Cite Unseen: A Plagiarism Detection System Using the Public Web as a Text Corpus

Brian St. Marie

A Thesis in the Field of Information Technology

for the Degree of Master of Liberal Arts in Extension Studies

Harvard University

May 2014

Abstract

Cite Unseen is a Plagiarism Detection System (PDS) that utilizes publicly searchable corpora for detection of plagiarism. Use of publicly searchable corpora avoids the construction and maintenance requirements of a private corpus of documents. Cite Unseen detects plagiarism by breaking a submitted document down into short phrases (n-grams) and passing these phrases through a designated search engine. The search results are then analyzed for common websites and other patterns that may indicate copying by the author. Cite Unseen supports Google Custom Search, Yahoo! BOSS Search, Microsoft Bing Search, and FAROO Search for public corpora. Cite Unseen can easily support additional web search engines by extending the existing code-base.

Cite Unseen identifies areas of plagiarized text through a process known as local sequence alignment. Sequence alignment is a common problem in computational biology that has many similarities to challenges in natural language processing. Cite Unseen uses a linear $O(n)$ time dynamic programming algorithm for local sequence alignment. Each resulting sequence represents a series of clustered, but potentially non-contiguous words that match some source document returned by the search provider. Each sequence is scored based on the frequency of matched terms in that sequence. Word sequences with the most significant scores are highlighted in the suspect text and displayed to the user.

Dedication

To my wonderful fiancée Brittany, for all her patience through the long days and even longer nights. To my loving family, for their encouragement and guidance. And to all my friends who let me tear their various papers to pieces in the name of research.

Acknowledgments

I'd like to thank the following people who were integral to completion of this project.

My thesis director, Dr. Paul Bamberg, for his original inspiration, continued insights, and the faith to put the project in front of a room of Harvard deans, sight unseen.

My research advisor, Dr. Jeff Parker, for the polite but persistent prodding that kept me on track, even when things seemed to completely derail.

Andrea Held and Jenn Phillips of Google for sticking with me through all the ups and downs.

Karen Chin and the Yahoo! BOSS Team for their support and encouragement throughout the project.

Charles Sawyer, who knew just how to put the screws to get things done when I needed it most.

And finally, a special thanks to Varun Ganapathi and Terminal.com for giving my project a home all these months.

Table of Contents

List of Equations

List of Examples

List of Figures

List of Tables

Chapter 1 Introduction

Academic plagiarism is a long-standing and major concern throughout higher education. Many high schools, universities, and colleges utilize commercial Plagiarism Detection Systems (PDSs) to identify plagiarism in all submitted student work.

Typical PDSs seeks out plagiarism through intrinsic and/or external means. Intrinsic detection relies entirely on the content of the source document to determine deviations from the usual style of the author. By fingerprinting the style of the author, unusual or outlying fragments can be identified for further review (Meyer zu Eissen & Stein, 2006). Intrinsic detection can locate plagiarism without any comparison to other documents, but is typically low-performing in comparison to external detection (Grozea & Popescu, 2010) and requires extremely large amounts of sample text (Stein, Lipka, & Prettenhofer, 2011). In contrast, external detection relies on comparison of the source document with a corpus of existing documents. The source document is broken down into fragments, typically word *n*-grams, that are then compared to material in the corpus through a variety of pattern matching methods (Potthast, Stein, Eiselt, Barrón-Cedeño, & Rosso, 2009). Any matches are flagged as possible plagiarism.

In practice, most commercial PDSs use the external method of detection. The basic process at many schools requires the student or faculty member to upload the work in question to a designated PDS partner, which then checks the suspect document against an internal database of potential source documents. This internal database is typically assembled both from public data on the internet and a submission database that grows

with each additional document upload. Each document checked against the database then becomes a part of that database. This process increases the coverage of the corpus, improving results over the age of the service.

