenience.

Annotations allow the user to keep track of interesting points in the data and return to them for further study if necessary. Determining what kinds of data should be open to annotation is a key point that could change based on each individual user's needs. It is therefore imperative that the annotation design covers as broad a use-case as possible while remaining clear and intuitive to the user. Considering the key components of the core graph structure – nodes representing the words of the text and edges representing the relationships in the text – it is logical that these should be fundamental points of the annotation system. Supplementing these annotations is the ability to directly quote a piece of text, and a generate comments that may be attached in a general fashion to a book.

3.5.1 | Technical Design

The major technical requirement for annotations is their capacity for quick retrieval, both manually by the user, and programmatically for interface updates. This requirement dictates that the lookup of a single annotation must be completed in $O(1)$ time; linear lookup would scale poorly if a user has thousands of annotations spread across multiple books.

Annotations are represented through the use of `Annotation` objects. These objects contain the text of the annotation, as well as its type and a hash of the specific collection it is associated with. When an annotation is created by the user, it is stored separately to the graph it corresponds to. The graph structure of any book will be significantly larger than the set of annotations that map to it. It is more efficient to simply load the required annotations as a separate data structure when required, as this does not incur the significant time cost associated with loading the graph. This allows the user to review the notes they have taken immediately when required.

Each annotation is stored centrally within *Links* using the `Library` singleton. This allows for consistent retrieval and editing throughout the tool; edits that are made in one section of the tool will be carried over to all other sections. The annotations are stored within a hash table of linked lists. This design allows for multiple annotations with the same key to be stored.

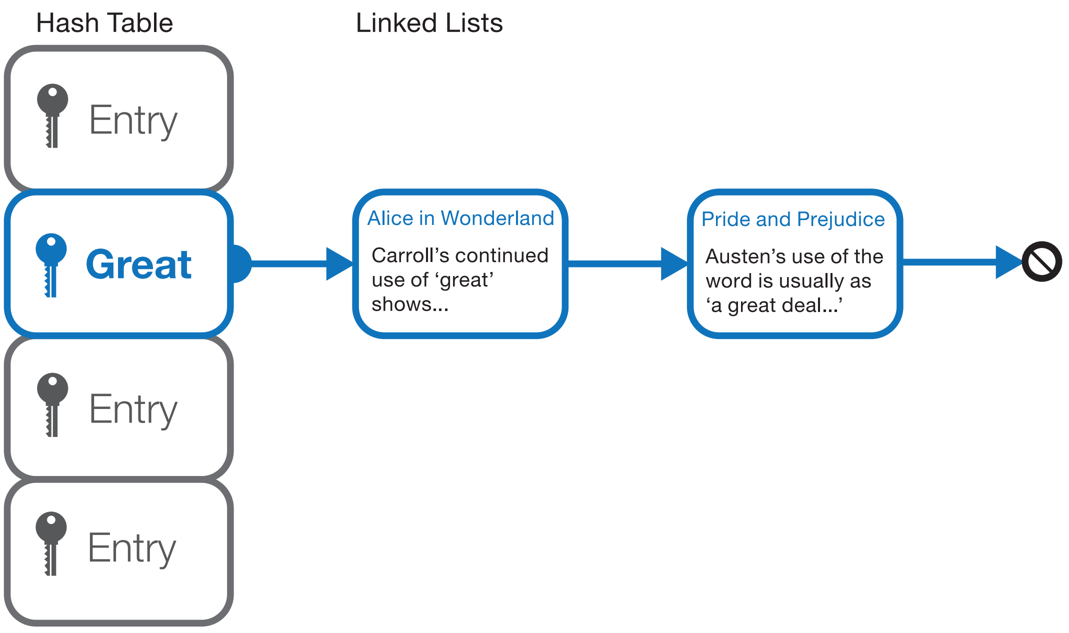

Hash Table
Linked Lists
Entry
Great
Entry
Entry
Alice in Wonderland
Carroll's continued use of 'great' shows...
Pride and Prejudice
Austen's use of the word is usually as 'a great deal...'

Figure 3.5: Overview of the annotation data structure.

In the case of a user annotating a word in the text, the key of the corresponding Node object will be used for insertion into the hash table. Knowing the key ahead of time allows for the tool to retrieve all related annotations by loading the linked list for that key. Related, in this instance, refers to annotations the user has made to the same word but in a different context, such as a separate novel. Showing related annotations allows the user to consider contrasting points between one text and another, and allows them to incorporate these views into new annotations.

Though many annotations may share the same key, each one is separately linked to the book or collection they are associated with. With this separation in place the user may create unique annotations for analysis of single texts, and for analysis involving multiple books, even in the case where collections overlap.

3.5.2 | Templates & Integration with Existing Filters

A template is a user-defined description that is designed to fit broad sections of the data. For example, the user may define a new annotation template *'family'*; they may then associate that tag to words in the graph – sister, brother etc. – using the template annotation. Templates provide two pronounced benefits to the tool:

1. The user gains the ability to tag large swathes of data in a much smaller space of time. Traditional text based annotations may be suitable for retaining knowledge; however, they do not provide an ability to discover new knowledge. When a user tags a piece of the graph they are essentially adding qualitative data to the underlying structure. This data could itself be visualised back to the user, for instance, showing them a graph of occurrences for the tag *'family'* in the text; this opens up the potential for user-assisted thematic analysis in the tool.

2. Annotation templates provide a way to extend the filtering system presently implemented within the tool. The tags created by the user may be incorporated into filters as a way of further restricting nodes based of their associated tags. Restriction in this fashion allows for more concise exploration of the data.

The proposed method for implementing the tagging system is to add a single 64-bit integer (`long long`) to each node. Each tag corresponds to a single bit of the integer, allowing for extremely quick lookup of a tags existence on a node by using simple bitwise operations. There are several advantages of this approach over alternative approaches, such as adding arrays of tags to a node, or using the node's key to lookup into a tag hash table:

- An additional 8 bytes is added to the size each node, this overhead is acceptable in comparison to the overhead used by adding a mutable array: 250 bytes.

- Using a hash table approach would not incur significant memory costs; however, there would be an additional time cost associated with the lookup of each tag.

- With an integer based approach, a node may be confirmed to have multiple required tags in a single operation. For example, given three tags with the bit positions, 0, 1, and 2 it is possible to ensure each of these bits are set by masking the unrelated bits and performing an exclusive-or with the binary number`0b0111`. If the output is non-zero then the node does not include all the required tags for the given filter.

The integer approach has the disadvantage of limiting the user to sixty-four available tags. However, the speed and memory advantages of this approach outweigh the disadvantage of limiting the total available of tags, as users are unlikely to require a larger tag amount.

3.6 | Designing a User Interface

Links is designed for use by students and researchers that are studying humanities subjects. It was imperative that the chosen user interface design make the tool as accessible to as broad an audience as possible as per Gibbs and Owens' recommendations [1]. Assumptions about the ability of the user are kept to a minimum; it is assumed they are comfortable using a touch interface from experience with the default applications that come with an iPad. Actions like scrolling and panning actions are available; similarly, navigation through view controllers using the navigation bar, and opening menu items using navigation bar buttons is available. Other interface options, such as multi-touch gestures, are regarded as being too advanced and are not used within the tool.

A major focus throughout the design of this tool was in enabling high-level interaction with the graph data while maintaining the usability required for use by a broader audience. This section focuses on the design choices made and the trade-offs between usability and capacity for knowledge discovery in the tool.

3.6.1 | Importing Texts

The natural starting point for designing a tool which interacts so deeply with text is to ensure that importing the material to be analysed is as simple as possible. The first barrier to entry for making use of the tool is the process of importing a text. Unlike other applications where users may learn by exploration of the interface, without importing texts there is nothing for the user to explore or learn inside the tool. Therefore it is extremely important that adding books or articles to the user's library is as straightforward as possible.

Upon installation, *Links* registers itself with the operating system as an application that can open plain text files. Plain text files were chosen for both their simplicity in processing and their ubiquity in computing; the majority of applications dealing with text will provide a method to save in `.txt` format. Registration with the operating system allows the user to open text files from any source that provides a file browsing ability, including email applications for opening attachments, cloud storage applications, and web browser downloads. Using standard operating system features is essential for increasing usability as users will both expect, and be comfortable with the feature.

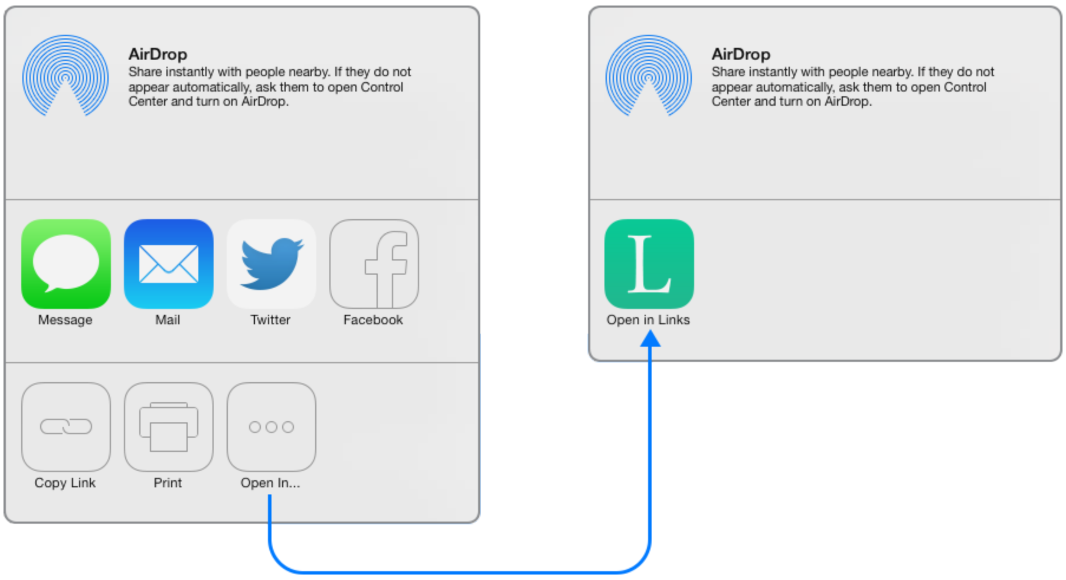

AirDrop
Share instantly with people nearby. If they do not appear automatically, ask them to open Control Center and turn on AirDrop.
Message
Mail
Twitter
Facebook
Copy Link
Print
Open In...
AirDrop
Share instantly with people nearby. If they do not appear automatically, ask them to open Control Center and turn on AirDrop.
Open in Links

Figure 3.6: The process of importing a text into *Links* from an external application.

Additionally, users may simply copy and paste text into the tool. This covers the case where a user encounters an interesting text they wish to analyse, but where they lack access to a text file of the piece. Once the text is copied, the user may paste it into a special area of the tool and continue as if they had loaded a file from disk. The paste action is achieved by extending the options of a `UILabel` to accept paste input. A label allows for the text to be taken from the operating system clipboard, but does not allow the user to accidentally change the content as would be possible in a standard text field. This method also remains consistent with the behaviour of loading a file from disk.

