5-2020

Shakespeare in the Eighteenth Century: Algorithm for Quotation Identification

Marion Pauline Chiariglione
University of Arkansas, Fayetteville

Shakespeare in the Eighteenth Century: Algorithm for Quotation Identification

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

by

Marion Pauline Chiariglione
IUT Dijon, University of Burgundy
Bachelor of Science in Computer Science, 2017

May 2020
University of Arkansas

This thesis is approved for recommendation to the Graduate Council

---

Susan Gauch, Ph.D.
Thesis Director

Qinghua Li, Ph.D.
Committee member

Khoa Luu, Ph.D.
Committee member

Abstract

Quoting a borrowed excerpt of text within another literary work was infrequently done prior to the beginning of the eighteenth century. However, quoting other texts, particularly Shakespeare, became quite common after that. Our work develops automatic approaches to identify that trend. Initial work focuses on identifying exact and modified sections of texts taken from works of Shakespeare in novels spanning the eighteenth century. We then introduce a novel approach to identifying modified quotes by adapting the Edit Distance metric, which is character based, to a word based approach. This paper offers an introduction to previous uses of this metric within a multitude of fields, describes the implementation of the different methodologies used for quote identification and then shows how a combination of both Edit Distance methods can help achieve a higher accuracy in quote identification than any one method implemented alone with an overall increase of 10%: from 0.638 and 0.609 to 0.737. Although we demonstrate our approach using Shakespeare quotes in eighteenth century novels, the techniques can be generalized to locate exact and/or partial matches between any set of text targets in any corpus. This work would be of value to literary scholars who want to track quotations over time and could also be applied to other languages.

Acknowledgements

I would like to thank Dr. Susan Gauch, for giving me the opportunity to continue working with her on this project started when I was interning in her laboratory. I would also like to thank her for guiding me through the project and helping me see what we could achieve with this work by being my advisor during these years. I am also grateful to my committee, Dr. Qinghua Li and Dr. Khoa Luu for accepting to review and provide feedback on my thesis.

I would also like to thank Dr. Kevin Labille for being the point of contact between Dijon, France and Fayetteville, Arkansas and giving me the opportunity to come study in the United States of America. I am also thankful to him for helping me get situated and get my bearings in the research environment.

Finally, I would like to express my deepest gratitude to my family and friends for their support throughout these years abroad.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivations

1.1.1 Linguistic motivation

Before and well into the eighteenth century, quoting an author in one's novel was not a very popular practice. Authors would take sentences and paragraphs from other works of literature without attribution, selling and distributing the work as their own. Some would go as far as fully copying the work and extending or modifying parts of it, then republishing it without mentioning the source of the material [1]. This slowly started to change at the beginning of the eighteenth century when footnotes began to appear in novels. Shakespeare was one of the most quoted authors, although as mentioned previously, his quotations were not always exact, either due to the author's error or a deliberate misquote on the part of the novel's character [2]. It is for this reasons, that it is hard to find Shakespeare quotes in novels from that century (and sometimes in following centuries), without being knowledgeable and familiar with a wide range of his work and the common text modifications that were present in those texts. Literary scholars often want to see how and where an author is quoted. However, due a variety of issues, this is not easily done, even when the works are available for online searching. The following section will describe common text modifications, or errors as we label them here.

1.1.1.1 Types of errors

When texts are digitized, a variety of errors can be unintentionally introduced, e.g., adding a space where there should not be one thus splitting a word, i.e., spelling errors. But some of those errors are not due to the digitization of text as they are just modifications of the quote by the author itself, i.e., paraphrasing errors. The following sections will show examples of those errors and the original quote it corresponds to.

Spelling errors

- Letter errors: The letter 's' being replaced by the 'long s' symbol ('ſ'), an old form of said letter.

Example: "the rattling tongue of **faucy** and **audaciouſ** eloquence"

Quote: "the rattling tongue of **saucy** and **audacious** eloquence"

- Splitting errors: Words being split in two other words, changing the meaning or sense of a sentence.

Example: "and all the men and women **me rely** players"

Quote: "and all the men and women **merely** players

Paraphrasing errors

- Addition errors: New words being added to the original quote.

Example: "like patience **sitting still** on a monument smiling at grief"

Quote: "like patience on a monument smiling at grief"

- Deletion errors: Original words being removed from the quote.

Example: "this above all [**missing words**] own self be true"

Quote: "this above all to thine own self be true"

- Synonym errors: Words being replaced by synonyms or totally different words in the original quote.

Example: "angels and ministers of **dignity** defend us"

Quote: "angels and ministers of **grace** defend us"

For easier understanding of the paper, the following mnemonic short names are attributed to each type of error: LongS and Split for the corresponding spelling errors, and Insert, Remove and Replace for the respective paraphrasing errors.

1.1.2 Computer Science motivation

With the rapid expansion of the Internet, and the amount of information it encompasses, data and text mining have rapidly become highly used tools. One is used for extracting data from structured input, e.g. databases, to generate new information and the other, used in this research, for extracting data from unstructured input, e.g., scientific texts. One of the techniques of text mining on which this research is focused on, is called the edit distance, a string similarity measure. Since this method is mainly used on small datasets and on a character basis, we wanted to alter it to instead use it on words and to use it in a big data environment; where we would measure its efficiency and accuracy in identifying quotes in a large dataset.

This method was chosen over a Deep Learning approach as there is no publicly available annotated dataset for our problem: identifying textual reuse within a large corpus of text. The dataset would need to have a training section where one would have identified by hand the quote that had been reused in a novel and map it to the actual Shakespeare quote. As this type of dataset is not yet publicly available to our knowledge, we chose to expend research in Natural Language Processing with the Edit Distance method.

1.2 Goals

To address the issues and motivations discussed above, we have identified two goals for our research into quotation tracking:

- Goal 1: Efficiently and effectively identifying an exact quote in a text corpus.
- Goal 2: Efficiently and effectively identifying a paraphrased quote in a text corpus.

The following chapters describe previous work related to our problem, the approaches taken, the evaluation of our algorithms and the conclusions drawn.

2 Related Work

This chapter explores existing research related to search engines, text similarity, sequence matching, and the Edit Distance metric applied to various fields.

2.1 Search engines

Our core problem is to find quotes within a relatively large corpus. This is the basic goal of search engines. By treating a quote as a query, a search engine can locate text passages in a corpus that are relevant to the quote, based on occurrences of words in the quote within the passage. They have limitations, though. By default, they do not require all words to occur in the passage and they do not take word order into account. Alternatively, the quote can be entered as a phrase and exact matches can be found. Although neither of these capabilities is exactly what we want to find partial quotes, we were able to employ search engine software to create the ground truth for our quote tracking. A search engine, by definition, is a natural language information retrieval system implemented with the intent to find relevant data to the user's query within a set of information stored on a server. The search results are usually presented to the user in a ranked list from the best match to the least and are usually called hits [3]. Search engines predate the debut of the internet itself: they weren't always Google, Yahoo, etc, allowing you access to thousands upon thousands of pages of information. For example, the WHOIS directory allowed users to query a single server which acted as a directory retrieving relevant information about people and entities, making it the first information retrieval tool of it's kind [4]. Our own search engine is accessible via the Internet [1] and queries a specific database containing our curated dataset. It was built with the open source information retrieval library, Lucene, made by Apache (see Section 3.1 for more details about Lucene).

[1] http://text.csce.uark.edu/clip/qt/

2.2 Text similarity

As explained in Section 1.1.1., authors previous to the eighteenth century did not give credits to authors whose text they used within their own work. Duhaime mentions "that the eighteenth century witnessed a revolution" in terms of citation and credit attribution and that "those writing in the seventeenth and early eighteenth centuries endorsed the model of *imitatio*" [5], literary method of emulating, adapting, reworking and enriching a source text by an earlier author [6]. Duhaime's work, porting on Eliza Haywood's novel Betsy and Thoughless, is highly relevant to the problem presented in this paper: both are trying to identify borrowed passages present in novels from the Eighteenth century and introduce novel algorithmic solutions used to identify text similarity within a large corpus of texts. Our approach in trying to identify reused passages within text, is based on the Edit Distance metric. No known works has been identified to have used this method, most works focusing on n-gram and sliding window methods (Duhaime, 2016 and Büchler, 2010) followed by a graph representation of textual reuse, linking two text passages if found similar [7]. A method that is closer to our metric, is the frequency method used by Bernstein [8], as the computation for it does include a resemblance to the edit distance, but instead of being between the target quote and the extracted one, it is the distance for each of those within their own body of text. Since no known work within Computer Science and Literacy have tackled this problem using the edit distance metric, we decided to look into other fields to identify works that have used said method in a context of sequence matching. This is presented in the following section.