Some institutions object to this type of PDS, as it inherently ignores student copyright by indefinitely storing uploaded documents for use in future scans. Institutions or students wishing to respect student copyright will not use a system that relies on archives of past submissions. This creates major challenges for plagiarism detection when student copyright is respected. In some cases, institutions opt to use no detection at all in lieu of creating copyright issues.

Cite Unseen solves this problem by using only publicly available documents for plagiarism detection, rather than relying on a corpus built from submitted works. This approach has several advantages, such as respecting copyright and offloading all database management and processing to the pubic corpus holder. However, such a system also faces challenges, such as being limited to only publicly indexed documents and dependency on the public corpus holder for continued operation.

The remainder of this chapter provides a high-level overview of Cite Unseen. The first section lays out the objectives of this project and how they were achieved. The final section discusses some assumptions made in the project based on current research in the fields of plagiarism detection and natural language processing.

The following chapters go into greater detail on the application methodology and design, the user interface, and test results. The second chapter discusses the logical and mathematical methodology used by Cite Unseen to analyze a document and locate signs of possible plagiarism. The third chapter outlines the user interface and the user

experience. The fourth chapter details the programmatic design and implementation of the application. The fifth chapter presents results and analysis of the Cite Unseen approach to plagiarism detection. The final chapter provides a summary of the research, major challenges encountered, and possible future expansions on the project.

Objectives

The main objective of this project was to determine if a viable PDS could be developed utilizing only publicly available information as a corpus, indexed and made searchable by a third party. To determine this, the project focused specifically on external detection using various web search providers as public corpora, such as Google Custom Search, Yahoo! BOSS Search, Microsoft Bing Search, and FAROO Search.

As most commercial PDSs build their proprietary corpus through both user submissions and public internet data, the project faced an inherent disadvantage in available data. Therefore, the goal of the project was not necessarily to perform better than existing commercial systems, but to demonstrate a "public only" approach as feasible and reasonably competitive, both in speed and accuracy, with commercial PDSs.

A final objective of the project was to develop a novel approach to detection. Many PDS algorithms have been designed and tested; however, there is a basic assumption that full access to both the suspect and potential source documents is available. In the case of Cite Unseen, full access to each potential source document is computationally impractical. Instead, Cite Unseen must compensate for incomplete representation of a source document through a novel algorithmic approach.

Assumptions

While most commercial systems do not share their detection methodology, there has nonetheless been ample public research and testing in the area of plagiarism detection. This research has defined several assumptions and concepts that formed the basis for the Cite Unseen project. These assumptions are detailed below.

n-Grams are Optimal for Detection

Research in external plagiarism detection algorithms has explored several avenues of detection. Many of these methods rely on full access to the content of both the suspect document and the target document. For instance, bag-of-words models rely on an unordered set of words for each document, along with the frequency of each word. Likewise, hashing or fingerprinting requires calculating unique values for all or part of both the suspect and target documents for comparison. This requirement to have the full text of each target document is prohibitive when dealing with an internet corpus, however; having to download each document for processing is simply too performance intensive. Other methods avoid this problem but have their own limitations. Full sentence matching misses obvious paraphrasing or word order swaps. Citation matching (using only cited documents as a target corpus) works well (Gipp & Beel, 2010), but applies only to documents with citations.

The most common solutions for plagiarism detection involve *n*-grams (Grozea, Ghel, & Popescu, 2009). To create an *n*-gram, the suspect document is broken down into sequences of *n* words or characters and these *n*-grams are then searched against the provided corpus for any possible matches. Research has shown that word trigrams are the optimal choice for external plagiarism detection (Barrón-Cedeño & Rosso, 2009; Lyon,

Barrett, & Malcolm, 2006). If characters are compared rather than words, research indicates character *18*-grams may be optimal (Grozea et al., 2009); these character *18*-grams typify the average 3 word character length while avoiding inefficiencies caused by very short, common words (e.g. "as far as"). However, as search engines index text by words, Cite Unseen uses the word *n*-grams approach.