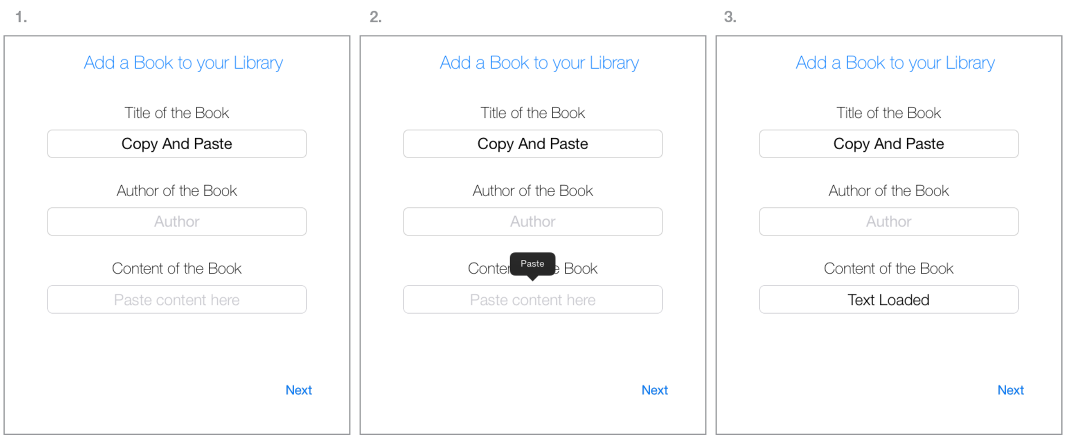
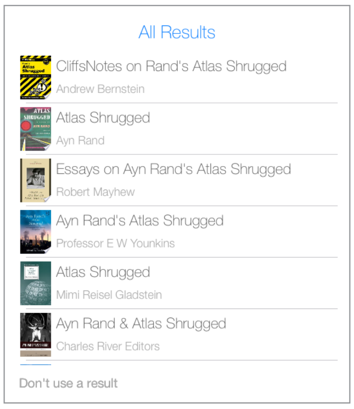


Figure 3.7: The process of adding text from the OS clipboard.

3.6.2 Processing the Text

The first step in processing the text is to acquire an optional title and author for the piece. If the user provides this information, *Links* will connect to the Google Books API [1] to retrieve possible matches. This has several benefits; firstly, it abstracts away the responsibility of adding additional metadata to the book from the user. This data, such as ISBN numbers, genres, publication dates, descriptions, images etc. is desirable to have yet tedious for the user to manually add. Secondly, by adding metadata the user quickly gains the ability for additional filtering options in their library, such as filtering on publication year and genres. Finally, being able to present cover images to the user that were sourced online enhances their ability to quickly find books in the library through recognition of the image rather than recalling and searching for the title.


All Results

CliffsNotes on Rand's Atlas Shrugged
Andrew Bernstein

Atlas Shrugged
Ayn Rand

Essays on Ayn Rand's Atlas Shrugged
Robert Mayhew

Ayn Rand's Atlas Shrugged
Professor E W Younkins

Atlas Shrugged
Mimi Reisel Gladstein

Ayn Rand & Atlas Shrugged
Charles River Editors

Don't use a result

Figure 3.8: The search results seen while adding Ayn Rand's *Atlas Shrugged*.

As previously described in section 3.3, the parsing process is the only component of the tool which requires the user to wait any significant amount of time for an action to complete. Parsing can take several minutes for a large book so there is a need to inform the user about the progress throughout the execution. The research done by Nielsen [22] and Card *et al* [23] on the topic of user interaction with long processing delays shows that ten seconds is the upper limit for keeping a user's attention focused on one dialog. As the parsing time in *Links* is many multiples of that attention span, there was a distinct need to both make the process more interesting, and enable the user to estimate the time remaining. While parsing the text, the user interface is continuously sent updates about the current count for each lexical category as tagged by `NSLinguisticTagger`. This is designed to keep the user

[1]Google Books API: `https://developers.google.com/books/`

interested in the progress as the information presented is both a visually appealing way and provides interesting information about the linguistic content of the book.

A similar process happens during graph construction; the current count of `Node` and `Edge` objects that have been created is displayed on the interface. In conjunction with this visual style, a loading bar also displays the total progress that has been achieved by the parser at that point in time. This allows the user to get a quick overview of the processing time that remains. Estimating the remaining time is possible, but will not be entirely accurate due the way in which the parsing algorithm works; work is not evenly spread throughout the text, for example, certain discarded lexical categories may make up the majority in one section and the minority in another meaning the first section will parse more quickly. Evened out over the size of an entire book, these timing differences between sections are negligible. The process of saving the graph to disk cannot be timed as the archival function of Objective-C does not have a progress callback. During this final part of the algorithm, a simple activity indicator is shown to the user to reassure them that progress is still taking place.

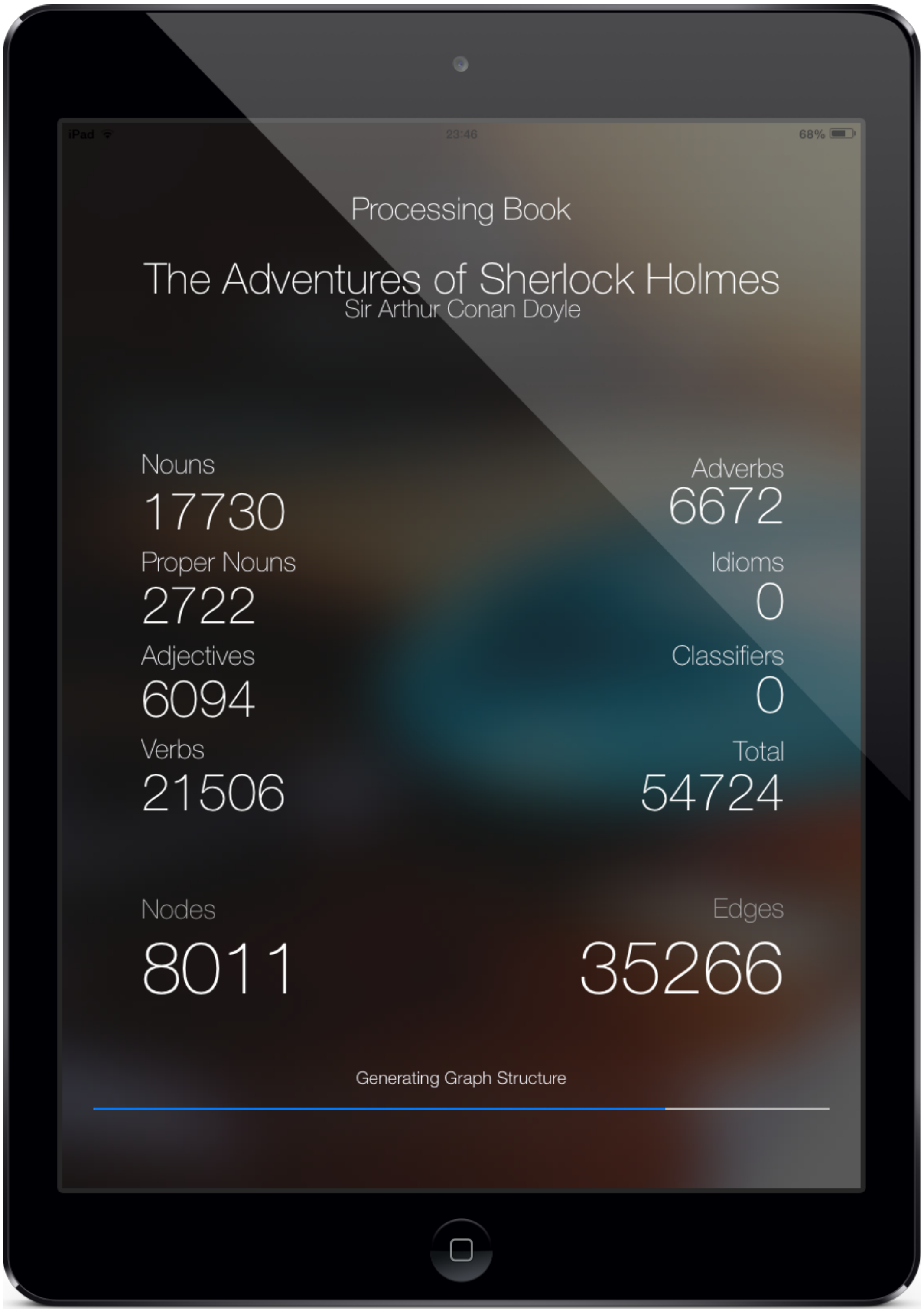


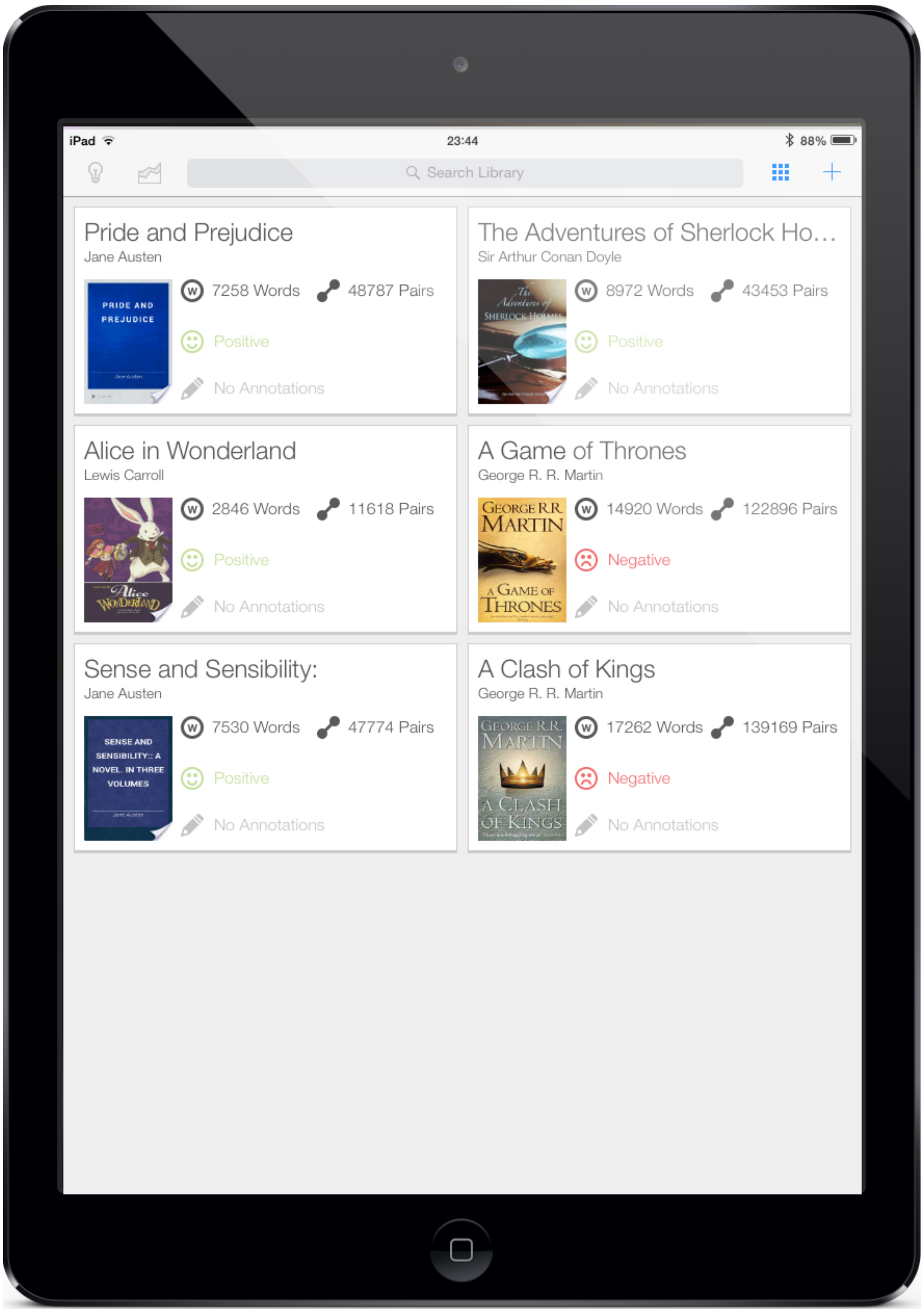
Figure 3.9: The visualisation of the parsing process shown for *The Adventures of Sherlock Holmes.*

3.6.3 | Home Screen

The home screen is the starting point of the tool; it shows the user's library with details about each book it contains, as well as the options to add a new book or progress to different parts of the tool. Each book is listed with its title, author and cover image; details specific to *Links* are also shown here, including the total number of nodes and edges within the graph for the given book (described to the user as words and pairs), the overall sentiment rating as calculated during parse time (Section 3.3.3), and the total number of annotations attached to the book.

