Extracting Social Network from Literary Prose

Tarana Tasmin Bipasha
University of Arkansas, Fayetteville

Extracting Social Network from Literary Prose

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science

by

Tarana Tasmin Bipasha
Bangladesh University of Engineering & Technology
Bachelor of Science in Computer Science, 2011

December 2019
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

Susan Gauch, Ph.D.
Thesis Director

Brajendra Nath Panda, Ph.D.
Committee Member

Qinghua Li, Ph.D.
Committee Member

ABSTRACT

This thesis develops an approach to extract social networks from literary prose, namely, Jane Austen's published novels from eighteenth- and nineteenth- century. Dialogue interaction plays a key role while we derive the networks, thus our technique relies upon our ability to determine when two characters are in conversation. Our process involves encoding plain literary text into the Text Encoding Initiative's (TEI) XML format, character name identification, conversation and co-occurrence detection, and social network construction. Previous work in social network construction for literature have focused on drama, specifically manually TEI-encoded Shakespearean plays in which character interactions are much easier to track in due to their dialogue-driven narrative structure. In contrast, prose is structured quite differently; character speeches are not very clearly formatted, making it more difficult to assign specific dialogue to each character. We implement two different parsing strategies based on context size (chapter scope and paragraph scope) to detect character interactions. To check the accuracy of our methods, we conduct one evaluation that is based on network statistics and another evaluation that involves measuring similarity (edit distance) between the networks constructed from manually encoded novels versus our constructed graphs. Our findings suggest that the choice of context size is non-trivial and can have a substantial influence on the resulting networks. In general, the paragraph level interaction approach seemed to be more accurate.

ACKNOWLEDGEMENTS

I would like to thank my thesis advisor Susan Gauch, Ph.D. for providing me with the opportunity to work with her during this project and her guidance throughout my graduate education. I also thank my thesis committee, Brajendra Nath Panda, Ph.D. and Qinghua Li, Ph.D. for reviewing my thesis.

I would also like to express my very profound gratitude to my parents and to my spouse for providing me with constant support and continuous encouragement throughout this journey and through the course of studying and writing this thesis. This achievement would not have been possible without them.

TABLE OF CONTENTS

1. INTRODUCTION

The key areas of research in literary studies could be divided into three categories; the first is philology - the study of words; the second is bibliography - the study of books as objects; and the third is criticism - the understanding of literary meaning over time. Close reading of texts has always been the primary task of literary scholarship since the very beginning of criticism. By studying specific words precisely, it constructs word-based interpretations. On the other hand, the computational approaches propose another technique for literary study called "Distant reading" (F. Moretti, 2011). According to Grayson et al., distant reading takes several forms, such as character frequency analysis, statistical topic models, character profiling and sentiment analysis (Grayson et al., 2016). It's not feasible for a person to read all available texts, nor very easy to perceive insight by reading them. In order to develop new literary perceptions, computational approaches must deliver information about the fictional texts. These approaches are being increasingly adopted by scholars to explore the unanswered questions in literary field.

Several computational approaches employ social network analysis. It not only adopts some of the existing analysis techniques but also offers a unique level of abstraction (i.e. representation of social graph with nodes and edges). In a literary context, the application of social network analysis usually involves the creation of character networks from text, where each vertex indicates a character and each edge signifies a relation between characters. Character associations are generally recognized by studying the co-occurrences between each pair of characters. The scholars often adopt the methodologies derived from constructed networks to assess novel literary hypothesis, even though the success of these approaches significantly rely on the superiority of the underlying networks. Extracting a character network from 18th and 19th century literary fiction is complex for two prime reasons; first, the characters are often referenced in ambiguous or implicit manners; and second, they share names and aliases with other characters in the same novel (Grayson et al., 2016). Moreover, building a social network from prose fiction is crucial since character interactions are much easier to track in play/drama due to its dialogue-driven narrative structure whereas fictions are formatted differently--character speeches are not very clearly formatted, making it more difficult to assign specific dialogue to each character.

With the emergence of digitization, humanities scholars are always in need of more and more texts to run corpus level computational text analysis. Even though digital texts are available from a wide variety of sources, one generally has to go through a number of preprocessing tasks to transform those texts into a standard format. The files found at numerous sources are not only comprised of plain, unformatted text, they also contain a lot of boilerplate text that must be removed prior to textual analysis. Computation approaches to literary analysis thus requires an automatic text encoder that takes these unformatted texts as input and transforms them into an XML file that is compatible with all kind of text processors/tools dealing with XML formatted corpus.

Our research is primarily concerned with three goals:

[1] First, building an automatic text encoder that tags plain, un-TEI encoded text in such a way that this approach can be made broadly applicable.

[2] Second, identifying characters (speakers) from literary text.

[3] Third, determining the appropriate context size to use when building a social network based on character interactions.

In this thesis we present a method to automatically encode plain literary text with the TEI format. Most of the studies in social network analysis of literature start with a manually processed and XML-encoded dataset that requires a huge amount of human effort to produce and is thus quite expensive. To address this issue and make computation literary analysis much more broadly applicable , we built an XML encoder that provides basic TEI tagging. Next, we developed a technique to identify characters from prose and produce a character list. Previous work in social network construction for literature have focused on drama. Gauch et al.'s previous study on social network analysis is based on TEI encoded Shakespeare's plays, in which character interactions are much easier to track in due to their dialogue-driven narrative structure (Gauch et al., 2018). In contrast, prose is formatted in a different way; character interactions are not clearly structured, making it difficult to find character interactions. Our thesis implements two strategies based on context size (chapter scope and paragraph scope) to observe character interactions. To check the accuracy of our methods, we conduct evaluations based on network statistics and network similarity (edit distance) measures. Our findings suggest that the choice of context size is non-trivial and can have a substantial influence on the resulting networks.

Chapter 2 presents a summary of related work on social network analysis strategies in the field of literature. In Chapter 3, we describe our approaches to build the TEI Encoder designed for converting plain text into TEI formatted XML and then describe our character identification methodology, along with our different parsing strategies to generated social networks of literary prose. In Chapter 4, we evaluate our techniques and present our findings. Finally, in Chapter 5, we present conclusions and discuss future work in this area.

2. RELATED WORK

In this chapter, we discuss social network analysis in the field of literature and the research on social network applications. At its most basic, a social network graph consists of vertices (social characters) and edges (social relations). However, we can extract a lot more information, both explicit and implicit, from its structure. Graph theory offers some mathematical as well as systematic methods for investigating the structure. In short, social network analysis is the process of studying social structures using graph theory concepts.

2.1 Social Networks in Literature

An array of different tactics have been considered to detect meaningful interactions among characters in novels. Alberich et al. conducted one of the first studies in this field (Alberich et al., 2002). They created the Marvel Universe network by recognizing connections between characters based only on the characters occurrence in the same comic but not the interactions among them. This technique was improved on by Gleiser who introduced weights, where weights represent the collaboration between actors throughout the script that occurs repetitively (Gleiser, 2007). F. Moretti, on the other hand, adopted a completely different tactic to analyze the works of Shakespeare. He constructed social networks based on dialogues indicating that interactions alone can play a major role in social network analysis (F. Moretti, 2011). Taking similar approach, Elson et al. also constructs social networks from 19th century literature (Elson et al., 2010). This method involves character name clustering and automated speech attribution as they extract conversations from sets of dialogue acts, which results in a high level of precision (96%) but lower recall (57). Agarwal et al. include another type of social event, observation, along with dialogue interaction while constructing the network for Alice in Wonderland (Agarwal et al., 2012). They construct a weighted directed network, where the direction indicates who is observing whom. Jayannavar et al. propose another extraction technique that considers general character interactions along with conversational interactions and observations (Jayannavar et al., 2015).