2.3 Sequence matching

2.3.1 Edit distance

At the core of our algorithm for identifying modified quotes and text reuse is sequence matching, and more precisely string matching. Surprisingly enough, this is something widely common in the field of Biology for comparing multiple strands of DNA since those are only

permutations of the letters A, C, G, and T. As Sankoff mentions, "genes evolve largely through the process of nucleotide substitution, insertion and deletion" [9], thus why papers related to this subject in Biology use the Edit Distance metric, as it relies on calculating a relevance score based on insertions, deletions and substitutions. It has also been proven to be effective and fast in generating accurate results, as Wang et al. show in the privacy implemented version of the metric, as " it would take [...] less than 200 minutes to search through 1 million breast cancer patients [...], based on edit distances between their genomes of lengths about 75 million nucleotides each." [10].

As such important work has been done in the Biology field regarding the use of the edit distance for problem solving, we want to provide the same drive for resolve in this paper, proposing a solution for the problem of textual reuse in novels from the eighteenth century through edit distance computation (see Section 3.2.2.2 for more details on its calculation).

2.3.2 Alignment-free models

Other sequence matching methods were considered before choosing the edit distance, all falling under the category of alignment-free models and are also widely used within the Biology field for DNA sequencing. According to Bao, Yuan and Bao, the alignment-free method converts any sequence "into a feature space based on words' probability distribution rather than directly matches strings", thus only employing word frequency information [11]. This type of method does not seem to be optimized to work for our problem. Let us show why this is not an appropriate method for our quote identification problem by a brief explanation of how the alignment-free method, k-tuple, is implemented. Let's take the following sequence:

$$S = ATCGTAACATTATGC \tag{2.1}$$

Then, let's consider the following k-words when k = 2:

$$AT, TC, CG, GT, TA, AA, AC, CA, AT, TT, TA, AT, TG, GC \tag{2.2}$$

Those k-words can be arranged into a table of words occurrences, counting their frequency and storing their start offset, or indices within the sequence. Then going over this same process with a query-sequence, we would compare both their table of k-words occurrences and count the number of time they would co-occur (i.e., start at the same index). The main problem with this method is that it works finding exact matches within k-words, and would not be able to find any quote that has for example the "s" characters replaced with "f": "the rattling tongue of saucy and audacious eloquence" and "the rattling tongue of **faucy** and **audaciouf** eloquence" would never be matched together. If we were to consider taking this approach to a word level instead of a character one, we would still be faced with the same problem since it would not be able to detect a paraphrased quote with synonyms added in. We would have to take $k = 1$ and compare those k-words together which compares to implementing the edit distance. Another problem with this method is that since it only relies on the words' frequency it wouldn't work on queries with common words like "to be or not to be" and "we know what we are". It is for those reasons that we decided to implement the edit distance metric within our algorithm instead of an alignment-free model.

In the following chapters and sections, we will explain our Approach (i.e., goals defined, algorithms implementation, etc.), the Evaluation of our proposed solution on a currated dataset including the validation of the solution on unseen data and the Conclusions drawn and possible future work.

3 Approach

We tackled two goals: first finding exact matches and then extending our approach to look for partial matches. In order to efficiently identify exact matches, we employed an open source information retrieval library made by Apache, Lucene[1]. The following sections will present the general approaches taken to tackle our two goals.

3.1 Goal 1: Efficiently and effectively identifying an exact quote in a text corpus

The following sections will explain in details our algorithm to identify and extract quotes from a text corpus.

3.1.1 Pre-processing

Before going into the implementation of the different algorithms chosen to address our problem, and while planning out the project, one of the problems we encountered was in relation to the actual content of the books. Since most of the novels dating from the eighteenth century are in the public domain, they have additional text before and after the actual content of the book that is not relevant to our data mining research (see Figure 3.1 below).

Thus, we chose to work with XML files to be able to easily manipulate the text inside the novels. XML stands for Extensible Markup Language and is a tool for storing and transporting data. It is very much like HTML in the sense that it uses tags (i.e., $< p >, < body >$, etc.) but where they differ is in their goal: HTML was created to display the data whereas XML was created to carry the data. Another point to our advantage is that XML does not use predefined tags. The author of the XML file is the one to define the tags used; thus, is the author wants to use tags resembling the following: $< title >, <$ $date >$ *or* $< author >$, it is completely permitted and will have no impact in the way the file is viewed or analyzed [12]. In our case, this is very useful since our main goal is to mine

[1]https://lucene.apache.org/

The Project Gutenberg EBook of Northanger Abbey, by Jane Austen

This eBook is for the use of anyone anywhere at no cost and with almost no restrictions whatsoever. You may copy it, give it away or re-use it under the terms of the Project Gutenberg License included with this eBook or online at www.gutenberg.org

Title: Northanger Abbey

Author: Jane Austen

Release Date: January 21, 2010 [EBook #121]
Last Updated: March 10, 2018

Language: English

Character set encoding: UTF-8

Figure 3.1: Part of the input file containing the public domain text not wanted

specific information from the input source.

There were two distinct parts in our algorithm for the pre-processing of the quotes and novels. The first part creates an interlingua, i.e., modifies some of the tags present in the novels with ours in order to be able to easily access the content of the book and its title. To implement that we used Java Properties and XPath expressions to store the path to the different parts of the document we wanted to change. Then, using DOM and this newly created Java Properties file, we were able to replace for example the tag $< text >$ which is very generic to our own tag $< CONTENT >$ which is more specific. This allowed us to use a Java Object called a Scanner that reads a file until it encounters a specified delimiter, which here was the newly created tag, and allowed us to extract the true content of the novel, without the public domain text. Wanting to keep the quote in its context, we were able to use this new interlingua to extract the exact number of bytes read before the actual start of the novel, allowing us to know the exact position of the beginning of the book, thus the exact position of the quote searched for.

After having created the interlingua, the second part of the pre-processing was to do

```
<change>
<date>2008-02</date>
<label>pfs</label>Batch review (QC) and XML conversion</change>
</revisionDesc>
</teiHeader>
<CONTENT>
<body>
<div type="text">
<pb facs="tcp:27698:1" rendition="simple:additions"/>
<!-- PDF PAGE 7 -->
<head type="illustration">
<figure>
<head>HONI SOIT QVIMAL Y PENSE</head>
<figDesc>royal blazon or coat of arms</figDesc>
</figure>
</head>
```

Figure 3.2: XML file containing the interlingua CONTENT

```
titlePart=//teiHeader/fileDesc/titleStmt/title
contentPart=//text
datePart=//teiHeader/fileDesc/sourceDesc//date
```

Figure 3.3: Configuration file containing XPath expressions

the actual mining of the needed information to give to the search engine. With the same tools used in the first part of the pre-processing, we iterated over the XML tree in each of the input files and extracted and stored in properties files: the title, the content, the publication date and the start offset of the book. Those generated output files were the ones given as input to the search engine and used to build the search index.

Figures 3.2, 3.3 and 3.4 respectively show an example of the interlingua, of the XPath expressions to retrieve the content needed and of the output generated.

3.1.2 Text indexing

At the core of every search engine is a powerful data structure called reverse indexing. Because in the index we have the word with each of the document it appears in and its

```
title=Northanger Abbey
CONTENTtagReplaced=text
content= <front TEIform="front"> <titlePage TEIform="titlePage"> <docTitle TEIform="do [...]
startOffsetCONTENT=3149
date=1818
```

Figure 3.4: Output file containing the content needed to build the search engine

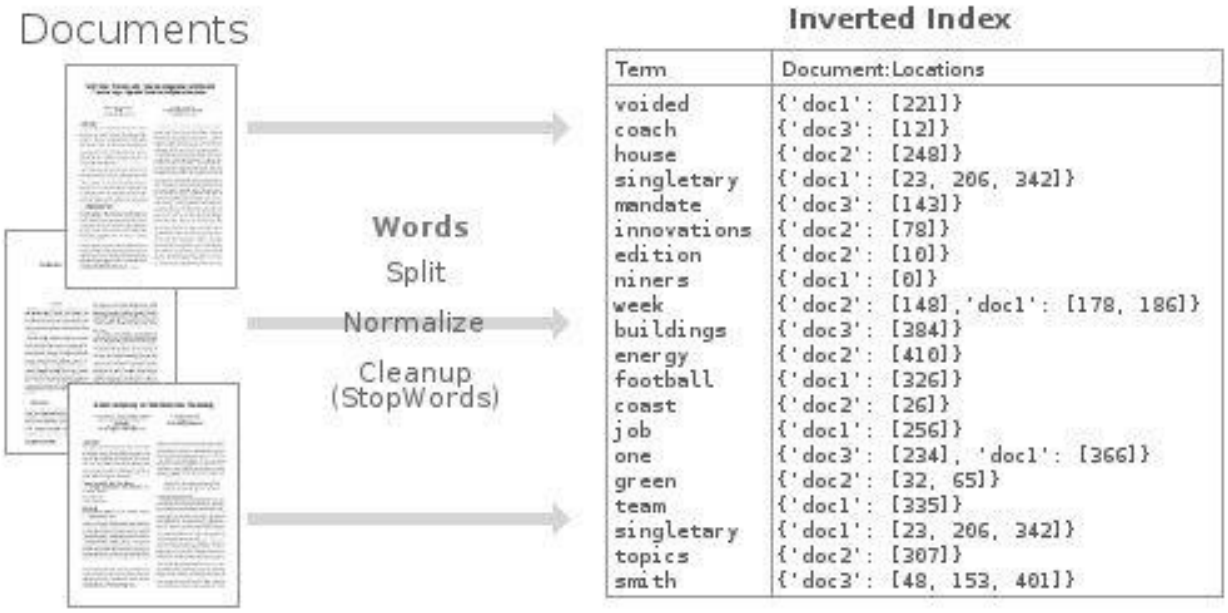

Figure 3.5: Example of an inverted index.
http://th30z.blogspot.com/2010/10/python-inverted-index-for-dummies.html