The major objectives of this screen were to present the details of a user's library in a clear and concise way, enabling the user to gain a quick overview of the contents of each book. Secondly, the user is presented with the abilities to search for a book through a standard search bar and to open the library filtering settings.

When a user wishes to explore the graphs for a selection of books, they may choose any combination from their library and simply press the analysis button, represented by a light bulb icon. Similarly, if they wish to perform sentiment analysis, they may select a single book and press the sentiment button, represented by a line chart icon.



Pride and Prejudice
Jane Austen
7258 Words 48787 Pairs
Positive
No Annotations

The Adventures of Sherlock Ho…
Sir Arthur Conan Doyle
8972 Words 43453 Pairs
Positive
No Annotations

Alice in Wonderland
Lewis Carroll
2846 Words 11618 Pairs
Positive
No Annotations

A Game of Thrones
George R. R. Martin
14920 Words 122896 Pairs
Negative
No Annotations

Sense and Sensibility:
Jane Austen
7530 Words 47774 Pairs
Positive
No Annotations

A Clash of Kings
George R. R. Martin
17262 Words 139169 Pairs
Negative
No Annotations

Figure 3.10: The home screen of the application showing a user's library.

3.6.4 | Exploration of the Graph Structure

Dadzie *et al* succinctly describe benefits of visualisation within their study [17]:

> The power and value of visualisation is seen in its ability to foster insight into and improve understanding of data, therefore enabling intuitive, effective knowledge discovery and analytical activity. This is achieved by removing the cognitive load encountered in managing the large amounts of complex, distributed, heterogeneous data common in today's technology and information-rich society, and relying instead on advanced human perceptual ability.

Knowledge discovery is the central motivation behind *Links*. To enable this process to take place, the visualisations of the graph structure needs to adhere to several principles as described by Dadzie:

1. The visualisation must have the ability to generate an overview of the underlying data.

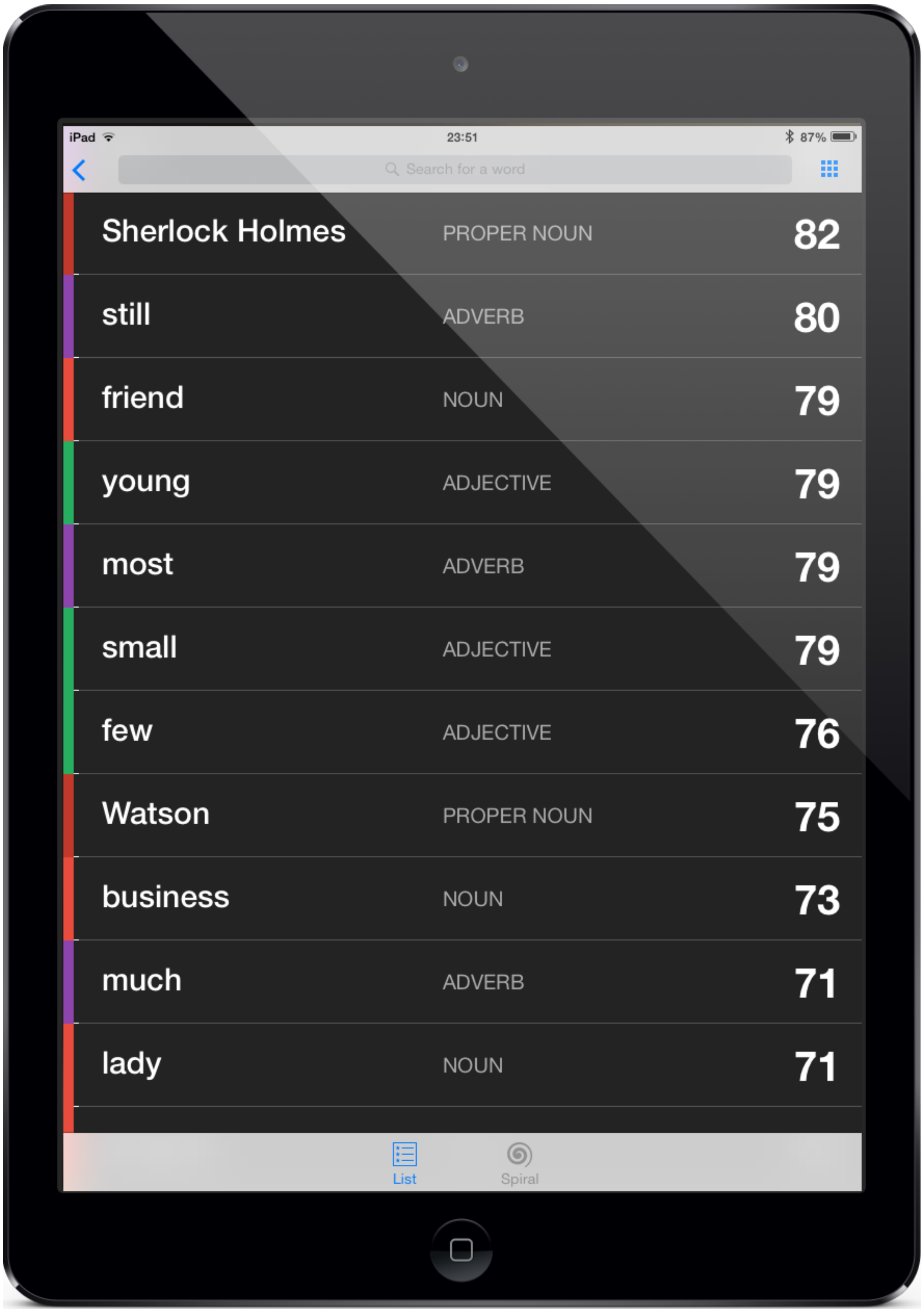
2. It must support filtering to remove less important data.

3. Regions of interest must provide a higher detail than the overview.

Additionally, the graph must remain navigable by the user; this is the process that enables movement along edges from node to node, simply displaying the graph as a static image does not lend itself to discovery. Furthermore, the performance of navigating the graph must be acceptable by the standards of the user; in *Links* acceptable performance is defined as requiring any navigation or filter action to be completed within the order of milliseconds. Longer waiting times risk frustrating the user and hindering the usefulness of the tool. To meet these principles, the visualisations *Links* needed to overcome a number of issues:

Graphs are an inherently difficult data structure to visually represent; as the number of nodes increases linearly in a graph, the number of edges increases exponentially, assuming there is an edge between each node. Each node in the graphs constructed by *Links* will have a minimum of four edges due to the adjacency calculation (Section 3.3.4); in reality each node will likely have a much higher number edges as the number increases linearly with each appearance of the word (Figure 3.3). Visualising all nodes and all edges in a single view may cause the user to experience an 'information overload'; by presenting so much information, the user loses the ability to see the significance of the data being shown. The visualisation methods in *Links* were chosen to highlight information so that it retains relevance to the rest of the graph and remains consumable by the user. Other implementations of graph visualisation, such as TopicNets, rely on providing multiple levels of detail and colour coding to abstract away large amounts of data into manageable sections for the user.

The visualisation of word frequency is the first view the user encounters upon entering the graph interaction function of the tool. The purpose of this view is to allow the user to find a root word from which to begin their traversal of the graph. The view presents every node in descending order as this works off the theory that important words with appear with a higher frequency than others. Building upon the recommendation made by Dadzie *et al*, and shown in TopicNets [10], *Links* colour codes each node by its lexical category to aid for quicker recognition by the user when they are browsing words.



Sherlock Holmes	PROPER NOUN	82
still	ADVERB	80
friend	NOUN	79
young	ADJECTIVE	79
most	ADVERB	79
small	ADJECTIVE	79
few	ADJECTIVE	76
Watson	PROPER NOUN	75
business	NOUN	73
much	ADVERB	71
lady	NOUN	71

Figure 3.11: List visualisation of frequency in *Sherlock Holmes*. Note the colour coded lexical categories.