Most *n*-Grams are Rare

Zipf's law, named after American linguist George Zipf, is an empirical law demonstrating that word frequency in natural language follows a power law probability distribution. Essentially, the *n*th most frequent word is proportional to a frequency of $1/n$. For example, in the Brown Corpus Manual (Francis & Kucera, 1979), it was shown that only 135 unique words account for over half of the approximately one million words in the corpus. Thus, commonly used language consists of a relatively small list of words, while most other words are comparatively rare. Zipf's law also extends to *n*-grams (Ha, Sicilia-Garcia, Ming, & Smith, 2002; Lyon, 2004). Therefore, most *n*-grams are rare and if two unrelated documents are compared, they should share relatively few *n*-grams between them.

In discussion of the *Ferret Copy Detector* (Lyon, 2004), a formula is presented for calculating the $Resemblance$, or percentage of shared *n*-grams, between two documents. Let R be the $Resemblance$ between the two documents A and B. If N_A and N_B are the collections of *n*-grams in each document, then the ratio of shared *n*-grams to all *n*-grams in both documents is the $Resemblance\ R$:

$$R = \frac{N_A \cap N_B}{N_A \cup N_B}$$

Equation 1: Resemblance

Assuming Zipf's law is correct for *n*-grams, there should be some threshold of R such that values above this threshold strongly indicate possible plagiarism. This hypothesis was tested using trigrams (Lyon et al., 2006) across several different corpora. The percentage of unique trigrams was in the range of 77% to 87%, even in the case of corpora consisting primarily of the same subject matter or the same authors.

These findings imply that a typical R value can be empirically determined for unique document pairs. Documents that share an R value higher than this threshold would be unusually similar, possibly indicating plagiarism. This method has been extensively explored in the plagiarism detection tools *Ferret* and *WebFerret* (Malcolm & Lane, 2008) and is considered effective. However, this method requires full access to both the suspect and source documents. Instead, Cite Unseen relies on a different approach as detailed in Chapter 2 .

Text Pre-processing has Limited Usefulness

One goal of many research projects is to pre-process the text of the suspect document in order to minimize the amount of searching and maximize matching with unique or unusual phrases. In testing of several text pre-processing methods, most are found to have only minimal benefit, often at the cost of performance (Ceska & Fox, 2009). The primary exceptions to this are removing punctuation, ignoring sentence boundaries, and tokenizing documents into *n*-grams. Cite Unseen uses all three of these pre-processing methods, but uses no other text pre-processing.

Chapter 2 Methodology

This chapter details the methodology and logical process Cite Unseen follows to identify areas of suspected plagiarism in a submitted document. Cite Unseen processes submissions through four distinct phases. First, the text is pre-processed, breaking it down into *n*-grams optimized for searching. Second, the chosen *n*-grams are processed through the search provider for matches. Third, the match results are processed to identify possible plagiarism; this phase involves elimination of URL duplication, creation of match sequences between the URLs and the submission, and the scoring of these sequences. Finally, the results are presented to the submitter for review. Each of these four phases is detailed below. To illustrate the function of each step, a simple example of plagiarism from Wikipedia is used.

The original text, taken from the Wikipedia article on plagiarism:

Plagiarism is not a crime per se but in academia and industry it is a serious ethical offense, and cases of plagiarism can constitute copyright infringement.

Example 1: Source Wikipedia text

The suspect text, paraphrased from the above sentence:

Plagiarism is not exactly a crime, but it is a serious ethical offense in academia and industry. Cases of plagiarism sometimes constitute copyright infringement.

Example 2: Suspect plagiarized text

Each step below demonstrates how the suspect text is processed and matched with the original text.

Pre-Processing Submitted Text

The first step in analysis removes any ambiguous or unnecessary information from the submitted documents. Removing excess information prevents needless searching and maximizes the possibility for detection.