positions, it allows the search time to be significantly faster. Thus, we decided to use a tool called Lucene by Apache that has the reverse indexing at the core of its implementation. It indexes documents and perform queries on said index returning a ranked list of documents that matched the query. Using Lucene has a lot of advantages, it possesses tools to analyze, parse and index textual information (e.g. stemming, synonyms, tokenization, etc.) in multiple languages, it also has a very good similarity function (to compare the search query against each document), at the heart of which are the Cosine Similarity (similarity between the query vector and document vector) and Inverse Term/Document frequency (whether a term is common or rare in a given document). This results in a fast and accurate search outcome with very little tweaking required afterwards. Figure 3.5 shows an example of an inverted index.

```
public class LuceneDocument {

   //VARIABLES
   private Field content;
   private Field title;
   private TextField date;
   private TextField offsetContent;
   private TextField fileName;

   [...]

}
```

Figure 3.6: Java class implementing a Lucene Document with fields as variables

In order to create the index, Lucene needs two elements: the Documents and their Fields. These are two classes in Lucene that represent a searchable item. Thus here, each Lucene Document created corresponds to one book we want to index and has multiple fields. The first field contains the whole content of the book, while the rest of the fields contain the following: the title of the book, the date it was published, the start offset of the content in the input file and the name of the input file indexed as shown in Figure 3.6 below.

One problem was brought up after the first pass of the pre-processing and the building of the index was that the tags themselves were included in said index, thus they would be searchable by the user if not stripped beforehand during pre-processing. In reality this problem is a more complex one since it also has an impact on the offset of the quote. Because we still want to know where the actual quote begins in the input file, and we do not want to modify the original file, removing the tags themselves is not a possibility. In the Lucene library, when building the index, one main object is used: an Analyzer[2]. It examines a string of text and generates a token stream using tokenizers and filters. Thus we decided to build our own Analyzer to suit our needs by overwriting the tokenizer and the filter functions; allowing us to use the special filter function that strips out tags from the content: HTMLStripCharFilter(). The advantage of that filter is that it gets the position of

[2]http://lucene.apache.org/core/8_0_0/core/org/apache/lucene/analysis/Analyzer.html

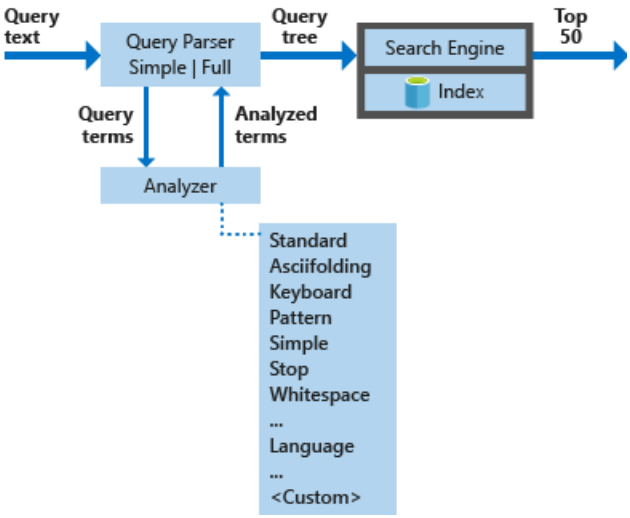

Figure 3.7: Diagram of a query search that Lucene implements.
https://docs.microsoft.com/en-us/azure/search/search-lucene-query-architecture

each word before removing the tags, meaning that the actual position of each word is kept and not influenced by the removal of the tags.

The Analyzer is also used, for example, to remove stop words from the document fields. There is a large number of different analyzers implemented in Lucene; some can, instead of removing stop words, tokenize the text or stem each word depending on the language used. In the Analyzer created, we kept a simple tokenization of the words and used an empty stop words list to initialize it since some of Shakespeare's quotes only contain stop words (i.e., "To be or not to be").

3.1.3 Index querying

Once the index was created and initialized with all the input files we proceeded to the next step: implementing the querying part of the search engine. Figure 3.7 below shows a diagram of a query search that Lucene implements. The process of querying the index is also one of the core functionalities of Lucene. It involves creating a Query and handing it to the IndexSearcher[3] : a class that goes through the whole reverse index to find the request. A

[3]http://lucene.apache.org/core/8_0_0/core/org/apache/lucene/search/IndexSearcher.html?is-external=true

lot of classes are available to create a query depending on what kind of search is needed: one word search, Boolean search, sentence search, etc. Here we used a class called QueryParser[4] to create the queries because it allowed us to put quotes around the query input in order to search for a multi-terms query. The output that Lucene gives from this query search is a ranked list of documents in which the request was found using the scoring formulae seen previously.

Though here, we not only want to know the document the quote appears in but also its exact position in said document to be able to see the context in which it was quoted. We came up with an algorithm using multiple classes from Lucene that would, for each document, compute the position and find the previous and following words in order the build back the context around the query. The output generated by our algorithm is a JSON file containing a structured list of all the books the quote was found in and different additional useful information. Figure 3.8 shows the JSON file with said information: the name of the file retrieved, its title, the number of quotes found in it (frequency) and the quote itself with its position in the file (startOffset and endOffset) and its context.

With the output data in this format, it was logical for us to make a simple web interface to try our search engine on a large scale. The interface can be seen in Figure 3.9 and can be accessed at the following web address: http://text.csce.uark.edu/clip/qt/.

3.2 Goal 2: Efficiently and effectively identifying a paraphrased quote in a text corpus

The following sections will explain in details the implementation of our algorithm to identify paraphrased or modified quotes found in a text corpus dating back to the Eighteenth century. For this phase of the research, we are using the same format of files for our corpora as for the previous experiment.

[4]http://lucene.apache.org/core/8_0_0/queryparser/org/apache/lucene/queryparser/classic/QueryParser.html

```
{
"books":[
    {
      "name":"./files/K062861.000_C.xml",
      "title":"An essay on the character of Hamlet: As performed by Mr. Henderson, at the Th
eatre Royal in the Hay-Market.",
      "frequency":"1",
      "quotes":[
        {
          "startOffset":"27146",
          "endOffset":"27164",
          "context":"\"paſſages gives a ſtrange turn to shakeſpear's meaning he interprets <b>t
o be or not to be </b> am i to exiſt hereafter or not hamlet at firſt\""
        }
      ]
    },
    {
      "name":"./files/K022555.000_C.xml",
      "title":"New hay at the old market: an occassional drama, in one act: written by George
Colman, (the younger, ) on opening the Hay-Market Theatre. On the 9th of June, 1795.",
      "frequency":"1",
      "quotes":[
        {
          "startOffset":"11515",
          "endOffset":"11534",
          "context":"\"hamlet before twelve tallow candles in the coun try daggerwood <b>to
be or not to be </b> fustian yes he's at it let me ſee turning over\""
        }
      ]
    }
  ]
}
```