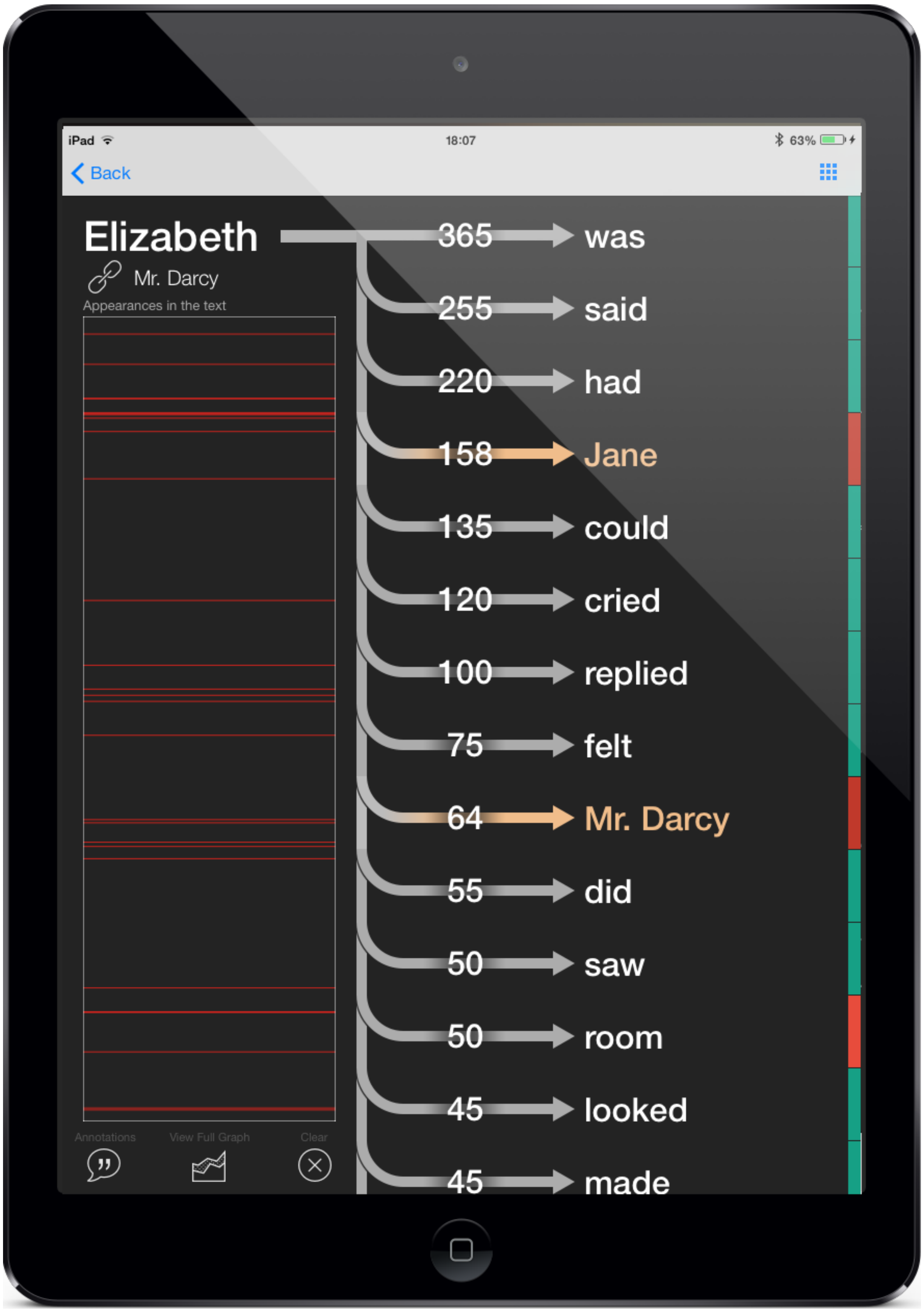
Tree View

When the user selects a word from the frequency view, the tool enters the graph. *Links* expands upon the approach taken by IBM's Word Trees. Presenting the edges from one node in an isolated fashion gives the user a very specific region of interest to examine as recommended by Dadzie *et al.* This design is implemented as the primary means of exploring the graph within the tool. The user begins by picking a root word, as with the Word Tree approach; at this point they are presented with a list of edges to that node, as well as the weighting applied to each of these edges. Edges are displayed in descending order so that potentially important relationships in the text are naturally displayed at the top of the list.

Users may navigate along edges by simply tapping on any of the linked words. This is implemented as part of a navigation view controller, therefore the user gets the benefit of being able to backtrack through their exploration to an earlier point in the graph. Pairs that have been previously annotated by the user are highlighted with a yellow colour to emphasise their importance to the user; this serves to remind the user that they have annotated the link.

Providing an overview of the data is one of the requirements from Dadzie *et al*; when viewing a specific node, *Links* shows an overview graphic of where that word appears in the text. From the overview, the user can gain qualitative information extremely quickly; for example, a user may notice that a character ceases to appear after a certain point in the novel. Tapping on the overview will bring the user to the corresponding location of the text where that word appeared; here they may draw their own conclusions about why that character stopped appearing.

Similarly, tapping on an edge will cause the overview to change to displaying the locations of that relationship in the text. This is especially useful for keeping track of the relationships between two characters, as the distribution provides hints about how that relationship changes throughout simply by the change in frequency. Comparing this approach to the Phrase Net approach (Figure 2.1); one can see that Phrase Net does not benefit from the overview functionality such as that provided by *Links*, as the user is given no indication of where exactly the pairs are occurring in the text. Furthermore, the Phrase Net implementation fails to provide a method for the user to easily find the quote, instead opting to only show a small extract of the sentence it is contained in. From a knowledge discovery standpoint, the Phrase Net approach hinders the user's ability to truly learn from the visualisation. The approach taken in *Links* is designed to overcome this shortfall.



365	was
255	said
220	had
158	Jane
135	could
120	cried
100	replied
75	felt
64	Mr. Darcy
55	did
50	saw
50	room
45	looked
45	made

Figure 3.12: Tree visualisation of a node. The colour coded pairs – Jane and Mr Darcy – indicate an annotation has been made for those links. The red lines provide an overview for where Elizabeth and Mr Darcy appear together in *Pride and Prejudice*

Spiral graph

Experimentation was done to determine if it was possible to create a visualisation technique that would maintain the exploratory nature of the tree view, while visualising frequency and pairs through size and colour respectively. The spiral graph was the result of this experiment. The design was inspired by the word cloud visualisations mentioned in Section 2.3.

Algorithm 3: The spiral graph layout process.

```
Theta = 0
Rotations = 1
LastPoint = 0
CenterX = 0
CenterY = 0
Place First Node At (CenterX, CenterY)
MaxFrequency = First Node Frequency
// The distance between the current spiral rotation level and the following level.
MaxDistance = (Second Node Frequency / MaxFrequency) * WIDTH / 2
// The distance between the current node and the next node.
BufferDistance = (Second Node Frequency / MaxFrequency) * WIDTH
foreach Remaining Nodes do
    CurrentPointWidth = Node Frequency / Max Frequency * WIDTH
    Theta += (LastPoint/2 + CurrentPointWidth/2) / ( (MaxDistance + BufferDistance *
    (Theta mod 1) ) * 2 * $Pi$)
    // Check if we have performed a full rotation and update max distance and buffer
       distance.
    if Theta >= Rotations then
        Rotations++
        MaxDistance += BufferDistance
        BufferDistance = CurrentPointWidth
    end
    LastPoint = CurrentPointWidth
    x = CenterX + (MaxDistance + BufferDistance * (Theta mod 1) ) * $cos$(2 * $Pi$ * Theta);
    y = CenterY + (MaxDistance + BufferDistance * (Theta mod 1) ) * $sin$(2 * $Pi$ * Theta);
    Place Node At (x, y)
end
```

The aim of the spiral graph is to the visually represent the frequency of each node through physical size as in a word cloud, whereby more frequently occurring nodes appear larger on the screen. This appearance is coupled with several other requirements. Firstly, the layout of the graph must be ordered; unlike a word cloud, the spiral graph strictly places nodes along a spiral shape. This has the advantage of making it easier for the user to explore the graph; high frequency words appear at the centre while and gradually decrease as the graph expands. The spiral also ensures that nodes do not overlap during layout. Secondly, the graph must have the ability to show the edges between nodes. Drawing all the edges on such a large scale graph causes too much visual noise; there is no way to draw 100,000 edges on a small two-dimensional screen and have the result remain comprehensible. To avoid this issue, edges are represented by colour rather than lines that physically link nodes. To display edges, a user selects a node; this causes the related nodes to be highlighted.

The spiral visualisation is built on top of a scroll-and-pan view. This is the same technology that is commonly used in mapping applications. The view allows the user to explore the graph by panning from node to node and zooming in on the smaller nodes at the fringes.

The spiral graph suffers from sluggish performance due to the large number of nodes it must display. The number of nodes drawn to the screen is capped at five hundred to prevent the visualisation becoming too slow to be useful. The common approach to visualisation on touch devices is to have each individual piece of data placed into a view cell. These cells are reused as the user scrolls through the data, meaning that only a small subset of the larger data set is ever drawn at any one time. However, because of the zoom functionality of the spiral, it is possible that a user may have all the data displayed at once, therefore requiring an individual cell for each piece of data. This is highly taxing on the system and causes noticeable slowdown. Limiting the distance that a user can zoom out, therefore limiting the amount of data on the screen, could help alleviate this problem. However, this limits the effectiveness of the visualisation, as users will no longer be able to zoom out to view all the highlighted pairs at any one time. In its current form, the spiral graph does not perform as well as the tree view for the purposes of knowledge discovery, though additional work could be productive in improving its practicality by focusing on speed improvements and additional exploration options, such as viewing edge locations.

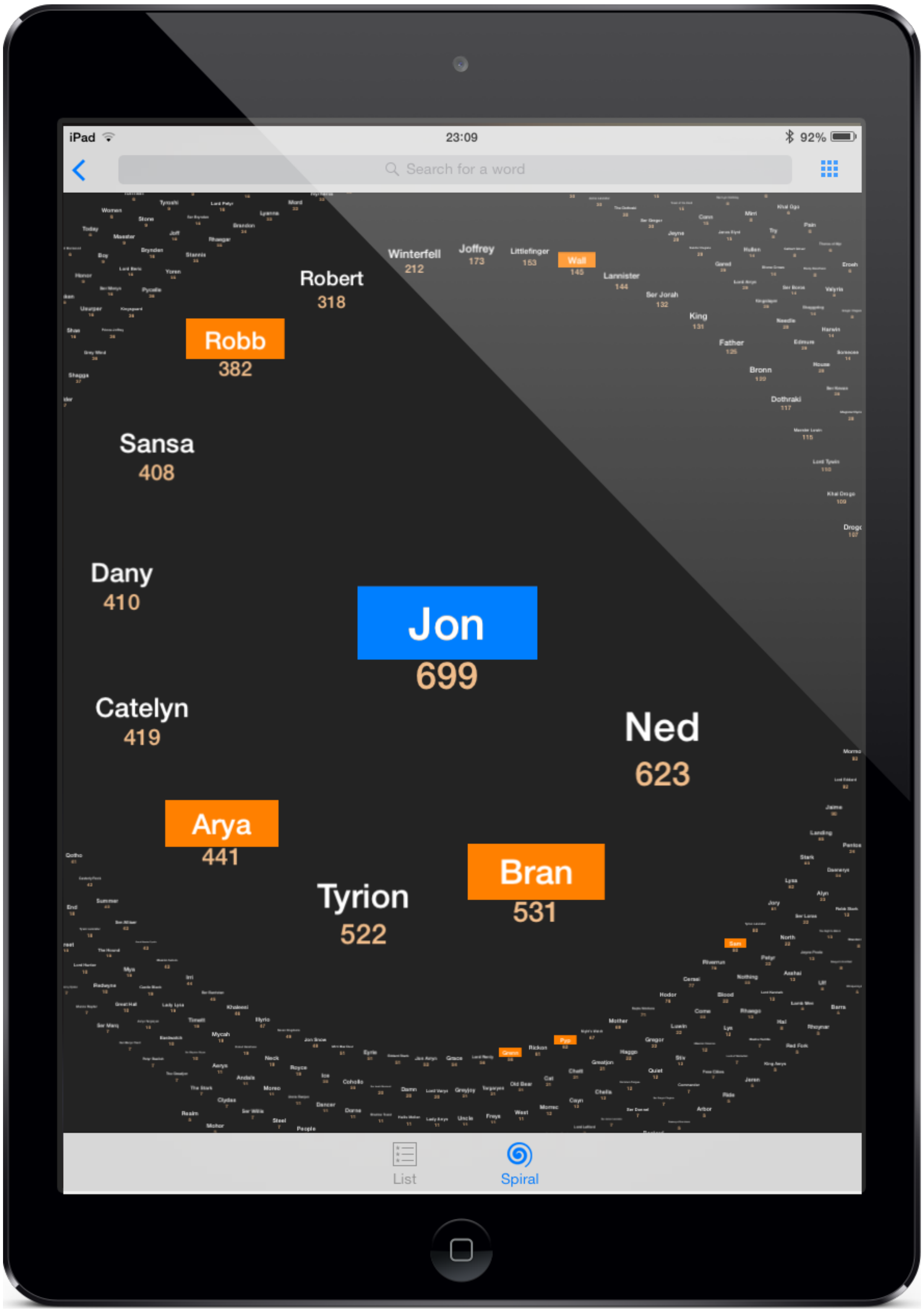


Figure 3.13: Spiral graph of characters in *A Game of Thrones*. The words highlighted orange indicate the presence of a close relationship with Jon