The text of the submission is first stripped of any irrelevant formatting and punctuation. Cite Unseen removes formatting based on Unicode Text Segmentation guidelines ("UAX #29: Unicode Text Segmentation," 2013). These guidelines adhere as closely as possible to what a user would expect when segmenting and removing formatting from a string of words. For instance, the periods in a URL address or the apostrophe in a contraction are preserved, while the quotations around a word or the periods at the end of a sentence are removed.

After formatting and all irrelevant characters are removed, the text is tokenized into *n*-grams. These n word groupings overlap and ignore sentence boundaries. Duplicate *n*-grams are eliminated through storage in a hash set. At the completion of this phase, the suspect text is simply a set of size t where t is the number of unformatted, unique *n*-grams in the set. Using the text from Example 2 above and assuming an n value of 3 provides the following set of *n*-grams.

Trigram	Trigram
plagiarism is not	ethical offense in
is not exactly	offense in academia
not exactly a	in academia and
exactly a crime	academia and industry
a crime but	and industry cases
crime but it	industry cases of
but it is	cases of plagiarism
it is a	of plagiarism sometimes
is a serious	plagiarism sometimes constitute
a serious ethical	sometimes constitute copyright
serious ethical offense	constitute copyright infringement

Table 1: Suspect text broken into unformatted trigrams

This *n*-gram set of size t is then submitted to search through the provider API.

Interfacing with the Search Provider

Each search provider supported by Cite Unseen makes available an application programming interface (API) for access to their search service. Cite Unseen interfaces with these APIs in order to obtain search results.

Maximizing Search Speed

Active searching is the most substantial performance obstacle for the Cite Unseen approach to plagiarism detection. Because Cite Unseen interfaces with public search providers rather than a private database, performance is affected by several factors outside the control of the application.

The primary impediment to search performance is the latency between query submission and receipt of results. As an example, the published typical query time of Google Search is 200ms (Hölzle, 2009). Assuming an average number of words per page of 250, which is also equal to the approximate number of word *n*-grams per page for any

reasonable value of n, this equates to an average of fifty seconds per page. In practice, testing shows providers respond on average of around 500ms, resulting in search times of roughly two minutes per page.

To overcome this challenge, Cite Unseen utilizes concurrent searches through multi-threading. Multi-threading increases query times to 10,000 searches or more per minute when run over a standard internet connection. This equates to roughly forty pages per minute. However, search providers impose varying limitations on the number of searches possible within a short time. Cite Unseen compensates by rate limiting this step to each search provider's requirements, as necessary.

Overall, this method proves adequate in improving the speed of Cite Unseen to a reasonable level. However, further improvements are available with certain search providers via specific selection of data returned or gzip type compression of returned results.

Parsing Search Results

Each query returns a Search Engine Results Page (SERP). This SERP contains the total number of results for that query, a fixed size list of URL results for the query, and various other information, such as description text for each URL with the search term in context. Each search provider returns a fixed number of results in each SERP. For instance, Google returns ten URL results per query while Yahoo! returns fifty. Thus, the size of the final collection of search results m' is proportional to the size t of the n-gram set searched.

The actual format of the returned SERP is typically selected through the original request sent to the API; common formats are JSON and XML. The individual SERPs for

each query are then parsed by Cite Unseen into a general, provider-independent format for further processing.

The Challenge of Common Terms

When searching for more common *n*-grams, a plagiarized source may not fall within the first ten, or even fifty search results for that *n*-gram. This creates inaccuracies in processing, as more common *n*-grams may be missing key URLs in their search results. For instance, searching a portion of the trigrams from Table 1 and locating the original Wikipedia source website in the results, shows the following:

Trigram	**Trigram in Source**	**URL Search Rank**
plagiarism is not	True	2
is not exactly	False	-
not exactly a	False	-
exactly a crime	False	-
a crime but	False	-
crime but it	False	-
but it is	False	-
it is a	True	100+
is a serious	True	100+
a serious ethical	True	17
serious ethical offense	True	2

Table 2: URL match and result rank for each trigram

Although there are five exact trigram matches between the trigrams in Table 2 and the trigrams of the original Wikipedia article, only two of them are found in the top ten results, and only three in the top fifty. The remaining two matches are not located in the top 100 returned URLs and would be missed by any processing.