2.2 Text Encoding Initiative (TEI) XML

XML is a markup language that states a set of instructions for encoding texts in a format that is both human-readable and machine-readable. The design goals of XML emphasize simplicity, generality, and usability across the Internet. This textual data format offers strong support for different languages via Unicode. Although the design of XML focuses on documents, the language is widely used for the representation of random data structures such as those used in web services (XML, 2019). The Text Encoding Initiative (TEI) is a text-centric community of practice in the academic field of digital humanities (Text Encoding Initiative, 2019). TEI is a consortium that jointly develops and maintains a standard for the digital representation of texts. Its chief deliverable is a set of Guidelines that specify encoding methods for machine-readable texts, mainly the ones in humanities, social sciences and linguistics. The format differs from other well-known open formats (HTML, OpenDocument etc.) in that it's primarily semantic rather than presentational. Since 1994, the TEI Guidelines have been widely used by libraries, museums, publishers, and individual scholars to present texts for online research, teaching, and preservation.

2.3 Named Entity Recognizer (NER) Tagging

One of the primary tasks in dealing with computational analysis of literature is the need to identify when a specific character is being referenced. Names for the same individual, in literature and in other printed matter, may appear in many forms. The problem of named-entity recognition (NER) (also known as entity identification, entity chunking and entity extraction) is a sub-task of information retrieval that seeks to locate and classify named entities in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc. (Named-entity recognition, 2019). Various NER systems have been created so far that use linguistic rule-based techniques as well as statistical models such as machine learning. Earlier NER systems are most often based on hand-crafted rules. These systems typically obtain better precision, but at the cost of lower recall and months of work by experts. Therefore, even though considered as highly efficient, they are expensive, domain-specific, and non-portable. Learning based models provide better accuracy compared to rule based models. Stanford NER, also knows as a CRF (Conditional Random Field) Classifier, is one of them ((Finkel et al., 2005)). It's an open source Java implementation of a Named Entity Recognizer. It provides a trained model to label sequences of words in a text which are the names of things, such as person, organization and location names. Another state-of-the-art NER Model is spaCy. Being a free and an open-source library, spaCy provides an efficient statistical system for NER in python (spaCy, 2015). Apache OpenNLP library is also a machine learning based toolkit for NER (Apache OpenNLP, 2019). Hybrid systems such as BI-LSTM-CRF model performs very well for many cases. Among the very recent systems, Bert NER looks very promising. In this thesis, we used Stanford NER because it is the most accurate and free NER system available.

2.4 Graph Analysis Software

The overall goal of this work is to create a social network that captures character interactions within prose. Typically, these networks are displayed visually. The graphs can be drawn manually, however, there are a variety of software packages to make this process much easier. Graphviz is an open source graph visualization framework. It has several main graph layout programs suitable for social network visualization (Graph Visualization Software, 2017). Graph-tool is another Python module for manipulation and statistical analysis of graphs which can be used to work with very large graphs in a variety of contexts (T. de P. Peixoto, 2019). R also contains several packages relevant for social network analysis. JUNG is a Java API and library that provides a common and extensible language for the modeling, analysis, and visualization of relational data. It supports a variety of graph types (including hypergraphs), supports graph elements of any type and with any properties, enables customizable visualizations, and includes algorithms from graph theory, data mining, and social network analysis (JUNG, 2010). Gephi, another widely used open-source network analysis and visualization software package, provides easy and broad access to network data (The Open Graph Viz Platform, 2017). The Gephi Toolkit project packages essential modules (Graph, Layout, Filters, IO etc.) in a standard Java library (Gephi Toolkit, 2017). Along with graph visualization and manipulation capability, Gephi also computes a wide range of mathematical features for each graph which is why we chose to use Gephi in this thesis and we make use of its built-in mathematical functions in our graph analysis work.

2.5 Graph Similarity Measure

Since we will be exploring different ways of extracting character interactions within prose, we need some method for comparing the resulting graphs. Graph similarity/matching, the problem of finding a similarity between graphs, is a well-studied problem in computer science. It has several applications in various fields (image processing, social networks, biological networks, chemical compounds etc.), hence numerous similarity measure algorithms have been suggested. The proposed techniques can be classified into three main categories: iterative methods, feature extraction methods, and edit distance/graph isomorphism (Koutra et al., 2011) .

The key idea behind the iterative methods is that “two nodes are similar if their neighborhoods are also similar”. The nodes exchange similarity scores among each other in each iteration. The whole process ends only when convergence is achieved. There are quite a few successful algorithms that belong to this group: the similarity flooding algorithm by Melnik et al. which solves the “matching” problem by determining the correspondence between the nodes of two given graphs (Melnik et al., 2002). SimRank is another algorithm that measures the self-similarity of a graph, i.e., it measures the similarity between all pairs of nodes in one graph (Glen Jeh and Jennifer Widom, 2002). It computes all pairs similarity scores iteratively by propagating similarity scores in the A^2 matrix. Zager and Verghese proposed a recursive method that introduced the idea of coupling the similarity scores of both nodes and edges; most of the proposed previous methods focus on the nodes’ scores only (Zager and Verghese, 2008).

In contrast to the previous iterative methods, the philosophy behind feature extraction methods is that graphs those who are similar share certain properties among them, such as eigenvalues, degree, diameter. The process includes extracting the features first, then applying a similarity measure to weigh in the similarity between the aggregated statistics, thus calculating the similarity between the graphs. These methods are efficient and scale quite well. However, depending on the selected statistics, the results that are achieved may not be always intuitive. For example, it is not impossible to get high similarity between two graphs that do not have very similar node set size, which is not always desirable.

Another approach to evaluating graph similarity is graph edit distance, which is a generalization of the graph isomorphism problem. In these methods, the goal is to transform one graph to the other by doing a number of operations such as additions, deletions, substitutions of nodes or edges, and reversions of edges. It calculates a graph similarity measure analogous to

Levenshtein distance for strings (Graph edit distance, 2019). It is defined as minimum cost of edit path (sequence of node and edge edit operations) that associates each operation with a cost, and it attempts to find the sequence of operations that minimizes the cost of matching the two graphs. In this thesis, we used graph edit distance to evaluate our resulting graphs. The mathematical definition of graph edit distance is dependent upon the definitions of the graphs over which it is defined, i.e., whether and how the vertices and edges of the graph are labeled and whether the edges are directed. Generally, given a set of graph edit operations (also known as elementary graph operations), the graph edit distance between two graphs $g1$ and $g2$, written as $\mathrm{GED}(g1, g2)$ can be defined as

$$\mathrm{GED}(g1, g2) = \min_{(e_1,\ldots..e_k)\in \mathcal{P}(g1,g2)} \sum_{i=1}^{k} c(e_i)$$

where $\mathcal{P}(g1, g2)$ denotes the set of edit paths transforming $g1$ into (a graph isomorphic to) $g2$ and $c(e) \geq 0$ is the cost of each graph edit operation e.