3.6.5 | Additional Interface Components

Sentiment Chart

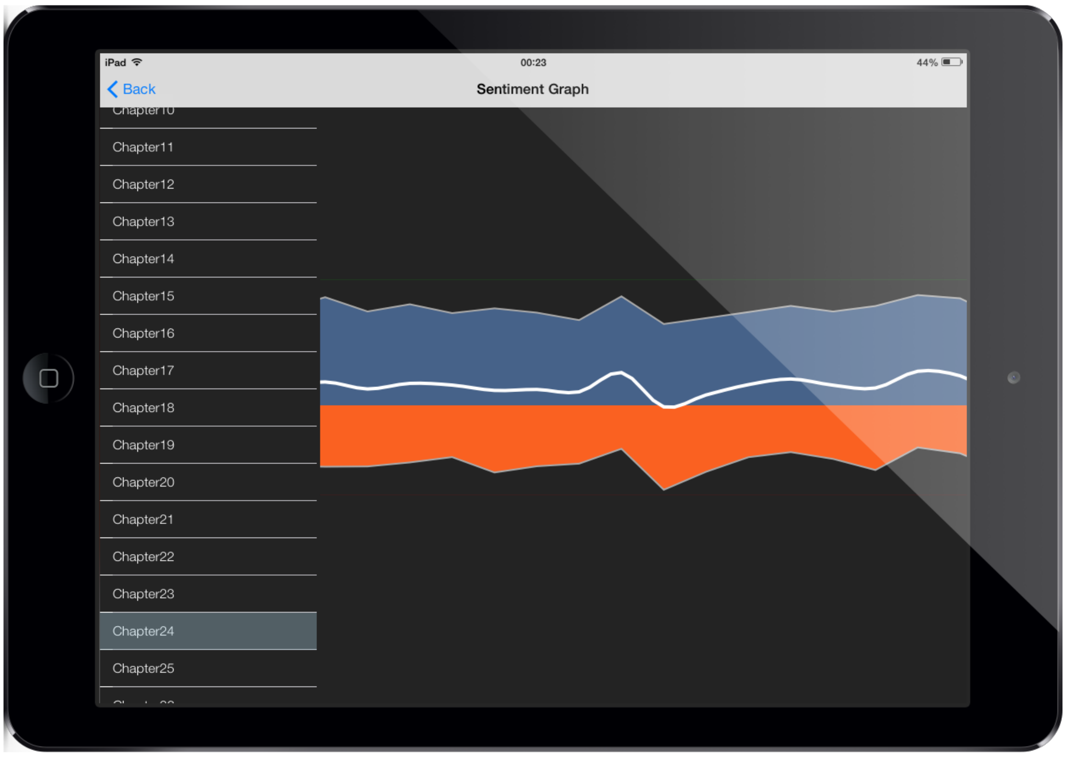
A user may view an overview of a text's sentiment by selecting the option from the home menu. The sentiment of a text is presented to the user using a traditional chart. The Y-axis of the chart maps sentiment from +1 to -1 and the X-axis represents the location in the text. Experimentation was done with an approach that only displayed the average sentiment, that is the average of both positive and negative sentiments combined. The result was that both polarities would essentially cancel each other out. This has the undesired effect of presenting a section that is completely neutral, i.e. a polarity of 0, in the same way as a section that is complete polarising, i.e. the positive and negative are both roughly equal. By displaying polarising sections as neutral, the user loses the ability to identify them as being potentially important.

To avoid this issue, two additional sets of data points are added to the chart to represent the positive polarity and negative polarity separately. This allows the user to quickly locate areas of extreme contrast, while retaining the ability to view the overall average. A similar approach is taken by etcML for visualising sentiment of social media sets; this visualisation uses an interlocking design with both polarities being drawn using the entire Y-axis (Figure 2.2). *Links* differs by dividing the Y-axis evenly between the two polarities.

The lines are drawn using a Catmull-Rom spline instead of using Bezier splines. Catmull-Rom splines provide the advantage of producing a line that curves *through* a set of control points whereas Bezier splines produce lines around a set of control points; this latter is not ideal for accurate representation of data. The implementation for the Catmull-Rom approach in Objective-C was based on work done by Erica Sadun [24].

Chapter markers are placed throughout the chart to aid the user in determining the exact location the chart represents at any given point. A list of chapters is displayed alongside the chart with the current chapter being highlighted to also aid the user in this respect. Users may quickly navigate the chart by selecting chapters from this list.

Tapping at any point in the chart will cause the tool to open the book at the related section. Words from this corresponding section are highlighted green or red based on their associated sentiment. The intensity of the background colour is based off the polarity rating for that word; very positive polarity produces a solid green, very negative polarity produces a solid red, and neutral words remain white.



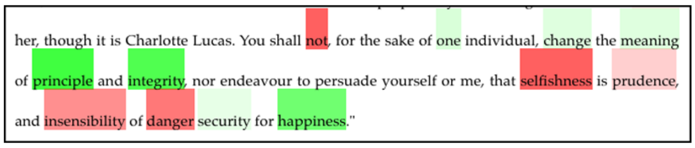

her, though it is Charlotte Lucas. You shall not, for the sake of one individual, change the meaning of principle and integrity, nor endeavour to persuade yourself or me, that selfishness is prudence, and insensibility of danger security for happiness."

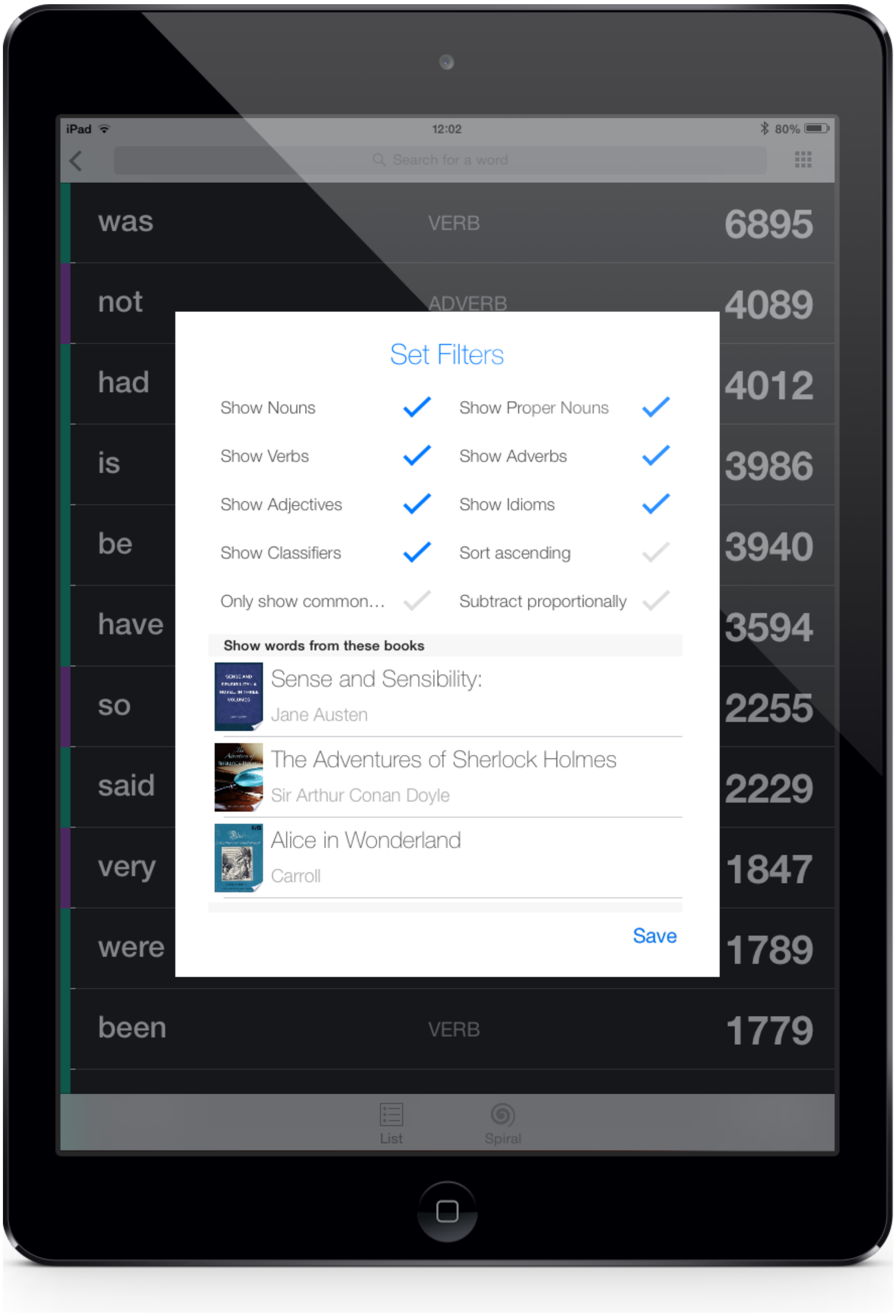
Figure 3.14: A sentiment chart for *Pride and Prejudice* showing a drop in average sentiment accompanied with a quote from that point.

Application of Filters

The addition of data filtering functionality is the final recommendation as described by Dadzie *et al.* Filters are accessible throughout each of the graph visualisations: frequency, word tree, and spiral. The behaviour of the filters remains consistent across each of these views; users select the filter option and a modal overlay appears containing the filters.

These filters present the option to restrict the lexical categories that are currently been shown. Options are presented as simple check boxes in a collection view; this has the advantage that should the need to expand these filter options become apparent, they may simply be added with no redesign of the interface. Lexical filtering provides the user with the ability to refine the presented data to fit the information they are seeking; for example, if the user wishes to view character relationships they may filter by proper nouns, thus leaving only characters and locations in the visualisation.

In addition to lexical filtering, the option to compare multiple books is presented. Each book in the user's current analysis collection is presented in a list. This list contains two sections: a retain section and a discard section. Users may drag and drop books from one section to the other simply by pressing and holding. This type of interaction is slightly more advanced than other interface techniques, however, as it is a core part of the iOS menu system – holding menu icons to rearrange them — it was believed that users would be comfortable performing such a task. After creating separate retain and discard sets, the user may select the proportional subtraction option to perform the operation described in Section 3.4.2. Similarly, they may select the common words option to perform a Boolean *and* operation over each book in the retain set as described in the same section.



was	VERB	6895
not	ADVERB	4089
had		4012
is		3986
be		3940
have		3594
so		2255
said		2229
very		1847
were		1789
been	VERB	1779

Set Filters

Show Nouns	✓	Show Proper Nouns	✓
Show Verbs	✓	Show Adverbs	✓
Show Adjectives	✓	Show Idioms	✓
Show Classifiers	✓	Sort ascending	
Only show common…		Subtract proportionally	

Show words from these books

Sense and Sensibility:
Jane Austen

The Adventures of Sherlock Holmes
Sir Arthur Conan Doyle

Alice in Wonderland
Carroll

Save

Figure 3.15: The filters modal view showing lexical category options and multiple books in the user's collection.