Figure 3.8: JSON file containing information about the query results

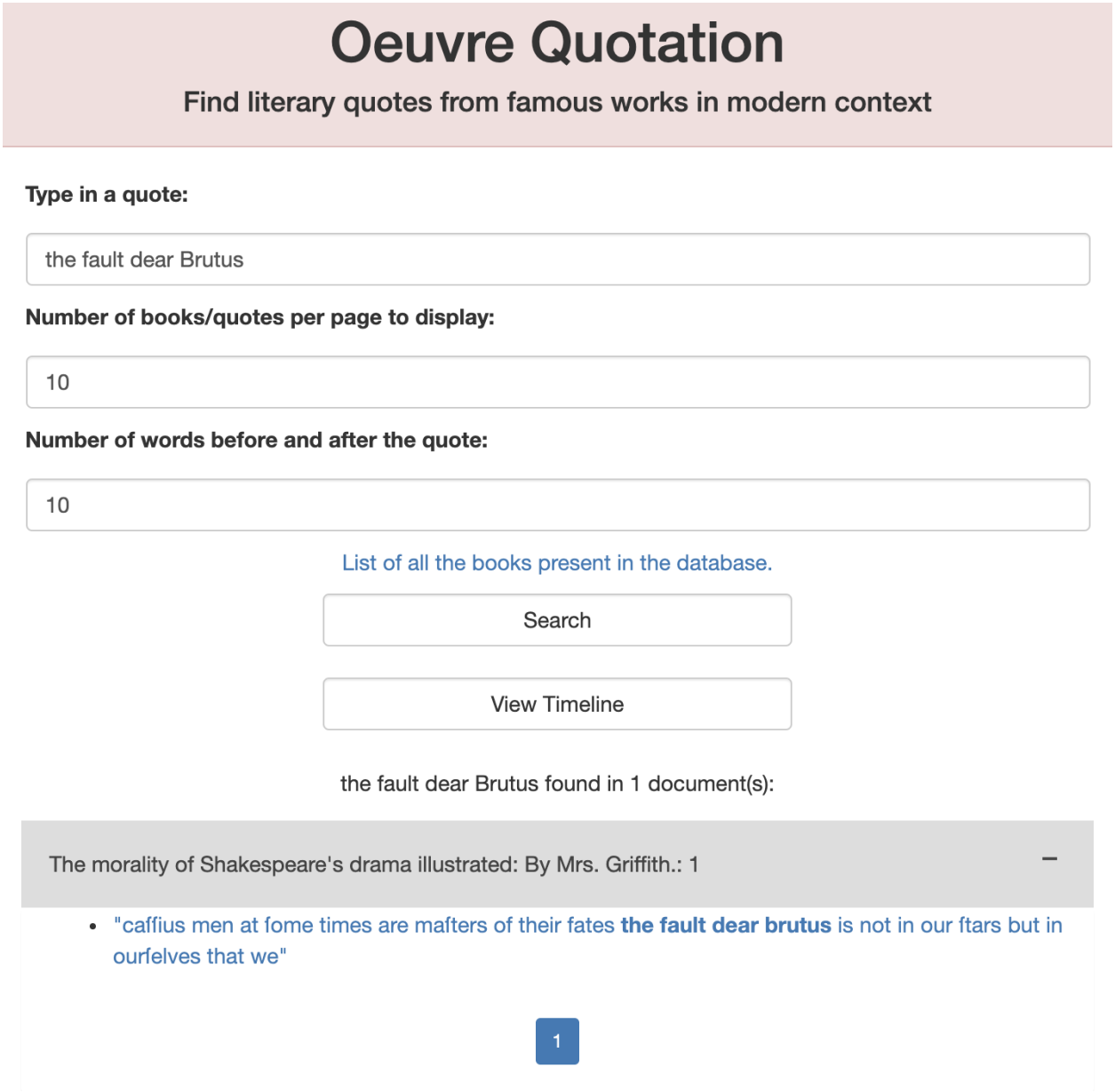

Figure 3.9: Web interface for the exact string matching algorithm implemented with Lucene

3.2.1 Input processing

The first step of any algorithm dealing with text is to pre-process it in order to have uniform text samples that can be used for querying and lead to accurate results. In order to do this, we decided to split up each novel present in our dataset into sets of paragraphs. This task was pretty straightforward since the files used here are in the XML format, as mentioned in Section 3.1.1, and encoded following the TEI sets of rules. Because of the way our novels are encoded, the only time an XML paragraph node (i.e., $< p >$) is present in the file is when it is part of the main body of the novel. Thus after loading each novel in memory, we split it up into paragraphs each time said node was encountered. Each paragraph extracted was then cleaned of any additional XML tags using the following simple regular expression: '$< .*? >$' and Python's RE module [5], and stored into a hashtable (i.e., a dictionary in Python) with a corresponding number (i.e., count of paragraphs encountered so far).

At the same time we were performing paragraph extractions, we were also building a hashtable of words to build our corpora's vocabulary. Thus, we tokenized each paragraph using NLTK [6] and stripped each token from punctuation and transformed them to lowercase. Then for each first occurrence of a pre-processed word present in a paragraph, we created a list and appended the paragraph numbers it appeared in and finally stored said list in a hashtable with the word as key: creating a fast lookup function for a word, returning all of its corresponding paragraphs in a novel.

3.2.2 Querying

The querying part of our dataset is split up in four individual steps, each explained in the following sections.

[5] https://docs.python.org/3/library/re.html
[6] https://www.nltk.org/

3.2.2.1 Sentence extraction

The first part of our querying algorithm is sentence extraction within a novel, meaning finding the words present in our quote within the novel we are exploring. To do that, we started by pre-processing the quote we are looking for in the same exact way as the pre-processing done for the novels explained in the section above (i.e., removing punctuation, etc.). Then for each pre-processed words of our quote, we look it up in our vocabulary hashtable to get the list of all paragraphs it was found in, resulting in a list of all paragraphs that contain at least one word present in the quote. This list was then ordered by the number of times one paragraph appeared in it (i.e., the higher the number, the more quote-words are present in said paragraph) and then arbitrarily filtered at 50%. What we mean here, is that we only take into consideration the paragraphs that have at least half of the quote-words present to move forward with our experiment (i.e., admissible-paragraphs).

Next is the actual extraction of the sentences to-be-compared. For each admissible-paragraphs recorded, and for each quote-words, we build a list of every occurrence of said words within each admissible-paragraph, keeping a record of their start offset (i.e., distance of the word from the beginning of the paragraph). Then, using said offset, we are able to extract a context surrounding it and create a sentence to compare against our quote using the Edit Distance metric (explained in the following section). We calculate the window of extraction using the following equation:

$$[start/end]offset\ sentence = offset \pm length\ quote * context\ parameter \tag{3.1}$$

With

$$context\ parameter = [0, 2, 4, 6, 8] \tag{3.2}$$

And

$$length\ quote = number\ of\ words\ present\ in\ quote \tag{3.3}$$

3.2.2.2 Character and Word Edit Distance

The Edit Distance, also called the Levenshtein distance, is a metric used for measuring the difference between two string sequences by computing the minimum number of editing operations (insertion, deletion, substitution) needed to transform one sequence into the other, the smaller the distance, the more similar the sequences are. The algorithm for its computation can be described like so: creating a 2D array for storing sub-problems and calculating the edit distance for each sub-strings of the two sequences until the array is filled and the answer to our problem is found in D[*length_sentence*1][*length_sentence*2] [13]. The following image shows how the actual edit distance is calculated: As we can see, we calcu-

$$\operatorname{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \operatorname{lev}_{a,b}(i-1,j) + 1 \\ \operatorname{lev}_{a,b}(i,j-1) + 1 \\ \operatorname{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Figure 3.10: Computation of the Edit Distance.
https://en.wikipedia.org/wiki/Levenshtein_distance

late the initial minimum between the indexes of both sub-strings 1 and then proceed with calculating the distance between (i-1) and (j) for an insertion, (i) and (j-1) for a deletion and (i-1) and (j-1) for a substitution and taking the minimum out of those with added 1. This process generates the edit distance for sub-problem (i,j) (e.g.., for sequences lengths = 1). We proceed with this same method for the length of both sequences until we fill out the whole matrix and find the solution for problem(i = length_sentence1, j = length_sentence2).

This computation is mainly used on a character by character basis, which is how we implemented it in the first place. But after analyzing the resulting numbers (see Section 4.3.1.1) we decided to modify our approach, thus modify this algorithm to fit our need to calculate the edit distance on a word basis; meaning the difference of words within a sentence and not characters. The main difference in the newly created algorithm, is that the length of the sequences are the number of words in each sentence and instead of iterating over

a string, we iterate over two lists of words, check as we go if the word in list_sentence1 matches the word in list_sentence2 following the same computation for insertion, deletion and substitution as for Char_ED explained above.

After implementing both methods and following the results obtained (see Section 4.3.1.2 and 4.3.1.3) we decided to combine both methods into one (i.e., calculating both Character and Word edit distance for one sentence). The problem that arose was that both edit distances were on different scales: the character one generated higher numbers than the word edit distance due to the fact that there are clearly more characters than words in a sentence. We present our normalization method to answer this problem in the following section.

3.2.2.3 Normalization

We used the following formula to normalize our data in the range of 0 and 1 in order to have both edit distances within the same scope:

$$normalized\_ED = \frac{(current\_ED - min\_ED)}{max\_ED - min\_ED} \tag{3.4}$$

With

$$[min/max]\_ED = [min/max]\ edit\ distance\ found\ across\ both\ methods \tag{3.5}$$

With this normalization we were able to compare both edit distances after computation and keep the best out of the two (i.e., the smallest one). The output of our algorithm is a file composed of sentences ranked in decreasing order by their corresponding normalized edit distance.

The following section will present the evaluation of our algorithm with details about the corpus chosen and the results obtained.