The set of elementary graph edit operation that we considered in our calculation includes:

- Node insertion to introduce a new labeled node to the graph.
- Node deletion to remove a single (often disconnected) node from the graph.
- Node substitution to change the label of a specific node.

In our work, we have chosen to use the graph edit distance (GED) method for graph similarity in our experiments because it is one of the most flexible and widely accepted similarity measure techniques available for graphs where the basic distortion operations of GED can cope with arbitrary labels on both nodes and edges as well as with undirected or directed edges. In contrast with other measures, GED does not have any restriction, thus can be applied to all kinds of graphs. Additionally, GED is also known to be error-tolerant with the capability to locate graphs that are of interest regardless of the presence of noises or errors in the data set.

3. APPROACH

The system for extracting the social network consists of three main components as shown in Figure 1. The “Text Encoder” converts plain text into the Text Encoding Initiative’s (TEI) XML format. The purpose of “Character Identifier” is to recognize all the characters of a prose. Finally, the “Social Network Generator” automatically parses XML formatted novels to extract basic information and generates social network based on various metrics for each novel. Figure 1 shows the main components of the system architecture, which are discussed in more detail in the following subsections.

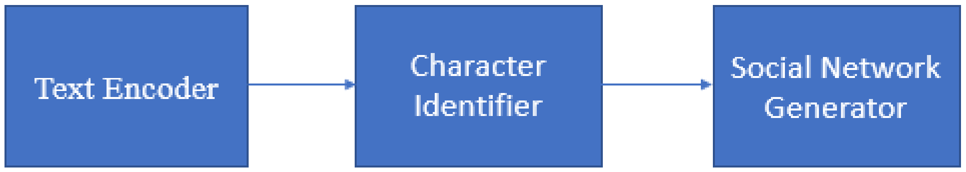

Figure 1: Block diagram of our system

3.1 Data Set

We focus our work on the six novels of Jane Austen, one of the most widely studied authors of English literature. These plain (unstructured) text novels have been digitized by project Gutenberg (Books by Austen, Jane, 1971). At the beginning, data preparation involves automatic annotation of the novels, i.e., we convert the plain Gutenberg text into a TEI-compliant xml file.

To evaluate our work, we collected the same six novels of Jane Austen, manually encoded in TEI format (Austen Said, 2017). We call them our "truth values". These novels are also passed through the parser to extract necessary information and generate character networks which are then compared with the social networks produced by our models.

3.2 Text Encoder

We have designed an automatic text encoder that takes in the plain text of a novel as input and converts them into an XML document that meets the basic TEI requirements. With the emergence of digitization, humanities scholars are always in need of more and more texts to run corpus level computational text analysis. Even though we have turned out to be an expert at finding texts from a wide variety of sources, we often have to go through a number of preprocessing tasks to transform those texts into a standard format. The files found at numerous sources are not only comprised of plain, unformatted text; but also, a lot of boilerplate text that demands to be removed prior to textual analysis. Our "Text Encoder" component provides a script that takes these unformatted texts as input and transforms them into an XML file that is compatible with all the text processors/tools dealing with TEI formatted XML.

In order to convert the plain Gutenberg text, we ran it though a python script that normalizes the formatting, detects headings and chapter breaks, removes metadata, etc. The output generates a teiHeader that includes the title and author of the novel and adds "text", "body", "div", and all the essential "p" tags which ultimately results in an XML document that meets the basic TEI requirements. Full TEI encoding uses some additional tags; for example,

<seg> tag inside <p>, tags related to publication information inside <teiHeader>, etc., but we are focusing on tagging for the purpose of identifying character interactions to build a social network. As such, we did not annotate with all TEI tags. In Table 1, we present the list of tags we used to annotate the un-TEI encoded text to make this approach more broadly applicable.

Table 1: List of tags we used to annotate the un-TEI encoded text

Primary Tags	1. <teiHeader>
	2. <text>
	3. <body>
	4. <div>
	5. <p>
Inner Tags	6. <fileDesc>
	7. <publicationStmt>
	8. <title>
	9. <author>
	10. <listPerson>

Our algorithm follows top down parsing strategy based on pattern matching in an iterative manner. It basically scans through every line from start to end. At the beginning, it detects the title and author information and generates the <teiHeader> of the XML. After that, it constructs the main part, i.e., the <body> of the XML. Inside the <body>, the parser looks for the chapter or volume information and starts the <div> tag which ends before the next chapter begins. So, each chapter is wrapped inside a <div> tag. The parser considers an empty line as the start of a new paragraph. Therefore, all other lines between the empty lines are concatenated into a single paragraph and tagged with <p> tag. The parser also removes metadata and redundant whitespaces before writing the output to a file. Figure 2 presents the algorithm for the overall task and Figure 3 shows a sample XML output from the converter.

```
Initialize encodedText
For each line of input text
    If the line contains “Author” information
        extract author
    If the line contains “Title” information
        extract title
    If the line starts with “**start**”
        startSeen = true
        paragraph = “”
        continue
    If the line starts with “**end**”
        Break
    If not startSeen
        continue
    Append title and author information to the header
    If not line
        paragraph = paragraph.strip()
        if paragraph
            encodedText.append(paragraph)
            encodedText.append("</p>")
        paragraph = "<p>"
    else:
        if line.startswith("Volume"):
            volume = line.strip()
            line = ""
```

Figure 2: Algorithm to convert plain text to TEI XML format

```
                if line.startswith("Chapter "):

                    if len(volume) > 0:
                        line = volume + ":" + line.strip()
                    if firstChapter:
                        paragraph = "<div><p>"
                        firstChapter = False
                    else:
                        paragraph = "</div><div><p>"
                paragraph += " " + line

        encodedText.append("</div></body></text></TEI>")
        remove metadata and whitespaces
        open file to write encodedText
close file
```

Figure 2: Algorithm to convert plain text to TEI XML format (Cont.)

```
<?xml version="1.0" encoding="utf-8"?><TEI xmlns=http://www.tei-c.org/ns/1.0
version="5.0">
<teiHeader>
      <fileDesc>
          <titleStmt>
              <title> Pride and Prejudice </title>
              <author> Jane Austen </author>
          </titleStmt>
          <publicationStmt>
               <availability status="free"><p>Project Gutenberg</p></availability>
          </publicationStmt>
       </fileDesc>
</teiHeader>
<text>
      <body>
          <div>
               <p> Chapter 1 </p>
               <p> “My dear Mr. Bennet,” said his lady to him one day, “have you heard …” </p>
               <p>Mr. Bennet replied that he had not. </p>
                ……………
          </div>
```

Figure 3: The structure of the generated TEI encoded XML of a novel

```
        <div>
              <p> Chapter 2 </p>
              <p>Mr. Bennet was among the earliest of those who waited on Mr. Bingley…. </p>
               ……………
         </div>
         ……………
     </body>
</text>
```

Figure 3: The structure of the generated TEI encoded XML of a novel (Cont.)

3.3 Character Identifier

One of the most challenging task of our thesis was to identify the characters (speakers) by "chunking" names (such as Mr. Bennet) from the plain text. Since our goal is to correctly identify the speakers of a novel, we focus on the characters who actively participate in direct conversations. The process itself comprises of several steps. First, we detect conversations between characters. Then, we automatically build a character map wherein each character has only one entry and the corresponding aliases for that character (i.e., all possible names). Using this map's unique name (key), we replace all occurrences of a character's aliases throughout the text.