Naturally, the more common a trigram, the more likely it is for a correct match to fall outside any search results obtained. These missed matches are important because

common *n*-grams are still beneficial for matching purposes once a possible source document is located (Ceska & Fox, 2009). Therefore, this challenge must be addressed.

Utilizing URL Summary Text to Improve Search Results

As previously mentioned, search providers typically provide a summary description with each returned result. This summary text, frequently referred to as a "snippet", includes the search term bolded for easy visibility, known as the "keyword in context" (KWIC). This summary text is used by Cite Unseen to overcome the limitations of locating common *n*-grams in search results.

As each individual search result is processed, Cite Unseen also analyzes the summary text for additional matches. The summary text is broken into *n*-grams and pre-processed identically to the original submitted document, then matched against that document. Any matching *n*-grams between the two indicates an additional match that may not have been found during the original searching process. This is especially true for common *n*-grams, which may not have been returned in the fixed number of search results provided by the search provider.

Using the trigram "plagiarism is not" from the trigram list in Table 1 demonstrates the benefit of this method. For instance, the summary text returned for the Wikipedia page on plagiarism after a Google query on "plagiarism is not" returns:

> ***Plagiarism is not*** *a crime per se but in academia and industry it is a serious ethical ..."**Plagiarism" is not** mentioned in any current statute, either criminal or civil.*
>
> **Example 3: Google search summary text**

Tokenizing this summary text into *n*-grams and comparing with the trigrams in Table 1 shows matches with "it is a", "is a serious", and "a serious ethical". These three additional matches allow the Wikipedia page to be added to the list of matched URLs for

each of these queries. This greatly improves on the results in Table 2, matching the Wikipedia source page with all five matching trigrams. In fact, all five are located within the top two search results for only two of the five matching trigrams.

In practice, this method has a significant effect on detection, especially in the case of particular common query terms like “it is a”, which may not be otherwise detected. In fact, using this method Cite Unseen is able to perform well with only the first ten search results for each query, substantially limiting the amount of searching necessary. The full ramifications of this improvement are explored in more detail in Chapter 5 .

It is important to note the additional searching of description text does impact performance. The description text lengths vary between providers; the Google description is 156 characters, Yahoo! is 161 characters, and Bing is 150 characters (Habermann, 2009). Assuming an average characters per word of approximately five for English (Hearle, 2003) plus one space, this equates to approximately twenty six words for each search description. Cite Unseen stores document *n*-grams in a hash set, which provides $O(1)$ lookup time for each match operation. Thus, looking up additional *n*-grams from the description text of each search result requires $1 \leq (27 - n) \leq 26$ additional $O(1)$ match operations for each *n*-gram. However, since *n* is chosen as a constant value at document submission, this impact is only constant in time for each *n*-gram in the document. Thus, the searching and parsing phase remains linear in time $O(t)$, and the total size of search results m' remains proportional to t, where t is the number of *n*-grams in the submitted document.

Identifying Significant Matches

After all search results have been parsed and all URL summary texts compared for additional matches, the final set of search results is then analyzed for patterns that may indicate plagiarism.

Visual comparison of the texts from Example 1: Source Wikipedia text and Example 2: Suspect plagiarized text identifies the exact match between the two texts.

> ***Plagiarism is not*** *exactly* ***a crime, but it is a serious ethical offense in academia and industry. Cases of plagiarism*** *sometimes* ***constitute copyright infringement****.*
>
> **Example 4: Suspect text with matching words in bold**

An ideal method for finding matches would directly compare the content of the suspect document with the full text of each potential source document. In fact, this direction comparison is the typical method for offline PDSs with private corpora of limited size. However, when utilizing public web pages, it's impractical to algorithmically look through each individual URL document for matching words. This would require a significant amount of time to download each document, as well as further text analysis after those documents were downloaded.