Addition of Annotations

The annotation functionality is the final interface addition to *Links*. As with filters, the annotation input functionality is contained in a modal view that remains consistent across each view controller that implements the feature. The current implementation of *Links* allows annotations to be applied to relationships, words, and as a book comments. Pressing the annotation button will present the user with an input field to enter text as well as a list of related annotations. Each annotation in the list is marked with its type, the data it refers to, and the book to which it is attached. Users are not required to enter an annotation at this point; they may simply browse previous annotations.

A related annotation is an annotation that the user has made to the same data point over their entire library. For example, if the user is analysing a series of books, they may have annotated a character in one book; when they are analysing subsequent books in the series, the annotations they have made in the first book will also be shown. Once an annotation is added, the corresponding data point becomes highlighted in the visualisation so that the user may easily find it again.

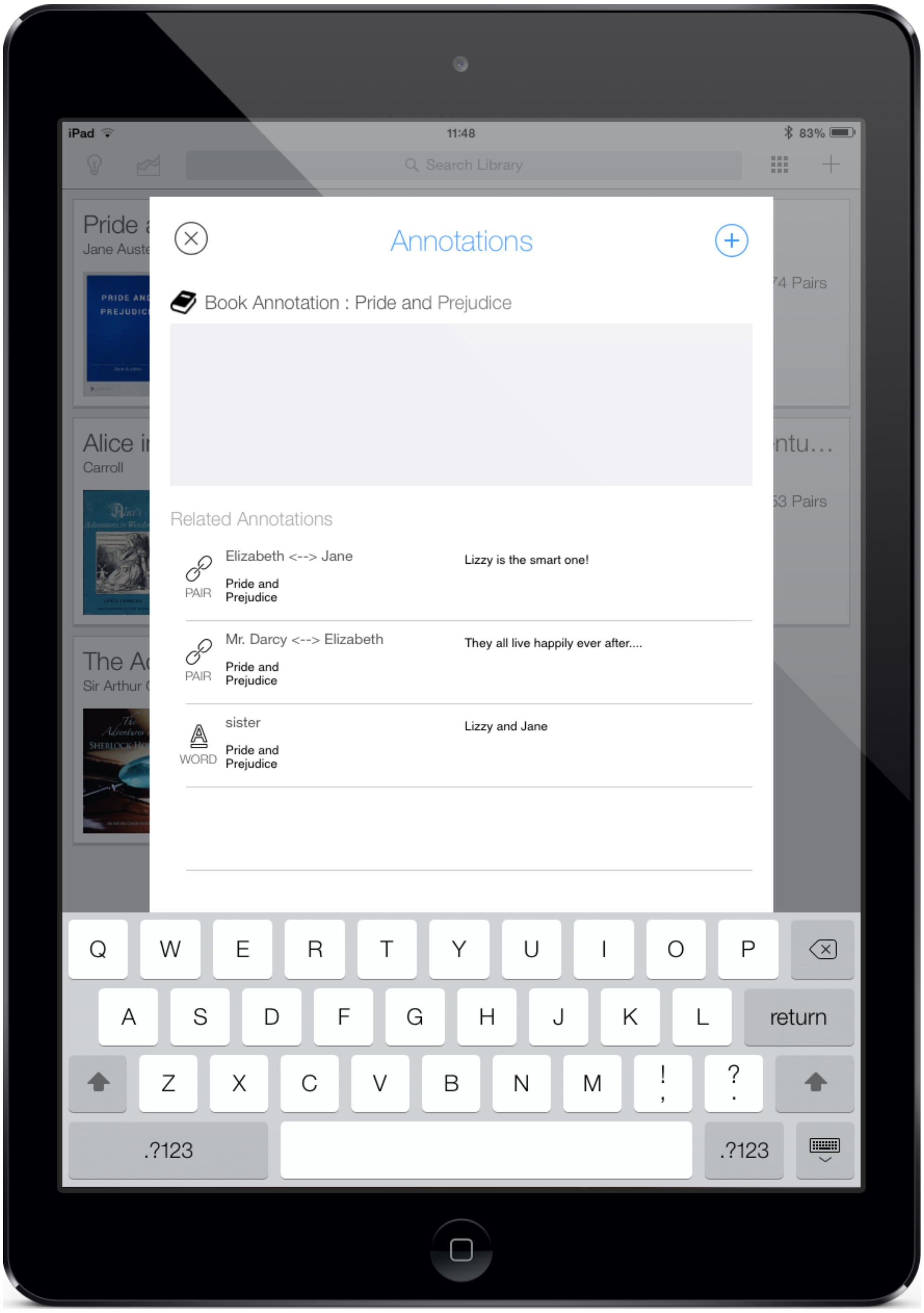


Figure 3.16: The annotation modal view showing annotations related to *Pride and Prejudice.*

4 | Evaluation

This chapter begins with a technical evaluation of the current implementation of *Links*. The evaluation commences with the topic of part-of-speech tagging, specifically `NSLinguisticTagger`, its performance, and its merits within an academic research project of this type. The underlying graph data structure is then discussed. The memory overhead is evaluated and potential improvements to its implementation are suggested. A qualitative evaluation of the graph construction algorithm is then given, with a specific focus on the quality of the calculated relationships and their usefulness to the user. The focus then moves to knowledge discovery within *Links*, with the discussion centring on how well this was achieved through the use of word relationships, location overview, and sentiment analysis. The chapter concludes its evaluation with a discussion on the merits of the interface design choices taken by *Links* in comparison to those taken by similar tools.

4.1 | Technical Evaluation

4.1.1 | `NSLinguisticTagger` Evaluation

As a central technical component of the tool on which the quality of the output heavily relies, it is imperative that the part-of-speech tagger performs to acceptable levels. In this context, performance refers to the accuracy of the tagger rather than speed or memory overheads.

High quality output from the tagger allows the graph construction algorithm to more accurately map relationships throughout the text, as characters, locations, et cetera are likely to be correctly attributed to their proper lexical category rather than erroneously split across multiple categories. As part of the technical evaluation of *Links*, the tagging accuracy of `NSLinguisticTagger` was measured over the Brown Corpus.

The Brown Corpus is a large selection of texts sourced from multiple different topics and writing styles that was compiled in the 1960s at Brown University [25]. Each of the texts within the corpus was manually tagged by linguists; therefore, due to its accuracy it has become the de facto standard is measuring the performance of part-of-speech taggers.

To evaluate the accuracy of `NSLinguisticTagger`, its output was tested against the known input set provided by the Brown Corpus' fiction category. It was believed that this category would provide the closest match to the type of content that users would be loading and would therefore provide the most accurate set of results.

The test made use of the open source Natural Language Toolkit (NLTK)[I], as it provides the ability to count the number of appearances for each lexical category in any of the Brown Corpus' text categories. The same set of texts was also processed by the `NSLinguistic-Tagger` and the counts for each lexical category were taken. The two sets of results are shown in Table 4.1.

Tag Source	**Nouns**	**Proper Nouns**	**Verbs**	**Adjectives**	**Adverbs**	**Total**
Links	11063	2618	12067	3382	3387	32517
Brown	10881	2409	11687	3226	3185	31388
Difference	182	209	380	156	202	1129
Classification Error	1.67%	8.67%	3.25%	4.83%	6.34%	3.59

Table 4.1: Classification error of `NSLinguisticTagger`.

On average, `NSLinguisticTagger` differs from the Brown Corpus by 3.5%. This difference is clearly quite low and represents extremely accurate classification; however, as described in Section 2.2.1, there is a trade-off between the quality of the output and the overall accuracy based on the number of lexical categories that are being analysed. The number of categories analysed by `NSLinguisticTagger` is much lower than the Brown Corpus. This is the reason why the results of `NSLinguisticTagger` have a higher count on average per category, that is, more lexical categories are being merged into a single representative category. For example, the word *'not'* is treated as an adverb by `NSLinguisticTagger` but

[I]NLTK: `http://www.nltk.org/`

as a special negation category in the Brown Corpus. These results may seem to indicate that the accuracy is very high but they do not indicate the overall quality. The lexical categories used by `NSLinguisticTagger` are quite broad and do not allow for extremely in-depth analysis, such a examining verbs by their tenses.

Proper noun detection differs by roughly 9%. Characters and locations form the fundamental relationships in literature and this is the type of data that users will analyse using the tool. Correct classification of proper nouns is therefore prioritised as being more important than other lexical categories, as they have a higher bearing on the quality of the output. A 91% accuracy rate represents a strong framework to build the tool upon and justifies the choice of `NSLinguisticTagger` as the natural language processing component of *Links*.

Despite its high accuracy rate, the closed source nature of `NSLinguisticTagger` prevents further improvement or alteration to be made to it by the academic community. This presents difficulties should additional functionality be required at a later stage of the development of *Links*. Open source alternatives are available such as the previously mentioned NLTK library. The majority of these libraries are written for Java or Python, and would therefore be unable to run in the iOS environment required by *Links*. C or C++ based implementations may be possible to incorporate within the tool and could help decouple *Links* from its reliance on `NSLinguisticTagger`. When making the choice about which tagging frameworks to use, consideration should be given to the inherent speed benefits that `NSLinguisticTagger` gains by being provided with lower level access to the operating system than alternatives. Experimentation may be carried out to determine if the benefits of an open source implementation would outweigh both the accuracy and speed of the current implementation.

4.1.2 | Evaluation of the Graph Data Structure

Due to the limited processing power of iOS devices, speed requirements were heavily emphasised during the development and implementation of *Links*, at the expense of a higher memory overhead. Section 3.2 describes how the choice of in-memory data structure representation was chosen over an on-disk approach.

The chosen method addressed the speed difficulties that were presented by the on-disk approach; however, this approach still presents memory usage challenges where large amounts of books are involved, due to how archival works in Objective-C. The `NSKeyed-Archiver` function, which is used to save an in-memory graph to disk, requires an entire copy of the serialised data to be present in memory before the flush to disk may occur. This is due to how object relationships are tracked by the function and it naturally contributes to a high associated memory cost. The memory cost of both approaches is shown in Table 3.1. The trade-off of larger memory requirements for access and filtering speed allows the user to quickly explore the data that has been generated rather than require them to periodically wait. This implementation lends itself to a more 'hands-on' user experience than what may be offered by alternative on-disk approaches.

Despite the memory challenges, this approach works adequately for small-scale analysis of less than ten books, and demonstrates that a graph data structure has benefits for use within the tool. It does introduce the drawback of requiring the user to wait when initially loading the tool, as entire object graphs must be loaded from disk before any analysis can occur. Moving forward with the implementation – taking the input sizes towards handing an entire corpus – will necessitate the reconstruction of the data model to handle this requirement. A possible alternative to the current design would be to use an on-disk graph database, such as ArangoDB[II]. Databases of this type are specifically designed with graphs as the first choice representational model rather than the traditional relational model found in Core Data; they therefore present performance gains where a graph data model is required for data representation. Such a model may meet the speed requirements needed by *Links* while also allowing the implementation to leverage an on-disk data store, eliminating the memory issues caused by the current approach.