Our algorithm (Figure 4) proposes an iterative, novel approach to extract and build character map with the help of Stanford NER tagger (Finkel et al., 2005). It passes each of the prose through the NER tagger to locate and classify named entities in the text and extracts only the noun phrases which are categorized as persons. During this tokenization procedure, we implemented N-gram (mainly bi-gram and tri-gram) technique to identify all the variations of a character name. For example, in "Pride and Prejudice", Elizabeth Bennet was referred to as Miss Bennet, Miss Elizabeth, Eliza Bennet, etc., throughout the text. Since the Stanford tagger is unable to identify the honorific titles preceding a person's name such as Mr., Mrs., etc., we used a list of these titles to resolve the names that contain prefix. The list consists of 10 titles (Mr., Mrs., Miss, Ms., Master, Lady, Mx, Sir, Dr, Lord) that are most common. Then, we combine all the variations of a character's name into a list and eventually store that list in a character map where the associated key represents the unique id of that entity. Figure 5a shows part of the constructed character map for the novel "Pride and Prejudice". Finally, we match and replace all the occurrences of a name throughout the text with the unique id associated to it as shown in Figure 5b.

```
Initialize and start ner_tagger server
Initialize a character map
For each line of text
    tokenized_text = word_tokenize(line)
    tagged_words = ner_tagger.tag(tokenized_text)
    For each word in tagged_words
        if the tag of the word is 'Person':
            check if this is a multiword name by checking the consecutive
                words before and after the current word
            put the name in character list
For each item of character list
    If it appears in a line that contains quoted speech:
        Put item in a character map

For key, value of character map
    Find co-referents of the item and combine them
    For item in value:
        If item in line:
            line = line.replace(item, key)
```

Figure 4: Algorithm for character name identification

character_1 : ['Mr. Bennet']

character_2 : ['Miss Lydia Bennet', 'Lydia Bennet', 'Miss Lydia']

character_3 : ['Catherine de Bourgh', 'Catherine de', 'de Bourgh', 'Catherine']

character_4 : ['Lady Anne Darcy', 'Lady Anne', 'Anne Darcy']

……. …….. ……..

character_25 : ['Michael S. Hart', 'Michael S.', 'S. Hart', 'Michael']

Figure 5a: Character map constructed from original text

Original Text :

“My dear Mr. Bennet,” said his lady to him one day, “have you heard that Netherfield Park is let at last?” Mr. Bennet replied that he had not.
“But it is,” returned she; “for Mrs. Long has just been here, and she told me all about it.”
Mr. Bennet made no answer.

Text after character name replacement:

<p> “My dear character_2,” said his lady to him one day, “have you heard that Netherfield Park is let at last?” </p>
<p> character_2 replied that he had not. </p>
<p> “But it is,” returned she; “for character_3 has just been here and she told me about it”</p>
<p> character_2 made no answer. </p>

Figure 5b: All the character names are replaced with their unique id throughout the text of the novel

3.4 Social Network Generator

The main purpose of this component is to determine appropriate context size to use when building a social network based on character interactions. We describe two different character network construction strategies/parsers; Chapter Level Parser and Paragraph Level Parser, to detect co-occurrences, based on the scope of context window over the text of a novel. With

Chapter Level Parser, we automatically parse TEI-encoded XML formatted novels to extract information within the scope of chapter to determine which characters were speaking to whom and total number of interactions among those characters. Paragraph Level Parser extracts similar information, but within the scope of a paragraph; significantly smaller context size compared to the chapter scope. These co-occurrences are then used to construct a weighted undirected network representation of the novel.

3.4.1 Information Extraction from TEI Encoded Xml

We used Java DOM API to parse the XML files as it works well with mixed content model and is language independent. This task is made easier since all our TEI encoded novels use a consistent tagging scheme. For each novel, we store the relevant parsed information into a Character object, which is later used for social network/graph analysis process. In this section we discuss how we extract information from our encoded novels using Chapter Level parsing. Paragraph level parsing basically follows similar procedure differing only in the size of context scope.

Our algorithm has three main steps. In the first step, we focus on the header section of the XML that contains information such as the title of the novel, which is extracted from the node 'title' (see Figure 6 for details). In the next step of our algorithm, we iterate through the node 'listPerson' and fetch the person list of the novel. Each <person> contains full name of the character (see Figure 7 for details). It is worth noting that some of the characters mentioned in the listPerson never appear in any direct conversation. We removed such characters from the list. In the final step, we fetch chapter specific information, where each chapter contains multiple paragraphs. We parse information within a <p> tag if it contains a quoted speech by a character.

To assign a speaker to the quoted text, we look for the pattern “character_[0-9]+” in the same text (see Figure 8 for details).

```
<teiHeader>
   <fileDesc>
       <titleStmt>
             <title> Pride and Prejudice </title>
             <author> Jane Austen </author>
         ……..
</teiHeader>
```

Figure 6: Novel Title Extraction from Encoded XML

```
<listPerson>
     <person xml:id=" character_2">
          <persName> Mr. Bennet </persName>
     </person>
     <person xml:id="character_7">
          <persName> Mrs. Bennet </persName>
     </person>
     ………..
</listPerson>
```

Figure 7: Character List information extraction from Encoded Xml

```
<div>
      <p> Chapter 1 </p>
      <p> “My dear character_2,” said his lady to him one day, “have you heard that Netherfield
                                                                         Park is let at last?” </p>
      <p> character_2 replied that he had not. </p>
     ……………
</div>
```

Figure 8: Speaker, Interaction extraction from Encoded Xml

The parsed information is stored in a Character object that has attributes to store the total number of words spoken by a character to all other characters, the total number of interactions between each two characters and the total number of segments in which it actively participated in a conversation, as mentioned in Figure 9. After building the Character object, we pass the list of characters to next component, Social Network Generator.

```
public class Character {

    String Name;
    ArrayList<String> segments;
    int no_of_interaction;
    int no_of_words_spoken;
    HashMap<String, ArrayList<Integer>> segmentInfo;
….
}
```

name - stores name of the Character.

segments – stores information about the chapter number or paragraph identifier in which the character appeared.

no_of_words_spoken - stores information about total number of words spoken by the character in the novel.

no_of_interaction - stores information about total number of interaction by the character in the novel.

segmentInfo - maps segments to number of words spoken by the character and number of interactions with other characters in that scene.

Figure 9: Character Object (Java)

3.4.2 Construct Network Graphs

Using the information generated by the Parsers, we create each novel's social network graph and then calculate social network metrics from the generated graph. These metrics allow us to compare the social network graphs created from our automatically tagged novels with those from the manually tagged novels. We could also compare automatically generated networks with manually created ones or compare different novels by similar authors. We use Gephi's API to generate the graph files. The Character object from the Parsers is used to calculate some graph metrics. Each entry of the Character object maps to a graph node. Once we have all the nodes identified, we use the segment information for each character to create an edge between this character and the remaining characters in the list, if they both appeared in the same segment.