4 Evaluation

In this chapter, we evaluate the effectiveness of our various algorithms on their ability to identify exact quotes and paraphrased quotes in a corpus of texts. The evaluation of the algorithms is performed on a synthetic dataset presented below and the validation (i.e., testing the algorithms on unseen data) is performed on the whole ECCO-TCP corpus since it contains works in which Shakespeare was quoted.

4.1 Dataset

4.1.1 Quotes

The quote dataset used to evaluate this work is derived from different available lists of the most popular Shakespeare quotes. We selected approximately 75 quotes and then filtered them to select 20 of those that appeared in novels from our chosen corpora (see section below). Each one of them was pre-processed following the different types of error presented in Section 1.1.1.1 and inserted in their corresponding novel where the original quote is located.

- "In my heart of hearts" (Hamlet)
- "Laid on with a trowel" (As you Like it)
- "Something wicked this way comes" (Macbeth)
- "Too much of a good thing" (As you Like it)
- "Break the ice" (The Taming of the Shrew)
- "Own flesh and blood" (Hamlet)
- "The fault dear Brutus" (Julius Caesar)
- "He will make the face of heaven so fine" (Romeo and Juliet)

- “We know what we are” (Hamlet)
- “And all the men and women merely players” (As you Like it)
- “His acts being seven ages” (As you Like it)
- “Cowards die many times before their deaths” (Julius Caesar)
- “The valiant never taste of death but once” (Julius Caesar)
- “Like patience on a monument smiling at grief” (Twelfth Night)
- “Wherefore art thou” (Romeo and Juliet)
- “Angels and ministers of grace defend us” (Hamlet)
- “Was ever woman in this humour woo’d was ever woman in this humour won?” (King Richards III)
- “The rattling tongue of saucy and audacious eloquence” (A Midsummer Night’s Dream)
- “Thou think’st ‘tis much that this contentious storm” (King Lear)
- “This above all: to thine own self be true” (Hamlet)

4.1.2 Corpus

The prose dataset used to evaluate this work is derived from three publicly available corpuses: the ECCO-TCP corpus[1] , the Victorian Women Writers Project[2] and the CELT corpus[3]. All three corpuses contain works starting from the eighteenth century, a time during which authors started to quote or borrow work from Shakespeare. They are provided by the Text Encoding Initiative (TEI) consortium which collectively develops and maintains a standard for the adaptation of texts in digital form. Those guidelines have been widely used by universities, museums, libraries and more for online search, teaching and preservation of texts [14]. Figure 4.1 below shows an example of an XML-TEI encoded novel from one of the corpuses used in this project.

[1]Eighteenth Century Collections Online – Text Creation Partnership by the University of Michigan, the University of Oxford, and Gale University. https://textcreationpartnership.org/tcp-texts/ecco-tcp-eighteenth-century-collections-online/

[2]Text Creation by Indiana University primarily concerned with the exposure of lesser-known British women writers of the 19th century. https://webapp1.dlib.indiana.edu/vwwp/welcome.do

[3]Corpus of Electronic Texts – Ireland's longest running Humanities Computing project. Free digital humanities resource for Irish history, literature and politics. https://celt.ucc.ie//