4.2 | Qualitative Evaluation

The area of literature analysis is subjective based on the views of the reader. Similarly, the quality of the output produced by *Links* is subjective to each individual user; data that is useful to one user may not meet the needs of another. This section will therefore evaluate the quality of the data output based on its performance with known texts. It will also evaluate the knowledge discovery potential of *Links* in comparison with similar available tools.

[II]ArangoDB: `https://www.arangodb.org/`

4.2.1 | Proximity as a Measure of Relationship

Section 3.3.4 outlines the technical design for using the proximity of words in a book to indicate the potential strength of relationships. It is this process that creates the data which allows the user to browse strongly linked words and see exactly where relationships occur within the text. The following examples are based on the data generated from processing Jane Austen's *Pride and Prejudice* [26].

Character relationships, and indeed relationships between all proper nouns, are a special case of the graph construction algorithm as they are given a much larger proximity limit than other lexical categories. This decision was made to strengthen the output by emphasising character relationships to the user. Taking the example of Elizabeth, the main character of *Pride and Prejudice*, her strongest relationships are calculated to be Jane, her sister and best friend; Mr Darcy, her eventual lover; Lydia, her younger sister; and Mr Collins, her suitor. Essentially, the relationships presented through the data for this novel accurately map those which one may reasonably expect a reader to conclude as being the most important from their own reading.

Pride and Prejudice has a central main character in Elizabeth, so the relationships between her and the supporting characters tend to be quite pronounced. *A Game of Thrones* [27] is a fantasy epic by George R.R. Martin in which there is no central main character; instead, the novel is told from the viewpoint of many different characters with each chapter being dedicated to a specific character. One may expect that because the book is fragmented in this way that the relationships between characters would not register as strongly as those shown in *Pride and Prejudice*. However, despite the fragmentation, character relationships are still pronounced in the output. Interestingly, each character's story arc occurs at distinct geographic locations. These locations are also apparent in the results generated by *Links*, more so than *Pride and Prejudice*.

The results shown by testing over several novels show that proximity of words within a text, and how frequently those words appear together, is an accurate indicator of how important those relationships are to the text. The relative noise in the data, or relationships that are not important, are kept to a minimum due to their inherent lack of frequency within a given work. Where this approach does not perform as well is in books where characters go by multiple names; full titles, common names, nick names etc. This has the effect of diluting the relationships so they do not appear as significantly in the results. Overall, for the known texts that were tested, the proximity approach shows a higher quality relationship calculation than the pattern-based approach taken by Phrase Net [3], as compared in Section 3.3.4.

4.2.2 | Knowledge Discovery Aspects of the Tool

Presenting strong relationships is the first step in enabling knowledge discovery within the tool. The goal of knowledge discovery is to provide the user with the ability to gain insight into the text from sources that they may not have considered before, as well as providing evidence to support the user's own views about the text. There are several avenues for this kind of discovery presented by *Links*.

Firstly, relationship weighting allows the user to see which exact relationships have been calculated as significant to the text. These relationships regularly match what the user's own expectations are, as described in the previous section. However, there is the possibility that the user may encounter relationships whose weighting are higher than expected. By identifying why the algorithm has calculated the weight in such a way, the user may gain additional insight from sections they would have previously have skipped. This kind of exploration and identification is especially useful among minor characters whose relationships to one another may not be as clear to the user from a single reading.

Secondly, providing a location overview of both word appearances and relationship appearances in the text allows the user to quickly identify significant occurrences and find relevant passages for quotations. An example of this usage, from *A Game of Thrones*, can be seen when examining the relationship between a major character, Jon, and his brother Robb. As previously mentioned, this novel dedicates different character viewpoints per chapter; this has the effect of *banding* the appearances, that is, relationships for a given character naturally appear most frequently within chapters expressed from their own viewpoint. This provides an interesting approach to utilising the location overview. Occurrences of a relationship *outside* the banded areas provide noteworthy results. In the case of Jon and Robb, the results provide anecdotes from other characters about their relationship, specifically their childhood together; the user may not have considered these opinions in their analysis and so this could provide an extra dimension to their study.

Finally, sentiment analysis allows the user to quickly identify sections of the book which are highly polarising. All of the books tested tended towards neutrality, that is, they will either be slightly positive or slightly negative on average, but rarely differing throughout. In general, the *'bag-of-words'* approach to sentiment analysis described in Section 2.2.4 does not allow for precise identification of particular passages, as the context of each word highly affects its intended sentiment. For example, negation is not handled; *'He was not happy'* would register as positive sentence due to the positive rating given to 'happy'. Despite the shortcoming of this approach, the data produced still creates usable sentiment data on a large scale, such as identifying where an entire chapter that has a slightly lower sentiment than the average. An example of this usage can be seen in *Pride and Prejudice.* Roughly half way through chapter twenty-four there is a noticeable drop in sentiment. This corresponds to a section where Elizabeth has become disillusioned with the world as is arguing with her sister, Jane:

> *My dear Jane, Mr. Collins is a conceited, pompous, narrow-minded, silly man; you know he is, as well as I do; and you must feel, as well as I do, that the woman who married him cannot have a proper way of thinking. You shall not defend her, though it is Charlotte Lucas. You shall not, for the sake of one individual, change the meaning of principle and integrity, nor endeavour to persuade yourself or me, that selfishness is prudence, and insensibility of danger security for happiness.*

The sentiment chart performs its role in knowledge discovery by allowing the user to find sections such as that described, simply through investigation of the chart, and without needing previous knowledge of the text. Additional use of context would improve its overall performance in this respect.

4.2.3 | Word Frequency as a Measure of Writing Style

Word frequency can be used to identify writing styles or themes in a text [13]. The use of certain adjectives and adverbs can especially highlight the authors intended sentiment in a book. For example, in Lewis Carroll's *Alice in Wonderland*, the most used adjectives include large, great, curious, and mad. This usage reflects the light-hearted children's story that Carroll has created. Such data is easily available to the user in *Links* through use of lexical filters.

Comparison of multiple books as described in Section 3.4.2 also produces interesting data in this regard. The user can quickly identify unique vocabulary usage amongst multiple authors. Comparing multiple books by the same author presents another opportunity for analysis. An example of this usage can be seen when directly comparing a novel and its sequel; the user can leverage the proportional subtraction feature, also described in Section 3.4.2, to identify how an author's vocabulary has expanded from the first novel to the second. With the *Game of Thrones* novella, Martin's usage of the word *'said'* drops almost 25% from the first book to the second, potentially indicating that he expanded his vocabulary in the area of character speech.

4.2.4 | Evaluation of the Interface Design

The design of *Links* is intended to overcome the knowledge discovery shortfalls and user interaction difficulties that are apparent in other text visualisation and analysis tools by adhering to the set of design requirements laid out by Dadzie *et al* [17] (Section 3.6.4). The design process explicitly focused on providing additional ways for the user to access and interact with the underlying data rather than follow the static visualisation approach taken by Phrase Net.

Firstly, the visualisation of relationships in *Links* – using the word tree approach – performs significantly better for knowledge discovery than the single-view based approach taken by Phrase Net (Figure 2.1). The Phrase Net approach limits the onscreen data to only fifty words. This is not enough information to accurately visualise all the important relationships within a text. The approach taken in *Links* allows for an unlimited amount of relationships to be visualised, albeit it only shows the relationships to a single word at any given time.

Secondly, Phrase Net does not provide the user with access to the underlying text. Therefore when occurrences of a relationship are provided, only small extracts are shown to the user. These extracts may not be enough for the user to gain an appreciation of the context in which a relationship is appearing, nor do they provide a visual overview of where the relationship occurs in a text. *Links* addresses this by allowing the user to directly access the underlying text for every occurrence of a relationship, as well as providing the user with an immediate visual representation of the occurrences (Figure 3.12). This visual overview may be used to quickly compare the occurrences of multiple relationships and generally lends itself to knowledge discovery more efficiently than the list approach taken by Phrase Net.

Finally, the Phrase Net approach does not provide a filtering function to the user, as is one of the requirements for user-friendly visualisations set out by Dadzie *et al.* Conversely, *Links* provides a relatively large set of options for the user to filter on. The data presented therefore moves from simple static visualisation to a dynamic, user directed visualisation. This approach is designed to make the data more useful to the user and to improve the quality of the visualisations.

From a user interaction perspective, the tool uses simple navigation techniques and interactions that are part of the common design patterns of iOS. It is believed that by adhering to these interface principles, the need for explicit user training is removed. Additionally, the user does not require any knowledge of the inner workings of the tool's algorithms to generate data or perform data filtering. The focus on ease-of-use should address the main concerns presented by Gibbs and Owens [1].

4.3 | Conclusion

This chapter presented an evaluation of both the technical and qualitative aspects of *Links*. The findings from the output of the tool show that there is merit in this approach to literature visualisation, and that the data gathered can aid in the process of literature analysis. Initial qualitative feedback received from a lecturer of English at University College Dublin has been positive. Subsequent research should focus on surveying users from a broad range of humanities disciplines to evaluate the quality of the data and the usability provided by *Links*, with adjustments made based on this feedback.

5 | Future Work

This chapter discusses the potential future work which may add to the research that has been presented in this dissertation. Section 5.1 outlines how user studies should be conducted and discusses their benefits to the overall usability and quality of *Links*. Section 5.2 then discusses further features that may be added to the tool, both in a usability perspective and in a data quality perspective.

5.1 | User Studies

Gibbs and Owens noted that only 33% of tools created for use within the digital humanities ever conducted usability reviews [1]. This point is emphasised as a deciding factor in the poor uptake of analysis tools in the field. To avoid the same shortfall from affecting the quality and usability of *Links*, it is clear that in-depth usability reviews should be next step to help achieve the original goal of the project: to create a user-friendly literature analysis tool that aids in knowledge discovery.

The target demographic for the tool is students and researchers studying English literature. Studies should therefore analyse the usage of the tool in this context. Examples include analysing the usage of the tool for an English literature assignment in a college course, or examining its uses in a research project in the field. User studies should focus on two metrics; the usability of the tool, and the quality and usefulness of the data according to the users.

Usability testing would allow us to carefully study how users interact with the tool. These tests could be conducted though observation of a set of users interacting with the tool during a ‘think-aloud’ session. Each user should be given a set task, such as identifying the relationships present in a book, and then they should outline their thoughts during the interaction with the tool. Questions may be posed to the user to gauge their understanding of the tool and its features. This process would allow us to build feedback on which features are clear and intuitive, and which are vague or difficult to use. Essentially, this approach should focus on questioning whether users can effectively learn how to use the features provided by the tool, and if the observations prove or disprove the assumptions made about user ability made in Section 3.6. The feedback gathered during these sessions may then be iteratively incorporated into the tool to increase the overall usability.

A second set of tests should analyse the quality of the generated data based on feedback from a set of users. These tests should focus on two metrics. Firstly, they should determine whether the knowledge discovery features of the tool are performing in a way that allows users to find new data in a real world context. Secondly, they should focus on the users' opinions about the data produced. As discussed in Section 4.2, literature and its analysis is entirely subjective, therefore there is a need to standardise the results of such a qualitative study as much as is feasible. An approach to standardisation would be to assign the users the same analysis task, such as analysing a certain novel. With a large enough sample size, the subjective views on the novel should converge into manageable sets; for example, if the majority of users mention that they think the relationship between two characters is the most important in the book, and this is not detected by *Links*, then there is an area immediately highlighted for improvement. It is important to avoid catering the iterative changes to the qualitative side of the tool based on feedback that is specific to single users. Such changes may skew results that users on a whole do not necessarily agree with.