For our non-directional graphs, we considered the total number of interactions between two characters as the weight of the edge. The output to the social network generator is gexf file. In figure 10 and 11, we present part of a generated gexf file. We chose gexf format as with this file format literary scholars can directly import the file into Gephi tool to visualize and perform various analysis on the graph. Once the basic structure of graph is ready, we compute the following graph metrics as shown in Table 2 using functions provided by Gephi.

Table 2: Metrics extracted from the novels. Here *g* represents a graph, *c* a character node in the graph, and *e* an edge between two character nodes in the graph.

Metrics Extracted from Text
1. No of edges = total number of edges of *g*
2. Total words = total number of words spoken by *c*
Metrics Extracted from Graph
3. Eigenvector Centrality = A measure of *c*'s importance in a network based on *c*'s connections.
4. Path Length = The average graph-distance between all pairs of nodes.
5. Average Degree = Sum of the degrees of all the nodes in the graph divided by the total number of nodes in the graph.

Some of the metrics we calculated were extracted from the novel itself, i.e., not generated by the network, e.g., the total number of words spoken by a character. We compute the graph metrics using Gephi's library. Node-specific metric such as *Eigenvector* capture information about a particular node in the graph. In contrast, Graph metrics such as Path Length and Average Degree capture information about the graph as a whole.

In order to be able to compare nodes in multiple networks, the Node metrics, we normalized the values by calculating the network centralization value using the following equation developed Newman (Newman, M.E.J. 2010):

$$C = \frac{\sum_i [c^* - c^i]}{\text{Max} \sum_i [c^* - c^i]}$$

where,

c* = maximum value for all the nodes in the graph

c^i = value of current node

The denominator represents the maximum of the summation over all the possible networks.

```
<node id="15" label="Mr. Bennet">
      <attvalues>
          <attvalue for="key" value="character_2"/>
          <attvalue for="words" value="1501"/>
          <attvalue for="total_interactions" value="23"/>
          <attvalue for="clustering" value="0.79"/>
          <attvalue for="eigencentrality" value="0.86"/>
          <attvalue for="degree" value="22"/>
       </attvalues>
    <viz:size value="20.0"/>
   <viz:position x="-584.0573" y="1552.123"/>
</node>
```

Figure 10: Sample output - part of gexf file (Node representation)

```
<edge id="141" source="9" target="15" label="Mrs. Bennet -- Mr. Bennet" weight="2113.0">
    <attvalues>
        <attvalue for="chapters" value="11"/>
        <attvalue for="character1" value="Mrs. Bennet"/>
        <attvalue for="character2" value="Mr. Bennet"/>
        <attvalue for="interaction" value="39"/>
        <attvalue for="character1_words" value="775"/>
        <attvalue for="character2_words" value="1338"/>
    </attvalues>
</edge>
```
Figure 11: Sample output - Part of gexf file (Edge representation)

4. RESULTS

4.1 Experimental Setup

Among the six novels of Jane Austen as shown in Table 3, we focused on one while developing our algorithms, “Pride & Prejudice”. We evaluated and trained our algorithms working exclusively with that novel. Once they were developed to our satisfaction, we evaluated the algorithms using the remaining five novels for testing.

Table 3: Dataset

Novel Name	Purpose
Pride & Prejudice	Experiment
Emma	Evaluation
Mansfield Park	Evaluation
Northanger Abbey	Evaluation
Persuasion	Evaluation
Sense & Sensibility	Evaluation

4.2 Experiment One: TEI Encoder Accuracy

4.2.1: Accuracy Based on Types of Tags

We first evaluated our TEI encoder by calculating the accuracy of the types of tags we used to annotate the novels against the tags that are used in the manually encoded novels. In Table 4, we present a sample evaluation of the various tags that are added for the novel "Pride & Prejudice" where the left column shows the tags that are used in the truth and the right column shows the tags used for encoding. Similarly, we calculated the correctness of the tag types that were added for all other novels of our study in Table 5. On average, we achieved around 80% accuracy in successfully implementing different types of tags over the plain, un-TEI encoded texts.

Table 4: Comparing types of tags used on "Pride & Prejudice" for truth vs encoder

Truth Tags	**TEI Encoder Tags**
<teiHeader>	<teiHeader>
<text>	<text>
<body>	<body>
<div>	<div>
<p>	<p>
<fileDesc>	<fileDesc>
<publicationStmt>	<publicationStmt>
<title>	<title>
<author>	<author>
<listPerson>	<listPerson>
<seg>	
<said>	
<floatingText>	
Total Tags = 13	**Total Tags = 10**
Accuracy 77%	

Table 5 : Tag accuracy based on type calculation for each novels of our study

Novel	**Accuracy**
Emma	72%
Mansfield Park	77%
Northanger Abbey	83%
Persuasion	83%
Pride & Prejudice	77%
Sense & Sensibility	83%
Average Accuracy	**80 %**

4.2.2: Accuracy Based on Total Number of Tags

We again evaluated the TEI encoder by calculating the precision, recall and accuracy of adding each tag with which we annotated the plain text to convert them to TEI- encoded html. In Table 6, we presented a sample calculation for the novel “Pride & Prejudice” where the correctness of each tag was measured by comparing the same with the manually encoded ones. We used false positive value as 100 for minor inner tags that are mainly considered in the header.

Table 6: Tagging statistics for “Pride & Prejudice” (Truth vs Encoder)

Tags	**Total count from Truth**	**Total Count from Encoder**
<teiHeader>	1	1
<text>	1	1
<body>	1	1
<div>	61	61
<p>	2184	2128
<head>	61	0

Table 6: Tagging statistics for “Pride & Prejudice” (Truth vs Encoder) (Cont.)

Tags	Total count from Truth	Total Count from Encoder
<fileDesc>	1	1
<publicationStmt>	1	1
<title>	1	1
<author>	4	3
<listPerson>	3	1

Here,

Precision = 0.96

Recall = 0.95

Accuracy = 91.3%

Similarly, we calculated the precision, recall and accuracy of adding each of these tags for all other novels in our study. Table 7 presents these values and their average. The overall average precision of correctly tagging the plain text of the novels is around 0.96, recall 0.92 and accuracy 92.1%.

Table 7: TEI Encoder precision, recall and accuracy for each novels of our study

Novel	Precision	Recall	Accuracy
Emma	0.95	0.94	89.4%
Mansfield Park	0.97	0.93	91.4%
Northanger Abbey	0.96	0.92	88.8%
Persuasion	0.94	0.85	83.4%
Pride & Prejudice	0.96	0.95	91.3%
Sense & Sensibility	0.98	0.94	92.1%
Average	**0.96**	**0.92**	**89.4 %**

4.3 Experiment Two: Character Identification

Our strategy to evaluate the proposed Character Identifier was to compare the similarity between the character names derived from our model and the character names from manually encoded novels. In table 8, we showed the comparison between the characters from the novel "Pride & Prejudice". Then, we measured the precision, recall and accuracy.