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<TEI xmlns="http://www.tei-c.org/ns/1.0">
<teiHeader>
<fileDesc>
<titleStmt>
<title>Lionel and Clarissa: A comic opera. As it is performed at the Theatre-Royal in Covent-Garden.</title>
<author>Bickerstaff, Isaac, 1735-1812.</author>
</titleStmt>
<extent>79 600dpi bitonal TIFF page images and SGML/XML encoded text</extent>
<publicationStmt>
<publisher>University of Michigan Library</publisher>
<pubPlace>Ann Arbor, Michigan</pubPlace>
<date when="2008-09">2008 September</date>
<idno type="DLPS">004806995</idno>
<idno type="ESTC">T41007</idno>
<idno type="DOCNO">CW112016809</idno>
<idno type="TCP">K040638.000</idno>
<idno type="GALEDOCNO">CW3312016809</idno>
<idno type="CONTENTSET">ECLL</idno>
<idno type="IMAGESETID">0380901000</idno>
<availability>
<p>This keyboarded and encoded edition of the
        work described above is co-owned by the institutions
        providing financial support to the Early English Books
        Online Text Creation Partnership. This Phase I text is
        available for reuse, according to the terms of <ref target="https://creativecommons.org/publicdomain/zero/1.0/">Creative
        Commons 0 1.0 Universal</ref>. The text can be copied,
        modified, distributed and performed, even for
        commercial purposes, all without asking permission.</p>
</availability>
</publicationStmt>
```

Figure 4.1: XML-TEI encoded novel from ECCO-TCP corpus

The following nineteen novels where extracted from the three corpuses mentioned above: sixteen from the ECCO-TCP corpus, two from the Victorian Women Writers Project, and one from the CELT corpus. The novel illustrated by the symbol * in the following list, was extracted twice since it contains two Shakespeare quote wanted: one duplication of the novel for each quote. The last novel added to the corpus, is Northanger Abbey by Jane

Austen. This novel is what started this research since in the beginning pages, Jane Austen appropriates herself a quote from Shakespeare's play Twelfth Night without ever mentioning its origin.

- The morality of Shakespeare's drama illustrated - by Elizabeth Griffith (ECCO)*
- Prose on several occasions: accompanied with some pieces in verse. [pt.3] - by George Colman (ECCO)
- The lucubrations of Isaac Bickerstaff Esq: revised and corrected by the author. ... [pt.2] - by Richard Steele and Joseph Addison (ECCO)
- The levellers: or, Satan's Privy-Council. A Pasquinade, in three cantos. The author, Hugh Hudibras, Esq - by Hugh Hudibras (ECCO)
- The lives of the poets of Great Britain and Ireland: to the time of Dean Swift. Compiled from ample materials scattered in a variety of books, ... by Mr. Cibber. In four volumes. ... [pt.1] - by Robert Shiells (ECCO)
- Madrigal and Trulletta. A mock-tragedy: Acted (under the direction of Mr. Cibber) at the Theatre-Royal in Covent-Garden. With notes by the author, and Dr. Humbug, ... - by Joseph Reed (ECCO)
- An essay on the character of Hamlet: As performed by Mr. Henderson, at the Theatre Royal in the Hay-Market - by Frederick Pilon (ECCO)
- Pamela: or, virtue rewarded. In a series of familiar letters from a beautiful young damsel to her parents: and afterwards, in her exalted condition, between her, and persons of figure and quality, ... The third and fourth volumes. ... By the editor of the two first. ... [pt.3] - by Samuel Richardson (ECCO)
- Clarissa: Or, the history of a young lady: comprehending the most important concerns of private life. ... Published by the editor of Pamela. ... [pt.3] - by Samuel Richardson (ECCO)

- Seduction: a comedy: As it is performed at the Theatre-Royal in Drury-Lane - by Thomas Holcroft (ECCO)
- Remarks, critical and illustrative, on the text and notes of the last edition of Shakspeare - by Joseph Ritson (ECCO)
- Reasons why David Garrick, Esq; should not appear on the stage, in a letter to John Rich, Esq - by David Garrick (ECCO)
- Lionel and Clarissa: A comic opera. As it is performed at the Theatre-Royal in Covent-Garden - by Isaac Bickerstaff (ECCO)
- English readings; a comic piece, in one act. Inscribed to George Colman, Esq - by James Cobb (ECCO)
- The shadows of Shakespeare: a monody, occasioned by the death of Mr. Garrick. Being a prize poem, written for the vase at Bath-Easton. The second edition. By Courtney Melmoth - by Samuel Jackson Pratt (ECCO)
- The Soul of Lilith, Volume 1 - by Marie Corelli (VWW)
- Marcella, Volume 2 - by Humphry Ward (VWW)
- The Cock and Anchor - by Joseph Sheridan Le Fanu (CELT)
- Northanger Abbey - by Jane Austen

All of those novels were duplicated multiple times and modified to introduce noise in the quotes (i.e., syntax errors, lexicon errors, etc.) creating a total of 120 files for the dataset. We chose 20 novels as a good midpoint to be able to have enough accurate results from precise data to be able to fully evaluate our algorithms. In the table presented below, we can see that the resulting corpus is big enough for us to evaluate our algorithm.

Table 4.1: Size of the dataset.

	Number of Sentences	Number of Words	Number of Characters
Initial 20 Novels	77,743	1,378,435	7,990,145
Resulting 120 Novels	466,458	8,270,610	47,940,870

The XML-TEI encoded novels used in this project were downloaded from the following three websites: University of Oxford Text Archive[4] , Indiana University: VWW Project[5] , and CELT: Corpus of Electronic Texts[6].

4.1.3 Ground truth

The ground truth for the experiment was built by recording where each quote appeared in the dataset. All quotes appeared, verbatim, exactly once. The novels and token numbers (i.e., where the quote occurs within said novel) were stored as explained in Section 3.1. We were able to locate 104 occurrences of these 20 quotes within the corpus (see Section 4.2.3 below).

4.1.4 Metrics

To measure the accuracy of our quote identification algorithms we compute the precision, recall and F-measure of the results. Precision measures the ability of the algorithm to retrieve only correct matches, without false positives; recall measures the ability of the algorithm to retrieve all try positives, without missing any; and the F-measure is a weighted combination of the previous two. We calculate those metrics using the following formulae:

$$Precision = \frac{|\{relevant\ quotes\} \cap \{retrieved\ quotes\}|}{|\{retrieved\ quotes\}|} \tag{4.1}$$

$$Recall = \frac{|\{relevant\ quotes\} \cap \{retrieved\ quotes\}|}{|\{relevant\ quotes\}|} \tag{4.2}$$

[4]http://www.ota.ox.ac.uk/catalogue/index.html
[5]https://webapp1.dlib.indiana.edu/vwwp/welcome.do
[6]https://celt.ucc.ie/

$$Fmeasure = 2 \times \frac{precision \times recall}{precision + recall} \tag{4.3}$$

The previous three metrics calculate the accuracy for a set of results, but do not take the order of those results into consideration. In contrasts, Average Precision (AP) takes the rank order of the results into account since it calculates the precision at cutoff k [15]. Both are calculated as follows:

$$Precision(k) = \frac{relevant\ quotes\ within\ subset\{1; k\}}{k} \tag{4.4}$$

Table 4.2: Example of Precision at Cutoff k = 4 of quote "something wicked this way comes".

Retrieved sentences	Corresponding Edit Distance	Precision(k)
very wicked this wa	13	P(1) = 1/1
eeting of this Court	18	P(2) = 1/2
hing very wicked thi	20	P(3) = 2/3
this way comes." T	21	P(4) = 2/4

$$Average\ Precision = \frac{1}{\{relevant\ quotes\}} \times \sum_{k=1}^{N} (P(k)\ if\ k^{th}\ item\ was\ relevant) \tag{4.5}$$

with

$$N = number\ of\ retrieved\ quotes \tag{4.6}$$

Table 4.3: Example of Average Precision of quote "something wicked this way comes" assuming N = 4.

Retrieved sentences	Corresponding Edit Distance	Precision(k)	Relevant
very wicked this wa	13	P(1) = 1/1	Yes
eeting of this Court	18	P(2) = 1/2	No
hing very wicked thi	20	P(3) = 2/3	Yes
this way comes." T	21	P(4) = 2/4	No

$$AP = \frac{1}{2} \times (\frac{1}{1} + \frac{2}{3}) = 0.833 \tag{4.7}$$

When Average Precision is itself averaged over multiple trials, in our case, 20 quotes, it is called the Mean Average Precision (MAP) and is calculated as follows:

$$Mean\ Average\ Precision = \frac{1}{total\ number\ of\ quotes} \times \sum_{i=1}^{Q} (AP of\ each\ quote) \tag{4.8}$$

with

$$Q = total\ number\ of\ quotes \tag{4.9}$$

4.2 Experiment 1: Evaluating an algorithm to identify exact quotes

The following sections will present the design and setup of this experiment and will discuss the results obtained after evaluation.

4.2.1 Experiment design

This experiment was realized to set up ground truth for our following problem: how to identify modified quotes in a corpus of texts. Thus, we first decided to set up an experiment to identify quotes that have not been paraphrased. To be certain that our algorithm was not able to identify modified quotes, we took each quote and modified it 5 times, following each of the types of error mentioned in Section 1.1.1.1, and inserted them in their corresponding file. We then preprocessed our synthetic dataset of 120 novels (i.e., remove punctuation, lower case, etc.), obtaining text files only containing our target content, and input those files into our Lucene search engine to index them. Finally, we preprocessed the original quotes in the same manner we did the novels and input them into the search engine to try to find them in the previously created index. We were able to identify 104 instances of those quotes as mentioned previously and as shown in the result section below.

4.2.2 Results and discussion

As a reminder for better understanding of the results, here are the corresponding types of error: LongS corresponds to the 'long s' symbol ('ſ') replacing the 's' character,

Split to words being split in two, Insert to new words being added, Remove to original words being removed, and Replace to words being replaced by synonyms or totally different words.

As can be seen in Table 4.4, 104 quotes were correctly retrieved from the dataset, corresponding to the total number of original quotes expected to be found in the novels. Not contrary to our expectations this algorithm yielded null results in regards to quotes containing our target errors, not finding any quote if it deviated in the slightest from the original.