To circumvent this problem, Cite Unseen creates a fragmented version of the given URL document from the search results. This fragmented version is then used to perform matching with the suspect document. Early versions of Cite Unseen calculated a rough approximation of $Similarity\ S$ between a submitted document and a URL document by dividing the number of times a URL appears in search results M by the total number of n-grams N in the document D.

$$S_{(URL,D)} = \frac{M_{URL}}{N_D}$$

Equation 2: Similarity

The application then highlighted URLs with the highest percent matches, or similarity, with the suspect document. This works surprisingly well for short suspect documents, but quickly breaks down for longer documents. A suspect document of 100 *n*-grams, ten of which match with some potential source URL, results in a similarity of 10%. However, a 1000 *n*-gram suspect text with the same match of ten results in a similarity of only 1%.

Using a fixed threshold for *n*-gram matches between the suspect document and the potential source URL avoids this problem. For instance, if the threshold is set to a value of ten, then both the example URLs above would be identified. However, this method does not the address situations where a possible source URL has several very short matches scattered throughout a suspect text. This scenario is a major issue in longer documents, which inevitably result in more matches, especially on common *n*-grams. Large numbers of common *n*-grams inflates the similarity score of possible source URLs to the point where the scoring becomes meaningless to the user.

It is clear that more sophisticated methods for document comparison are necessary. To tackle this problem, Cite Unseen identifies significant similarity through an algorithmic process known as sequence alignment.

Sequence Alignment

Sequence alignment is a frequently used algorithmic matching process in computational biology. Sequence alignment is typically a pairwise matching process that

compares two sequences for optimal alignment via dynamic programming. The algorithm assigns a match score to each pair of sequences based on the similarities between them. Similarity is determined by the number of matches, mismatches, and insertions/deletions between them. In a pair of sequences, an insertion in the first is logically equivalent to a deletion in the second. Therefore, insertions and deletions are typically combined into a single instance known as *indels*, or simply *spaces* (Gusfield, 1997). Gaps are defined as a maximal, consecutive run of spaces in one string that allows for optimal matching with the second (Gusfield, 1997). Therefore, a gap may consist of one or more spaces.

Scoring parameters W for each match (W_m), mismatch (W_{ms}), and gap (W_g) possibility are held in a scoring matrix and are typically determined through empirical methods (Xia, 2007). Gaps are frequently scored through an affine gap weight model in the form of $W_g = W_o + qW_s$, which assigns an initial penalty (W_o) known as the *gap opening penalty* or *gap initiation penalty*, followed by a *gap extension penalty* (W_s) for each additional space in the gap (Gusfield, 1997; Xia, 2007). Combining these parameters in the form of $\sum W_m - \sum W_{ms} - \sum W_g$ provides a simple method for scoring each sequence.

Sequence alignment algorithms are easily extended to natural language processing, though scoring poses significant challenges. The set of all word pairs in any given language is vastly larger than that of protein or nucleotide pairings, which makes creation of an explicit scoring matrix impractical. Other alternatives for natural language processing are constant scoring or scoring based on edit distance, morphological similarity, or semantic similarity between words (Bourdaillet & Ganascia, 2006).

There are two major approaches to sequence alignment; global alignment, which matches entire sequences with one another, and local alignment, which searches for optimal subsequence matches between each sequence.

Global Sequence Alignment

The Needleman-Wunsch algorithm is one of the earliest sequence alignment algorithms, first presented in 1970. The algorithm was developed to locate the largest overlap of amino acids between two proteins, accounting for gaps in alignment (Needleman & Wunsch, 1970).

Global alignment is performance intensive, running quadratic or even cubic in time (Gusfield, 1997). In practice, global alignment algorithms are most useful when comparing sequences of roughly equal length or when the positions of elements between each sequence are in similar orders or arrangements (Polyanovsky, Roytberg, & Tumanyan, 2011).