Table 8: Finding similarities between the character names of "Pride & Prejudice" (Truth vs Character Identifier)

Truth Value	Derived Value	Matched
Elizabeth Bennet	Elizabeth Bennet	Yes
Mrs. Bennet	Mrs. Bennet	Yes
Mr. Darcy	Mr. Darcy	Yes
Jane Bennet	Jane Bennet	Yes
Mr. Bennet	Mr. Bennet	Yes
Miss Bingley	Miss Bingley	Yes
Mr. Wickham	Mr. Wickham	Yes
Mr. Collins	Mr. Collins	Yes
Lady de Bourgh	Lady de Bourgh	Yes
Mr. Bingley	Mr. Bingley	Yes
Mrs. Gardiner	Mrs. Gardiner	Yes
Lydia Bennet	Lydia Bennet	Yes
Charlotte Lucas	Charlotte Lucas	Yes
Colonel Fitzwilliam	Colonel Fitzwilliam	Yes
Mr. Gardiner	Mr. Gardiner	Yes
Mrs. Reynolds	Mrs. Reynolds	Yes
Sir Lucas	Sir Lucas	Yes
Mary Bennet	Mary Bennet	Yes
Mrs. Hurst	Mrs. Hurst	Yes
Maria Lucas	Maria Lucas	Yes
Mrs. Hill	Mrs. Hill	Yes
Mrs. Phillips	Mrs. Phillips	Yes
Lady Lucas	Lady Lucas	Yes
Mr. Hurst	Mr. Hurst	No
Mr. Collins	Lewis de Bourgh	No
Mr. Denny	Miss Darcy	No
Kitty Bennet	Mrs. Long	No
	Mrs. Collins	Unavailable
	George Wickham	Unavailable
	Mr. Jones	Unavailable

Here,

Precision = **0.77**,

Recall = **0.85**

Accuracy = **68%**

Similarly, we calculated these values for all other novels as shown in Table 9. “Persuasion” had the best precision value whereas “Northanger Abbey” scored the best recall and accuracy. On average, we had 69% accuracy for this character identifier model. While building the character dictionary, we ended up with unwanted names on some occasions. For example, in the novel Emma, “Donwell Lane” were recognized as a person while actually it belongs to a location name. We also extracted names such as Mr. Dixon and Miss Smith from Emma as they are mentioned many times throughout the text, however they are not documented as character in the manually encoded corpus since they never actually speak but are merely referred to. As a result, our extracted social networks produced more characters compared to the truth networks in most cases. If we filtered out some of these minor characters from our list, i.e., those who never speak, the accuracy would be a lot higher. Moreover, due to the limitation of NER tagger, some of the PERSON entities were left unrecognized. The tagger also performs poorly while identifying prefix or suffix of a name. We had to pay special attention to this issue since many characters in Jane Austen’s novels are addressed with prefix followed by last name.

Table 9: Precision, Recall and Accuracy in character identification for the novels in our study

Novel	**Precision**	**Recall**	**Accuracy**
Emma	0.75	0.84	66%
Mansfield Park	0.76	0.85	67%

Table 9: Precision, Recall and Accuracy in character identification for the novels in our study (Cont.)

Novel	Precision	Recall	Accuracy
Northanger Abbey	0.80	0.89	73%
Persuasion	0.83	0.83	72%
Sense & Sensibility	0.73	0.84	64%

Average Precision	Average Recall	Average Accuracy
0.77	**0.85**	**69%**

4.4 Experiment Three: Context Size

Our thesis implements two strategies based on context size (chapter scope and paragraph scope) to observe character interactions. To check the accuracy of our methods, we conducted one experiment that is based on network statistics and another evaluation involved measuring similarity (edit distance) between the networks constructed from manually encoded novels versus our constructed graphs which can be categorized as follows:

1. Comparing chapter versus paragraph level interactions based on network statistics

2. Comparing chapter versus paragraph level interactions based on edit distance

4.4.1 Accuracy Based on Network Statistics

This experiment involved observing the network statistics we derived from each novel for both chapter and paragraph level parsers. Evaluating a social network is tough since there does not exist any established, quantitative metric for the evaluation. This suggests that the evaluation essentially be of a qualitative nature which largely depends on the person who investigate the network and judge the result; hence, the notion of importance is ambiguous here. In our thesis, we compute graph metrics such as Average Degree, Eigenvector, and Path Length along with total number of words and edges to capture information about the graph as a whole. A summary of each novel's metrics for both network construction strategies is given in figure 12, 13 and 14.

Novel	**Edges**	**Words**	**Eigenvector**	**Path Length**	**Avg Degree**
Emma	273	75682	0.265	1.328	0.352
Mansfield Park	130	59054	0.241	1.316	0.351
Northanger Abbey	68	27974	0.404	1.569	0.493
Persuasion	119	26958	0.449	1.530	0.573
Pride & Prejudice	152	47959	0.475	1.598	0.644
Sense & Sensibility	93	51239	0.396	1.511	0.509

Figure 12: Social network metrics generated from manually encoded novels

Novel	**Edges**	**Words**	**Eigenvector**	**Path Length**	**Avg Degree**
Emma	361	57633	0.286	1.357	0.379
Mansfield Park	231	55988	0.125	1.163	0.178
Northanger Abbey	125	19638	0.216	1.269	0.301
Persuasion	173	30175	0.138	1.176	0.195
Pride and Prejudice	273	21467	0.322	1.413	0.441
Sense & Sensibility	199	28339	0.274	1.337	0.366

Figure 13: Social network metrics generated by Chapter Level Parser

Novel	**Edges**	**Words**	**Eigenvector**	**Path Length**	**Avg Degree**
Emma	217	58067	0.286	1.657	0.573
Mansfield Park	155	56205	0.125	1.475	0.299

Figure 14: Social network metrics generated by Paragraph Level Parser

Novel	**Edges**	**Words**	**Eigenvector**	**Path Length**	**Avg Degree**
Northanger Abbey	94	19820	0.216	1.552	0.611
Persuasion	117	30187	0.138	1.502	0.490
Pride & Prejudice	130	21520	0.322	1.919	0.512
Sense & Sensibility	113	28339	0.274	1.641	0.406

Figure 14: Social network metrics generated by Paragraph Level Parser (Cont.)

In Table 10, we presented a comparative analysis between the Chapter and Paragraph Level Parser based on the data on Figure 12, 13 and 14. We can see that the Paragraph Level Parser clearly has higher overall accuracy than Chapter Level Parser on all novels except for Emma.

Table 10: Comparing chapter versus paragraph level interactions based on network statistics

Novel	**Chapter Level Parser Average Accuracy**	**Paragraph Level Parser Average Accuracy**	**Winner**
Emma	**85%**	**74%**	Chapter Level Parser
Mansfield Park	**72%**	**88%**	Paragraph Level Parser
Northanger Abbey	**67%**	**81%**	Paragraph Level Parser
Persuasion	**67%**	**93%**	Paragraph Level Parser
Pride & Prejudice	**64%**	**74%**	Paragraph Level Parser
Sense & Sensibility	**66%**	**77%**	Paragraph Level Parser

4.4.2 Accuracy Based on Edit Distance

This experiment involved calculating the edit distance between the social network graphs of each novel for both chapter and paragraph level parsers. In the area of pattern analysis, inexact graph matching plays a very crucial role. Graph edit distance (GED) is the base of inexact graph matching, i.e., it measures the similarity between pairwise graphs error-tolerantly. For this purpose, we used NetworkX which is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks. The default algorithm is sub-optimal for some graphs and the problem of finding the exact GED is NP-hard, so it is often slow (Hagberg et al., 2008).