Table 4.4: Results obtained for Experiment 1.

Type of quotes	Number expected in dataset	Number returned	Number correctly retrieved	Precision	Recall	F-measure
Original	104	104	104	100	100	100
LongS	0	0	0	0	0	0
Split	0	0	0	0	0	0
Insert	0	0	0	0	0	0
Remove	0	0	0	0	0	0
Replace	0	0	0	0	0	0

We computed both precision and recall using the numbers found in the table above and the two formulas previously explained and got one hundred percent for both precision and recall; showing that our algorithm is accurate and proficient in retrieving exact matches in textual content.

4.3 Experiment 2: Evaluating an algorithm to identify modified quotes

The following sections will present the design and setup of this experiment and will discuss the results obtained after evaluation.

4.3.1 Experiment design

For this experiment, we used the exact same dataset and the same method as explained for the previous experiment: we modified our initial quotes following our 5 target errors and inserted them into their corresponding duplicated novels. After preprocessing both the quotes and the novels, we input each one of our original quotes into our algorithm

to try to find them in said modified dataset. The results of this second experiment are presented in the sections below.

4.3.1.1 Tuning Character Edit Distance

As explained in Section 3.2.1, the first method tested to identify modified quotes in a text corpus, is the character edit distance. We tested an implementation of it with a first arbitrary context length for sentence extraction like follows:

$$Context\ Length = length\ quote \times 4 \tag{4.10}$$

We then computed each metric discussed above. Those results can be seen in tables 4.5 and 4.6.

Table 4.5: Character ED - Average Precision of all twenty quotes with a context length parameter of 4.

Original	LongS	Split	Insert	Remove	Replace
0.325	0.325	0.042	0.019	0.019	0.019
0.540	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	0.083	0.091	0.091
0.001	0.034	0.063	0.018	0.001	0.001
0.016	0.091	0.333	0.001	0.002	0.006
0.503	0.002	0.003	0.002	0.004	0.001
1.000	1.000	1.000	1.000	0.333	1.000
0.067	0.015	0.067	0.015	0.077	0.056
1.000	0.250	0.250	0.250	0.143	0.045
1.000	1.000	1.000	1.000	0.250	1.000
0.143	1.000	0.100	0.143	0.000	0.042
0.833	0.167	0.500	0.500	0.125	0.250
0.014	0.014	0.010	0.001	0.056	0.500
1.000	1.000	1.000	1.000	1.000	1.000
0.333	0.333	1.000	0.200	0.333	1.000
1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000
0.001	0.002	0.002	0.001	0.005	0.002
0.500	0.059	0.077	0.009	1.000	0.033
0.011	0.016	0.040	0.040	0.003	0.001

Table 4.6: Character ED - Mean Average Precision of all six types with a context length parameter of 4.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.514	0.465	0.474	0.364	0.322	0.402	0.424

As can be seen in Table 4.6, the overall mean average precision for this method is low: 0.424. Thus we decided to try tunning the character edit distance by having the context length parameter vary between 0 and 8 by increments of 2. The results of those variations can be found bellow.

Table 4.7: Character ED - Mean Average Precision of all six types with a context length parameter of 0.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.330	0.296	0.390	0.200	0.276	0.205	0.283

Table 4.8: Character ED - Mean Average Precision of all six types with a context length parameter of 2.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.861	0.730	0.623	0.555	0.517	0.540	0.638

Table 4.9: Character ED - Mean Average Precision of all six types with a context length parameter of 6.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.416	0.404	0.401	0.371	0.286	0.344	0.370

Table 4.10: Character ED - Mean Average Precision of all six types with a context length parameter of 8.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.210	0.202	0.228	0.209	0.157	0.188	0.199

As can be seen in the five tables above, this method performs the best with a context length parameter of 2 and on types of errors related to spelling. It does not do very well on paraphrasing errors, thus why we decided to implement a word edit distance method as a second method. The results for this are presented in the following section.

4.3.1.2 Tuning Word Edit Distance

As explained in Section 3.2.2, the second method tested to identify modified quotes in a text corpus, is the word edit distance. The tests for this methods follow the same parameters as the character edit distance. The results can be seen in the tables bellow.

Table 4.11: Word ED - Average Precision of all twenty quotes with a context length parameter of 4.

Original	LongS	Split	Insert	Remove	Replace
0.054	0.054	0.051	0.053	0.053	0.059
0.545	0.750	1.000	1.000	1.000	1.000
1.000	0.500	1.000	1.000	0.100	0.100
0.001	0.001	0.001	0.000	0.001	0.001
0.001	0.000	0.000	0.001	0.000	.0001
0.833	0.002	0.170	0.250	0.002	0.002
1.000	1.000	1.000	1.000	0.167	1.000
0.167	0.167	0.167	0.015	0.015	0.167
1.000	0.500	0.111	1.000	0.500	0.500
1.000	1.000	1.000	1.000	0.143	1.000
1.000	0.011	0.059	1.000	0.000	1.000
1.000	0.500	0.500	1.000	0.500	0.500
0.500	0.500	0.019	0.001	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000
0.200	0.000	0.500	0.500	0.200	0.500
1.000	1.000	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000
0.200	0.200	1.000	0.003	0.200	1.000
1.000	0.015	1.000	1.000	1.000	1.000
1.000	1.000	1.000	1.000	0.022	1.000

Table 4.12: Word ED - Mean Average Precision of all six types with a context length parameter of 4.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.675	0.460	0.579	0.641	0.395	0.641	0.565

As can be seen in Table 4.12, the overall mean average precision for this method is pretty low: 0.565. Thus, we decided to tune this method following the same steps as for the previous method: varying the context length parameter by increments of 2. The results can be found in the tables below.

Table 4.13: Word ED - Mean Average Precision of all six types with a context length parameter of 0.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.412	0.324	0.416	0.511	0.321	0.391	0.396

Table 4.14: Word ED - Mean Average Precision of all six types with a context length parameter of 2.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.838	0.591	0.508	0.641	0.521	0.557	0.609

Table 4.15: Word ED - Mean Average Precision of all six types with a context length parameter of 6.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.458	0.334	0.321	0.516	0.320	0.345	0.382

Table 4.16: Word ED - Mean Average Precision of all six types with a context length parameter of 8.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.272	0.161	0.152	0.264	0.099	0.263	0.202

As can be seen in the five tables above, this method also performs best with a context length parameter of 2 and this time on types of errors related to paraphrased sentences. In the next section we compare both methods to each other.

4.3.1.3 Comparing Word and Character Edit Distance

To compare both methods with each other, we took the best results from each method: results when the context length parameter is set to 2.

Table 4.17: Character ED - Mean Average Precision of all six types with a context length parameter of 2.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.861	0.730	0.623	0.555	0.517	0.540	0.638

Table 4.18: Word ED - Mean Average Precision of all six types with a context length parameter of 2.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.838	0.591	0.508	0.641	0.521	0.557	0.609

As we can observe in the two tables above, both methods are very effective at identifying the quote in its original context, with MAP values of 0.861 and 0.838 respectively. They also both perform equally poorly on quotes that had words removed with MAP values of 0.540 and 0.557 respectively. That is normal since the edit distance for a sentence with less words will be larger than other extracted sentence: it will take a bigger number of operations to match the two sentences, thus resulting in a bigger edit distance. Where they greatly differ, is on the specific types. We can clearly see that Character ED performs better at detecting quotes with spelling errors with a MAP of 0.730 and 0.623 for the two spelling error types; where Word ED underperforms with 0.591 and 0.508 for those errors. However, Word ED performs better than Character ED on the paraphrased error types, especially when a word is added to the quote, with a MAP value of 0.641.

Discussion: Combining Word and Character ED

Following the results of both methods, since one was better on spelling errors and the other on paraphrased errors, we decided to create a third method that combines both, to see if it can perform well across all types of errors. As explained in Section 3.2.3, for each sentence extracted from a novel, this method computes both the character edit distance and the word edit distance between said sentence and the quote searched for; and, after normalization only keeps the smallest value. This method is evaluated in the next section following the same metric used for the other methods, mean average precision.

4.3.2 Results and discussion

Table 4.19: Combined - Average Precision of all six types with a context length parameter of 2.

Original	LongS	Split	Insert	Remove	Replace
1.000	0.500	0.500	0.500	0.500	0.500
1.000	1.000	0.833	0.583	1.000	0.583
1.000	1.000	0.500	1.000	1.000	1.000
0.500	0.500	0.500	1.000	0.000	0.000
1.000	0.333	0.333	1.000	0.083	1.000
1.000	0.125	0.125	1.000	1.000	0.125
1.000	0.500	0.500	0.007	0.500	0.333
1.000	0.500	0.500	1.000	1.000	1.000
1.000	1.000	1.000	0.333	1.000	1.000
1.000	0.500	1.000	0.500	0.167	0.500
1.000	1.000	0.000	1.000	0.000	1.000
1.000	0.500	1.000	1.000	0.500	0.500
1.000	1.000	1.000	1.000	1.000	0.500
1.000	1.000	0.500	1.000	1.000	0.500
0.500	0.500	0.500	0.500	0.500	0.500
1.000	1.000	1.000	1.000	0.000	0.500
1.000	1.000	1.000	1.000	1.000	1.000
1.000	0.500	0.500	0.500	1.000	0.500
1.000	1.000	1.000	0.500	1.000	1.000
1.000	1.000	1.000	1.000	1.000	1.000

Table 4.20: Combined - Mean Average Precision of all six types with a context length parameter of 2.

Original	LongS	Split	Insert	Remove	Replace	Overall
0.950	0.723	0.665	0.771	0.663	0.652	0.737

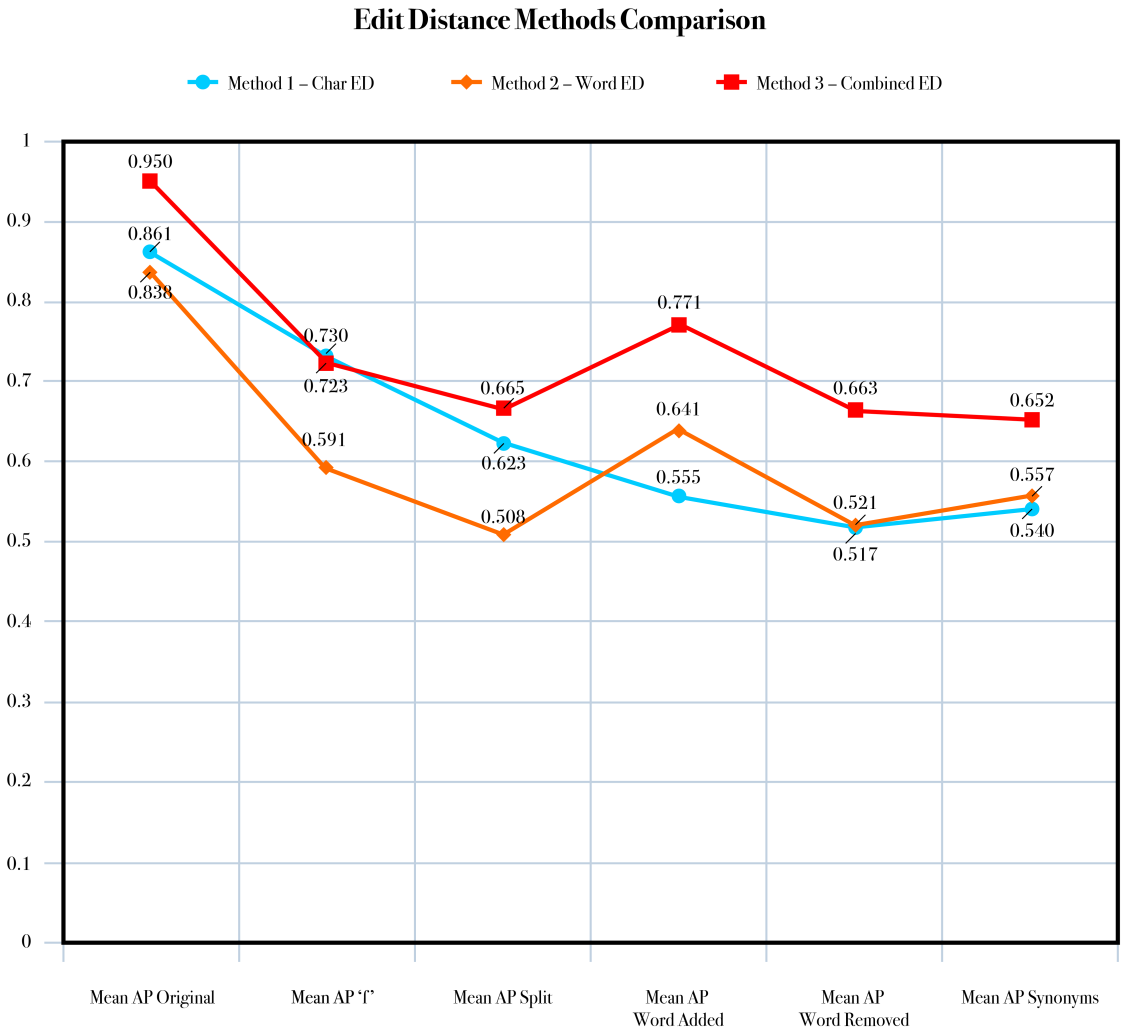

Figure 4.2: Chart comparing all three methods for computing the Edit Distance.