Local Sequence Alignment

In contrast, local alignment focuses on matching in cases of sequences of significantly different length, or when the optimal match between asymmetric sequences is needed (Polyanovsky et al., 2011). In particular, the focus of local alignment is to locate the optimal match between sequences when multiple matches and multiple sources may be present (Gusfield, 1997).

The earliest local alignment algorithm was presented by (Smith & Waterman, 1981). Known as the Smith-Waterman algorithm, this algorithm focuses on optimal matching of local sequences of all lengths. This algorithm runs in quadratic time, though

newer options such as BLAST and FASTA are more time efficient and extensively used in modern bioinformatics and computational biology (Polyanovsky et al., 2011).

The Cite Unseen Approach

Cite Unseen uses a local sequence alignment algorithm to identify significant word sequences. The Cite Unseen algorithm looks for notable sequences of matches by tracking the proximity of matched *n*-grams that share a common URL in their search results. Several *n*-grams in relatively close proximity that share a common URL in their search results indicates increased similarity between the URL and the submitted document and strongly suggests plagiarism or text reuse in that section of the document. This approach avoids the problems of more basic approaches outlined earlier; document length is no longer critical and short matches that are not in close proximity are significantly devalued. Thus, the likelihood of false positives is greatly reduced over simpler methods such as Equation 1: Resemblance and Equation 2: Similarity.

However, there are major differences between the Cite Unseen approach and traditional local sequence alignment algorithms. In particular, while Cite Unseen has a full copy of the submitted document, it does not have a full copy of each possible source document. Instead, a fragmented version of each possible source document is constructed from the URL search results. This fragmented version is an unordered set of *n*-grams, lacking position information for each matched *n*-gram. An *n*-gram that returns a particular URL in search results indicates only that the *n*-gram is present in the URL content, but not where or in relation to what other *n*-grams. This lack of position information prevents identification of mismatches between the submitted document and a given URL. As position information for matched *n*-grams in the source URL is not

known, any break in a match sequence with the submitted document can only be seen as a gap.

Due to this inability to recognize mismatches, Cite Unseen needs only the two scoring categories for matches and gaps, and has no category for mismatches. While this limits the matching ability of the algorithm, it has the added benefit of simplification, which results in improved speed. The end result is a scoring function s, such that:

$$s(S_D, URL_M) = \sum W_m - \sum W_g$$

Equation 3: Cite Unseen Scoring Function

In Equation 3, S_D is some sequence of *n*-grams in the submitted document D and URL_M is some URL in the search results M.

Cite Unseen explores several scoring methods for assigning match and gap weights. The most basic version assigns a match score of W_m= 1 and a linear gap penalty of $W_g = 1 \cdot q$ where q is defined as the number of consecutive unmatched *n*-grams.

Locating the most promising possible source URLs via this method is performed through several steps detailed below.

Converting URL Matches to Numeric Arrays

After completion of the searching phase, each *n*-gram is associated with a set of URLs returned as search results, which are collectively referred to as m'. The collection m' is then merged, with all duplication removed, resulting in a new set m of unique URL matches. Analysis of each unique URL in m begins by constructing a numeric array representation of each URL's match results to the suspect document. Any *n*-gram in the submitted document that matches with a particular URL is represented by a positive

weight value, while any *n*-gram that does not match with a particular URL is represented by a negative weight value.

This process is illustrated using the trigrams from Table 1 compared to Example 1: Source Wikipedia text, with base values of $+1$ for a match and -1 for a missing match:

Trigram	**Score**	**Trigram**	**Score**
plagiarism is not	+1	ethical offense in	-1
is not exactly	-1	offense in academia	-1
not exactly a	-1	in academia and	+1
exactly a crime	-1	academia and industry	+1
a crime but	-1	and industry cases	-1
crime but it	-1	industry cases of	-1
but it is	-1	cases of plagiarism	+1
it is a	+1	of plagiarism sometimes	-1
is a serious	+1	plagiarism sometimes constitute	-1
a serious ethical	+1	sometimes constitute copyright	-1
serious ethical offense	+1	constitute copyright infringement	+1

Table 3: Trigrams scored against given URL

In this example, each *n*-gram is calculated individually, resulting in t operations and $O(t)$ time complexity for each URL in m. However, when all *n*-grams are weighted equally, only one weight per string of matches or gaps needs to be calculated, which can reduce run time. This concept is discussed in more detail in Chapter 5 .