The “graph_edit_distance” method from NetworkX package assumes that both graphs, g1 and g2, have equal number of nodes. To fulfill this requirement, we removed the extra nodes from our generated graphs using Gephi software so that both graphs have similar number of character nodes before they are passed to the “graph_edit_distance” method. We achieved this by filtering the character nodes based on interactions. For instance, the total character nodes in the manually encoded “Pride & Prejudice” is 28 whereas the number is 31 for the network generated from the Character Level Parser. We first sorted the characters in descending order by the number of their interactions and then removed the 3 characters with lowest interaction value. After this modification, we pass the gexf files to the “graph_edit_distance” method to calculate the edit distance. Figure 15 and 16 show the network graphs generated from the gexf files using Gephi.

The NetworkX package of Python reads the graph properties from our generated gexf files while calculating the edit distance. Since it only reads certain versions of gexf files, we had to modify the version number located in the gexf file header in order to make the graph files compatible to the Python library.

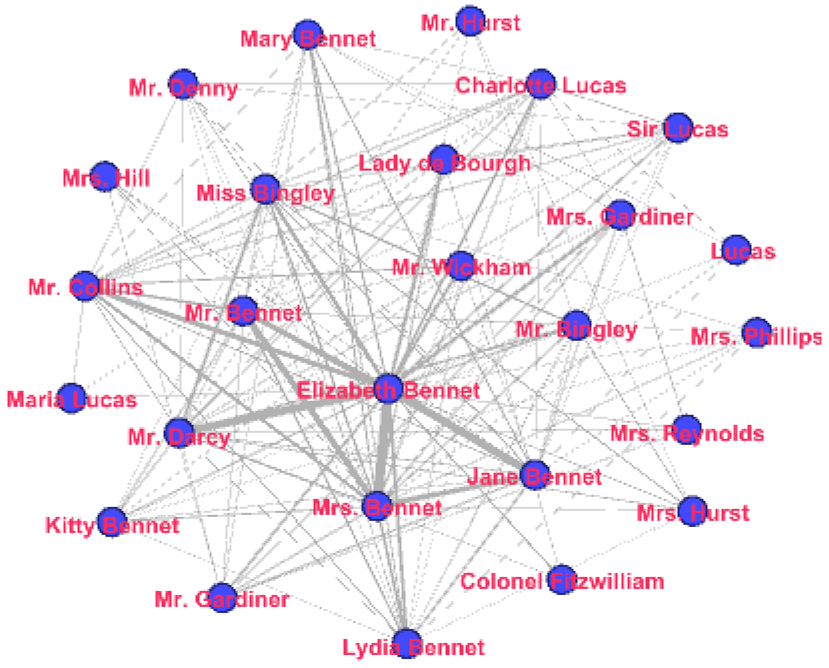

Figure 15: Truth network graph of Pride & Prejudice

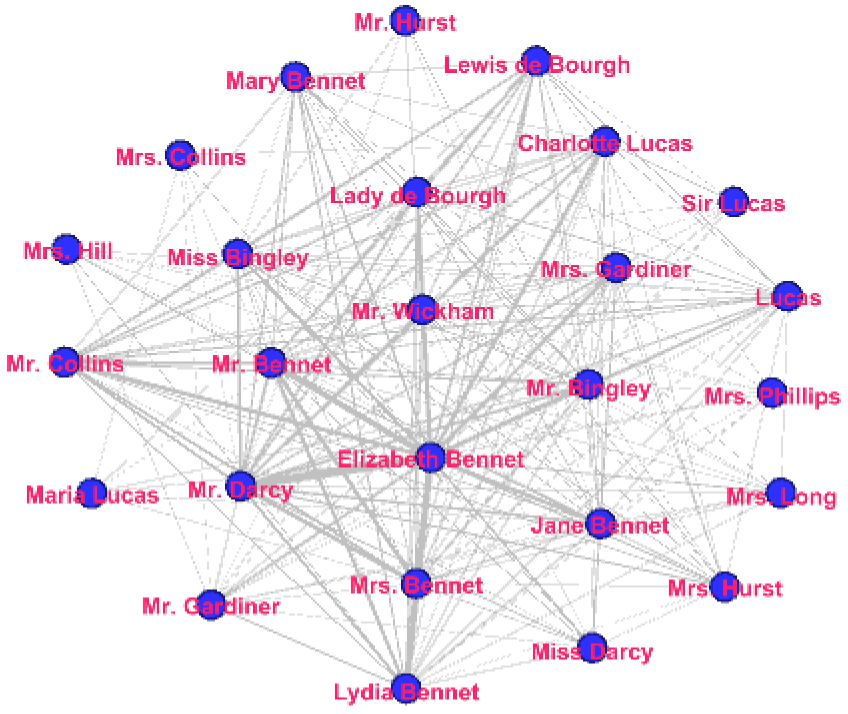

Figure 16 : Network graph of Pride & Prejudice generated by Character Level Parser

Similarly, we filtered the character nodes generated by Paragraph Level Parser based on interaction and generated gexf files using Gephi. Figure 17 shows the network graph generated from the corresponding gexf file using Gephi.

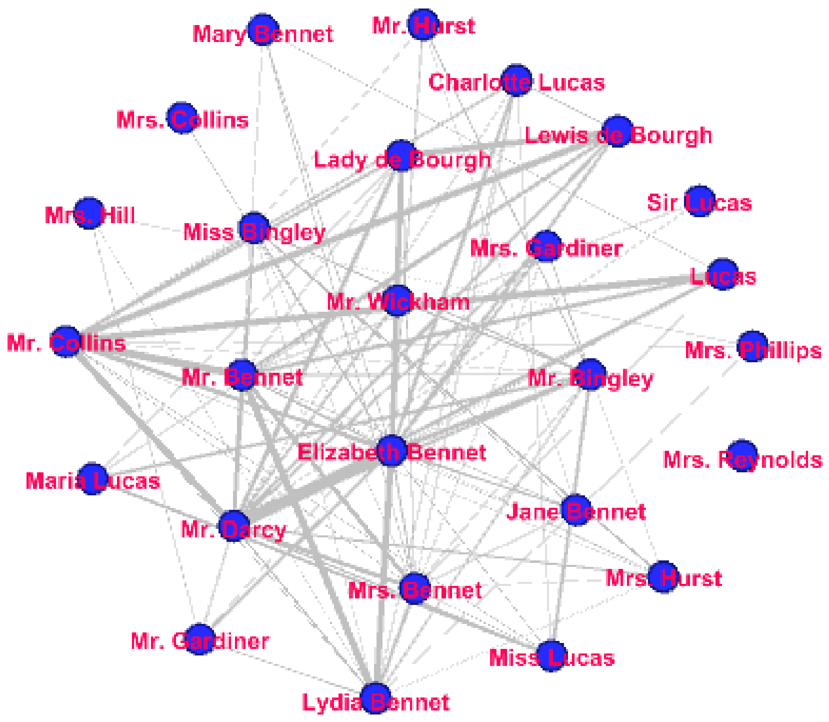

Figure 18 : Network graph of Pride & Prejudice generated by Paragraph Level Parser

In Table 11, we have shown the results of edit distance calculation for each pair of graphs categorized by parsing methods (context size) based on character interaction. We observe that Paragraph Level Parser outperforms Chapter Level Parser on all six novels.