As we can see in the tables and the graph above, combining the character and word edit distances is better than each individual method across all types of errors. It improved the overall MAP value by 10%, bumping it to 0.737. It also improved both methods regarding the

error pertaining to words removed from the quote: jumping from 0.521 to 0.663, an increase of almost 15%. The fact that this method is also at 95% accuracy on identifying original quotes, shows how effective it is. It is very close to the algorithm built with Lucene with identifying exact quotes (see Section 3.1) with an accuracy of 100%. Overall, this shows that given a dataset of curated novels and quotes, our algorithm with combined edit distances can successfully identify modified quotes, whether it is a spelling error or a paraphrased error. In the next section, we will be running our twenty quotes on the entire dataset to generate validation results and observe whether or not our algorithm can identify those quotes in unseen novels.

4.4 Validation

After the positive results obtained on our curated dataset (see Section 4), we decided to validate our algorithm on data that was not previously included. Here, we decided to use the entirety of the freely available section of the ECCO-TCP corpus (i.e., about 2,400 novels). This decision was made on the basis that those texts are specifically within the time period Shakespeare would have been the most quoted without credit attribution (i.e., 1700 - 1800), thus where there would be a higher chance for us to encounter some of the quotes previously chosen (see Section 4.1.1). The results obtained are presented in Table 4.21 below and explained in the following paragraph. In Table 4.21, zeros were omitted for the legibility of the data obtained.

For this experiment, we chose to only select quotes with an Edit Distance (ED) value strictly inferior to 0.5 in order to include quotes in the Removed category. Surprisingly enough, this category of error, along with Insert, came back empty; showing that authors were actually not purposefully modifying quotes but instead probably just copying and pasting the original within their own work without proper attribution. One quote stood out for the category Replace: "wherefore art thou", which was the only quote that appeared to have been modified within a new context by "why art thou". Though here we cannot say that this was used with the intent of re-using a Shakespeare saying, since those were very

common words to use in that period of time. One would have to look more in depth at the novel to understand whether or not this was an intentional modification of a pre-existing quote.

Finally, as we can see in the table below, we can say that the most common type of error present in texts from the Eighteenth century is the 's' character being replaced with the long s symbol. This type of error not being foreseen is something that could hinder one's search for quotes within a corpora spanning over that century, thus it is important to take it into consideration when doing so.

Table 4.21: Validation results over ECCO-TCP corpus with an Edit Distance threshold of 0.5

	No error	LongS	Split	Insert	Remove	Replace
in my heart of hearts	6					
laid on with a trowel	2					
something wicked this way comes						
too much of a good thing	4					
break the ice						
own flesh and blood		9				
the fault dear Brutus	1					
he will make the face of heaven	3					
we know what we are	1					
and all the men and women merely players	23					
his acts being seven ages	3	1				
cowards die many times before their deaths	3					
the valiant never taste of death but once	2	1				
like patience on a monument smiling at grief	9	3				
wherefore art thou			1			6
angels and ministers of grace defend us		7				
was ever woman in this humour woo d? was ever woman in this humour won?						
the rattling tongue of saucy and audacious eloquance	2					
thou think st tis much that this contentious storm						
this above all to thine own self be true	2					

5 Conclusion and Future Work

5.1 Conclusion

This paper describes an effective technique for locating exact and modified quotes in texts from the Eighteenth century by implementing an algorithm using the Edit Distance metric. In this work, we have successfully implemented a method widely used in adjacent fields, like Biology, to works of literature to provide an additional baseline of study regarding quote identification.

We collected popular Shakespeare quotes and 20 novels starting from the eighteenth century and pre-processed them to introduce the errors observed in our pre-analysis of the eighteenth century corpus (long s instead of a common s, split words, etc.). We first indexed said constructed corpus with our Lucene-based algorithm to generate our ground truth, successfully locating 104 occurrences of these 20 quotes (i.e., the exact versions of the quote looked for). We then investigated different context length for extraction of a quote within a novel, ranging from 0 (i.e., only the length of the quote) to 8 (i.e., 8 times the length of the quote). We calculated the Character Edit Distance (CharED) and the Word Edit Distance (WordED) to the original quote for each parameter value and found that the best results were generated with a context length parameter of 2; with overall results of 0.638 and 0.609. During this experiment we were also able to show the effectiveness of each metric in identifying quotes from specific error categories with CharED mostly identifying spelling errors and WordED paraphrasing errors.

Proceeding with a context parameter of 2 for our last experiment of combining both methods, we found that it yielded the best overall results, with an accuracy of 0.950 for original quotes, 0.695 across all 5 types of errors and an overall accuracy of 0.737, showing that the edit distance is a metric successful in identifying paraphrased quotes.

5.2 Future Work

We believe that this work can be extended to answer more questions on both Computer Science literary analysis. We can extend it by first looking into improving the algorithm presented above: modifying the different parameters set within our implementation (i.e., number of sentences to go through after filtering), trying different context extraction methods, or adding another existing metric to improve the results of the combined Edit Distance. This work could also potentially employ a Machine Learning approach with an accurately annotated dataset of quotes and novels as it would be of interest in comparing those results to the ones presented in the above sections.

Another way this work can be extended is by scaling up our dataset to answer questions like the following: what would it take to search for all possible Shakespeare quotes in 1,000 10,000 or even 1,000,000 texts? How would our algorithm fare within those settings? If it isn't efficient, how can we work to improve the algorithm? Would we have to change method and abandon the edit distance? Those are important questions that should be looked into in follow-up work.

This research can also be adapted to the need of tracking quotes through time by identifying the publication dates of the resulting novels and building a timeline for one specific quote. This would be very helpful for scholars working on the Literary side of this research, seeing how it would be very difficult and time consuming to have to read through multiple novels to first identify the possible quote and then construct the timeline for it by hand. Thus, we believe it would be helpful to develop this side of the work within future work. It could also be extended to other time periods (i.e., observing textual reuse in novels from the Fifteenth century or Twenty and Twenty-First century) or even within other languages around the world: is it as prominent as in English novels from the eighteenth century? The results could be analyzed in a manner of culture: are the similarity/differences due to cultural differences? Or to cultural evolution (in the case of observation through time)?

Bibliography

[1] V. Joynes, "Into the 18th century: Shakespeare in performance," May 2016. [Online]. Available: https://www.shakespeare.org.uk/explore-shakespeare/blogs/18th-century-shakespeare-performance/

[2] F. Ritchie and P. Sabor, *Shakespeare in the Eighteenth Century*. Cambridge University Press, 2012.

[3] "Search engine (computing)." [Online]. Available: https://en.wikipedia.org/wiki/Search_engine_(computing)

[4] "Whois." [Online]. Available: https://en.wikipedia.org/wiki/WHOIS

[5] D. Duhaime, "Textual reuse in the eighteenth century: Mining eliza haywood's quotations," *DHQ: Digital Humanities Quarterly*, vol. 10, no. 1, 2016.

[6] "Dionysian imitatio." [Online]. Available: https://en.wikipedia.org/wiki/Dionysian_imitatio

[7] M. Büchler, "Unsupervised detection and visualisation of textual reuse on ancient greek texts," *Journal of the Chicago Colloquium on Digital Humanities and Computer Science*, vol. 1, no. 2, 2010.

[8] N. Bernstein, "Comparative rates of text reuse in classical latin hexameter poetry," *DHQ: Digital Humanities Quarterly*, vol. 9, no. 3, 2015.

[9] D. Sankoff, "Edit distance for genome comparison based on non-local operations," in *Combinatorial Pattern Matching*, A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 121–135.

[10] X. S. Wang, Y. Huang, Y. Zhao, H. Tang, X. Wang, and D. Bu, "Efficient genome-wide, privacy-preserving similar patient query based on private edit distance," *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS 15*, 2015.

[11] J. Bao, R. Yuan, and Z. Bao, "An improved alignment-free model for dna sequence similarity metric," *BMC Bioinformatics*, vol. 15, no. 1, 2014.

[12] "Introduction to xml." [Online]. Available: https://www.w3schools.com/xml/xml_whatis.asp

[13] "The levenshtein distance algorithm." [Online]. Available: https://www.educative.io/edpresso/the-levenshtein-distance-algorithm

[14] "Tei: Text encoding initiative." [Online]. Available: https://tei-c.org/

[15] S. Sawtelle, "Mean average precision (map) for recommender systems," October 2016. [Online]. Available: http://sdsawtelle.github.io/blog/output/mean-average-precision-MAP-for-recommender-systems.html