Finally, an interesting user study would be to evaluate how the tool copes with foreign language literature. The tagging algorithm present in the tool should detect the language that the input text is written in and perform lexical analysis accordingly. The graph construction algorithms, and relationship detection features, were designed with English in mind. It is entirely possible that the grammatical structure of other languages would cause the algorithm to miss important data. For example, in German a conjunction causes the verb to move to the end of the sentence. This would undo the intention of setting *two* to be the adjacency limit during parsing as described in Section 3.3.4. Overcoming these challenges could prove to be an interesting area of research.

5.2 | Additional Features

The features implemented in *Links* represent only a small subset of the possible analysis metrics that could be provided to the user. This section describes the possible additions to the feature set provided by the tool.

5.2.1 | Supporting Additional Text Formats

From a usability perspective, the exclusive reliance on the plain text format currently seen within the tool presents an unfortunate limitation on users; any texts they may wish to analyse must be in this format. Most novels are unlikely to be available in plain text formats; in particular, any e-books that a user owns will likely be available only in the `epub` or `mobi` formats. If a user cannot import the texts that they own for analysis, then they will have little reason to make use of the tool.

Addition of importers for the `epub` and `mobi` should therefore be a priority for further development of the tool. Open source importers for these formats are widely available and should be relatively straightforward to combine with the existing parsing functionality present in the tool.

5.2.2 | Emotive Analysis

As is demonstrated with the implementation of sentiment analysis in the tool, pre compiled data sets can be used to provide additional interesting analysis metrics to the user. *Links* could be extended to implement emotive analysis. Emotive analysis is the process of tagging a word with an emotive category. The vast majority of words do not have an associated category and are therefore considered to be neutral in this context. An example of such a database is WordNet-Affect [28]. This database maps words to their affective categories by utilising the synonym information available in WordNet. For example, the words *'joy'* and *'happiness'* would be tagged as positive emotions. The database is still a work in progress and has only mapped a very small percentage of WordNet's entirety. Nevertheless, the ability to explicitly tag these sets of words within pieces of text, especially literature, would be a powerful ability to provide to the user. Combined with the distribution information that is collected for each word, visualising this data would allow the user to easily identify important or emotive sections within a text, and would help in expanding the knowledge discovery options of the tool.

5.2.3 | Collaboration

Experimentation with the addition of collaborative functionality could produce increases in the effectiveness of knowledge discovery and the quality data visualisations provided by the tool. Allowing users to share their annotations and tags among one another would allow for research groups to delegate analysis work between each member.

Research may be done to identify similar collaboration tools in use in the digital humanities, and user studies could be conducted to determine how collaborative analysis processes are currently handled for literature analysis. The results gathered could be then be used to determine what the user requirements are, and how the additional functionality could be implemented.

5.3 | Conclusion

Chapter 4 demonstrated that the data produced by *Links* can be useful in a variety of circumstances. The recommendation of the author is that future work on the tool should focus primarily on user interaction aspects, ensuring that it is usable by as broad of an audience as possible. Further research should evaluate the worth of the data being produced, and tailor the output to the feedback received from testing. Additional research may also focus on extending the analysis options provided by the tool based on the cutting edge research being undertaken in the field.

6 | Conclusion

The tools developed for use within the humanities have long suffered from an inability to reach a broader audience due to their complexity [1]. The aim of this dissertation has been to create a literature visualisation and analysis tool that would augment the study of literature, while still remaining open and intuitive in its presentation of features and data.

As the technology industry moves forward, the rate of tablet users increases while traditional desktop usage decreases. It is imperative that the tools of the future are available to users through the medium they are most comfortable with. The technical design and implementation of *Links* has shown that it is possible to implement these types of complex analysis tools on a tablet device. The work underlines how special concern must be given to both time and memory constraints to create a tool that is fast enough to meet user expectations. Furthermore, the focus on user interaction has shown that visualisation and exploration of large-scale graph structures is possible on such devices, despite limited screen size and processing power, by following an established set of design principles.

The central motivation of *Links* is to enable knowledge discovery by presenting interesting quantitative data to the user. This data should help them identify pertinent sections of the text that they would not have ordinarily examined. The qualitative evaluation of the tool has shown that the data produced by *Links* meets this aim of supporting knowledge discovery within the text.

Using proximity to measure the strength of a link between words has proven itself to be a valuable and reliable metric for highlighting important relationships in the text. The ability to move from a visual overview of the data to the exact quote in the text that generated it, enables the user to study literature in an interactive and exploratory way that was not possible before. The comparison of multiple books opens up new avenues for the user to examine the differences and similarities between authors. Similarly, sentiment analysis has shown it provides the user with the ability to quickly discovery highly polarising sections of the text, simply by exploring a chart.

Future work may focus on enhancing the results through the addition of new analysis features to the tool. User studies need to be carried out to ensure that the tool is as user friendly as possible, and that the data being produced is of a high enough standard to positively aid in analysis.

Through both the technical and qualitative evaluations of the tool, the author believes that the original aims of *Links* have been met. *Links* represents a significant departure from the world of complex tools used exclusively by researchers, to one in which any person can perform literature analysis. In these modern times where school children are replacing textbooks with tablets, the research undertaken by this dissertation shows that with further work, the fundamental way in which we study with literature could quickly change in the future.

Bibliography

[1] F. Gibbs and T. Owens, "Building better digital humanities tools: Toward broader audiences and user-centered designs," *Digital Humanities Quarterly*, vol. 6, no. 2, 2012.

[2] J. Titcomb, "Tablets forecast to overtake PC sales at end of year," *The Telegraph*, September 2013.

[3] F. van Ham, M. Wattenberg, and F. B. Viegas, "Mapping text with phrase nets," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1169–1176, Nov. 2009. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2009.165

[4] F. B. Viegas, M. Wattenberg, F. van Ham, J. Kriss, and M. McKeon, "Manyeyes: A site for visualization at internet scale," *IEEE Transactions on Visualization and Computer Graphics*, vol. 13, no. 6, pp. 1121–1128, Nov. 2007. [Online]. Available: http://dx.doi.org/10.1109/TVCG.2007.70577

[5] Stanford University Engineering Department, "Stanford scientists put free text-analysis tool on the web," https://engineering.stanford.edu/research-profile/stanford-scientists-put-free-text-analysis-tool-web.

[6] R. Weiss, "Measuring media bias: A computational method of indexing affective slant amongst news outlets," unpublished - Information available at: http://www.etcml.com/blog/investigate-the-news : Last Accessed Thursday 24 April 2014.

[7] M. J. Collins, "A new statistical parser based on bigram lexical dependencies," in *Proceedings of the 34th Annual Meeting on Association for Computational Linguistics*, ser. ACL '96. Stroudsburg, PA, USA: Association for Computational Linguistics, 1996, pp. 184–191. [Online]. Available: http://dx.doi.org/10.3115/981863.981888

[8] D. Mackenzie, "Literature by the numbers," *Nautilus*, vol. 1, no. 6, Oct. 2013.

[9] P. Juola, "How a computer program helped reveal J. K. Rowling as author of A Cuckoo's Calling," *Scientific American*, vol. 309, no. 3, Sep 2013.

[10] B. Gretarsson, J. O'Donovan, S. Bostandjiev, T. Höllerer, A. Asuncion, D. Newman, and P. Smyth, "Topicnets: Visual analysis of large text corpora with topic modeling," *ACM Trans. Intell. Syst. Technol.*, vol. 3, no. 2, pp. 23:1–23:26, Feb. 2012. [Online]. Available: http://doi.acm.org/10.1145/2089094.2089099

[11] A. Voutilainen, "Part-of-speech tagging," *The Oxford handbook of computational linguistics*, pp. 219–232, 2003.

[12] B. White and E. Cambria, "Jumping NLP curves: A review of natural language processing research," *IEEE Computational Intelligence Magazine*, vol. 9, p. 2, 2014. [Online]. Available: http://sentic.net/jumping-nlp-curves.pdf

[13] H. R. B. Gery W. Ryan, "Techniques to identify themes," *Field Methods (FMX)*, pp. 85–109, 2003.

[14] R. H. Baayen, *Word Frequency Distributions.* New York City, NY, USA: Springer Publishing, 2001.

[15] G. A. Miller, "Wordnet: A lexical database for english," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, Nov. 1995. [Online]. Available: http://doi.acm.org/10.1145/219717.219748

[16] B. Pang and L. Lee, "Opinion mining and sentiment analysis," *Found. Trends Inf. Retr.*, vol. 2, no. 1-2, pp. 1–135, Jan. 2008. [Online]. Available: http://dx.doi.org/10.1561/1500000011

[17] A.-S. Dadzie and M. Rowe, "Approaches to visualising linked data: A survey," *Semant. web*, vol. 2, no. 2, pp. 89–124, Apr. 2011. [Online]. Available: http://dx.doi.org/10.3233/SW-2011-0037

[18] F. Å. Nielsen, "AFINN," Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, mar 2011. [Online]. Available: http://www2.imm.dtu.dk/pubdb/p.php?6010

[19] S. Baccianella, A. Esuli, and F. Sebastiani, "Sentiwordnet 3.0: An enhanced lexical resource for sentiment analysis and opinion mining," in *Proceedings of the Seventh*

International Conference on Language Resources and Evaluation (LREC'10), N. C. C. Chair), K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, and D. Tapias, Eds. Valletta, Malta: European Language Resources Association (ELRA), may 2010.

[20] M. S. Zarra, *Core Data (2nd edition): Data Storage and Management for iOS, OS X, and iCloud.*

[21] D. Biber, S. Johansson, G. Leech, S. Conrad, and E. Finegan, *Longman Grammar of Spoken and Written English (Hardcover).* Pearson ESL, November 1999.

[22] J. Nielsen, *Usability Engineering.* San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1993.

[23] S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The information visualizer, an information workspace," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '91. New York, NY, USA: ACM, 1991, pp. 181–186. [Online]. Available: http://doi.acm.org/10.1145/108844.108874

[24] E. Sadun, *The Advanced iOS 6 Developer's Cookbook*, 4th ed. Addison-Wesley Professional, 2013.

[25] W. N. Francis and H. Kucera, "Brown corpus manual," Department of Linguistics, Brown University, Providence, Rhode Island, US, Tech. Rep., 1979. [Online]. Available: http://icame.uib.no/brown/bcm.html

[26] J. Austen, *Pride and Prejudice.* RD Bentley, 1853. [Online]. Available: http://books.google.ie/books?id=kQ0mAAAAMAAJ

[27] G. Martin, *A Game of Thrones*, ser. Song of ice and fire. HarperVoyager, 2011. [Online]. Available: http://books.google.ie/books?id=3Wf_ffkFQmgC

[28] C. Strapparava and A. Valitutti, "WordNet-Affect: An affective extension of WordNet," in *Proceedings of the 4th International Conference on Language Resources and Evaluation.* ELRA, 2004, pp. 1083–1086. [Online]. Available: MISSING