Table 11: Comparing chapter versus paragraph level interactions based on edit distance

Novel	**Edit distance**		**Winner**
	Chapter Level Parser	**Paragraph Level Parser**	
Emma	30	22	Paragraph Level Parser
Mansfield Park	20	14	Paragraph Level Parser
Northanger Abbey	20	16	Paragraph Level Parser
Persuasion	24	12	Paragraph Level Parser
Pride & Prejudice	22	20	Paragraph Level Parser
Sense & Sensibility	16	12	Paragraph Level Parser

5. CONCLUSION

This thesis successfully implements several approaches to extract social networks from unstructured prose fiction. Our process involves automatic encoding of plain literary text into TEI formatted xml, character name identification, conversation and co-occurrence detection, and social network construction. We successfully evaluate our proposed methods using several strategies. The accuracy of text encoder and character identifier are measured by calculating precision and recall whereas the extracted social networks are evaluated by network statistics and also by measuring the graph similarity (edit distance) with the manually constructed (truth) networks.

The text sources that are examined consist of six novels from Project Gutenberg by author Jane Austen. We design an automatic text encoder which serves the purpose of tagging plain, unformatted literary text, and then converting the text to a TEI-compliant XML file in such a way that this approach can be broadly applicable. We also propose a method to identify characters from literary prose which comprises of tasks such as generating a character lexicon where each entry belongs to a single character and the corresponding aliases to that character.

One of the primary goal of our thesis is to determine appropriate context size to use when building a social network based on character interactions. We describe two strategies to detect character co-occurrences, based on the context size of the text. The Chapter Level Parser extracts information using chapter scope to determine which characters were speaking to whom and total number of interactions among those characters. Paragraph Level Parser extracts similar information, but within the scope of a paragraph. These co-occurrences are then used to construct a weighted undirected network representation of the novel. To check the accuracy of our methods for extracting social networks, we conduct evaluations based on network statistics and edit distance. Our findings suggest that the choice of context size is non-trivial and can have a substantial influence on the resulting networks. Overall, the paragraph level interaction approach provided more accurate result.

For future work, it would be very beneficial to implement a pronoun resolution system. Since literary texts contain a large number of pronouns to represent the named entities, resolving what those pronouns refer to would certainly boost up the overall accuracy of the constructed network. Considering that our methods are highly extensible, future work applying these parsers can be extended for literary analysis of not only TEI encoded novels, but also, unformatted plain literary text, written by different authors, as well as written in different languages.

REFERENCES

1. F. Moretti. Network Theory, Plot Analysis. New Left Review, 68:80–102, 2011.

2. Siobhán Grayson, Karen Wade, Gerardine Meaney, Jennie Rothwell, Maria Mulvany, and Derek Greene, 2016. Discovering structure in social networks of 19th century fiction. In Proceedings of the 8th ACM Conference on Web Science (WebSci '16). ACM, New York, NY, USA, 325-326.

3. S. Grayson, K. Wade, G. Meaney, and D. Greene, "The Sense and Sensibility of Different Sliding Windows in Constructing Co-occurrence Networks from Literature," IFIP Advances in Information and Communication Technology Computational History and Data-Driven Humanities, pp. 65–77, 2016.

4. Manisha Shukla, Susan Gauch and Lawrence Evalyn, 2018. Theatrical Genre Prediction Using Social Network Metrics. In 10th International Conference on Knowledge Discovery and Information Retrieval, Seville, Spain.

5. R. Alberich, J. Miro-Julia, and F. Rossello. Marvel Universe looks almost like a real social network. arXiv:cond-mat/0202174, (February 2008):14, 2002.

6. P. M. Gleiser, "How to become a superhero," Journal of Statistical Mechanics: Theory and Experiment, vol. 2007, no. 09, 2007.

7. D. K. Elson, N. Dames, and K. R. McKeown. Extracting social networks from literary fiction. In Proc. 48th Meeting of Assoc. Comp. Ling., pages 138-147, 2010.

8. A. Agarwal, A. Corvalan, J. Jensen, and O. Rambow. Social network analysis of Alice in Wonderland. In Proc. Workshop on Comp. Linguistics for Literature, pages 88-96, 2012.

9. P. A. Jayannavar, A. Agarwal, M. Ju, and O. Rambo. Validating literary theories using automatic social network extraction. Proc. 4th Workshop on Comp. Linguistics for Literature, pages 32-41, 2015.

10. "XML," Wikipedia, 31-Jul-2019. [Online]. Available: https://en.wikipedia.org/wiki/XML [Accessed: 01-May-2019].

11. "Text Encoding Initiative," Wikipedia, 07-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Text_Encoding_Initiative. [Accessed: 02-May-2019].

12. "Named-entity recognition," Wikipedia, 30-Jul-2019. [Online]. Available: https://en.wikipedia.org/wiki/Named-entity_recognition. [Accessed: 09-Jun-2019].

13. J. R. Finkel, T. Grenager, and C. Manning, "Incorporating non-local information into information extraction systems by Gibbs sampling," Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics - ACL 05, 2005.

14. “spaCy · Industrial-strength Natural Language Processing in Python,” · *Industrial-strength Natural Language Processing in Python*. [Online]. Available: https://spacy.io/. [Accessed: 05-Jun-2019].

15. “Apache OpenNLP,” Brand. [Online]. Available: https://opennlp.apache.org/. [Accessed: 04-Jun-2019].

16. “Graph Visualization Software,” Graphviz. [Online]. Available: http://graphviz.org/. [Accessed: 12-Jun-2019].

17. T. de P. Peixoto, “tool,” graph. [Online]. Available: https://graph-tool.skewed.de/. [Accessed: 01-Aug-2019].

18. “JUNG,” JUNG. [Online]. Available: http://jung.sourceforge.net/. [Accessed: 12-Jun-2019].

19. “The Open Graph Viz Platform,” graph exploration and manipulation. [Online]. Available: https://gephi.org/. [Accessed: 01-Jun-2019].

20. “Gephi Toolkit,” graph exploration and manipulation. [Online]. Available: https://gephi.org/toolkit/. [Accessed: 01-Jun-2019].

21. Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. "Algorithms for graph similarity and subgraph matching." In Proc. Ecol. Inference Conf, vol. 17. 2011.

22. Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. “Similarity flooding: A versatile graph matching algorithm and its application to schema matching”. In 18th International Conference on Data Engineering (ICDE 2002), 2002.

23. Glen Jeh and Jennifer Widom. “SimRank: a measure of structural-context similarity”. In Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD ’02, pages 538–543, New York, NY, USA, 2002. ACM.

24. L Zager and G Verghese. Graph similarity scoring and matching. Applied Mathematics Letters, 21(1):86–94, 2008.

25. “Graph edit distance,” Wikipedia, 22-Feb-2019. [Online]. Available: https://en.wikipedia.org/wiki/Graph_edit_distance. [Accessed: 05-AJun-2019].

26. “Books by Austen, Jane,” Project Gutenberg. [Online]. Available: http://www.gutenberg.org/ebooks/author/68. [Accessed: 01-Aug-2018].

27. “Austen Said:” Austen. [Online]. Available: http://austen.unl.edu/. [Accessed: 01-Sep-2018].

28. Newman, M.E.J. 2010. Networks: An Introduction. Oxford, UK: Oxford University Press.

29. Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008.