Next, sequences of positive and negative values are summed; positive to positive, negative to negative. This process creates an array of values that represents the number of consecutive matched trigrams and mismatched trigrams in the suspect text:

1	-6	4	-2	2	-2	1	-3	1

Figure 1: An array of trigram sequence scores

Finally, a value of $n - 1$ is added to each score in Figure 1; these adjusted scores are shown in Figure 2. Since each score in Figure 1 represents the number of successive trigrams and trigrams represent groups of three words, adding two to each score will roughly indicate the number of sequential words. For instance, one trigram is equal to three words and two consecutive trigrams are equal to four consecutive words. Likewise, one missing trigram indicates a gap of at least one word, while two missing consecutive trigrams indicates a gap of at least two consecutive words. Figure 2 represents these approximate word sequence scores.

3	-4	6	0	4	0	3	-1	3

Figure 2: An array of approximate word sequence scores

Overlaying the values from Figure 2 onto Example 2: Suspect plagiarized text illustrates the relation between the numeric and textual representations. Note that the zero values are simply ignored; a zero value indicates the phrases before and after the zero value were not adjacent in the original document, but rearranged by the author to be adjacent in the suspect document.

> ***Plagiarism is not*** *exactly a crime, but* ***it is a serious ethical offense in academia and industry***. ***Cases of plagiarism*** *sometimes* ***constitute copyright infringement***.

Example 5: Suspect text with approximate word matches in bold

Comparing Example 5 to Example 4 shows that this method works well. The only trigram missed is “a crime but”, which does not appear as a trigram in the original source text at all. Instead, it appears as the bigram “a crime” and the single word “but”. So it is not surprising that these matches were missed.

Locating Significant Subarrays

While Example 5 makes it visually obvious that plagiarism has been detected, further analysis to confirm plagiarism is essential. Although most *n*-grams are rare, it is reasonable to expect some matches even in the case of non-plagiarism. Simply highlighting all matched sections as in Example 5 is essentially the basic similarity approach outlined in Equation 2, with a high likelihood of false positive matches.

A simple way to avoid false positives is to set a scoring threshold; any score above a certain threshold would be considered likely plagiarism. However, this is a double edged sword; too high a threshold can miss plagiarism, while a threshold too low allows false positives. For instance, with a threshold of five applied to the array of scores in Figure 2, only the value of six would be highlighted, which equates to:

> *Plagiarism is not exactly a crime, but* ***it is a serious ethical offense*** *in academia and industry. Cases of plagiarism sometimes constitute copyright infringement.*

Example 6: Text with five or more matching, consecutive words highlighted

This example certainly appears suspicious, but ignores other areas of matching text that make it obvious that plagiarism is present. Although these other areas may be smaller and not of interest on their own, they are significant when taken together. To avoid this problem, significant relationships between groups of matching words must be identified.

To identify such significant relationships, Cite Unseen uses a fast, efficient dynamic programming algorithm based on the principles of local sequence alignment. With this algorithm, Cite Unseen locates significant subarrays within the numeric match array of each possible source URL.

Kadane's Maximum Subarray Algorithm

Finding the subarray of highest net value within an array of values is known as the "maximum subarray problem" and is done quite easily in linear time using Kadane's algorithm (Bentley, 1984). However, for the algorithm to serve the purpose of identifying all possible locations of plagiarism, it must locate not only the maximum subarray, but any subarray with a positive value; in particular, subarrays with values at or above the designated threshold. Some simple modifications to Kadane's algorithm provide this additional functionality, as outlined below.

Kadane's maximum subarray algorithm steps through each value of the array and maintains two values; the **current** running total for this subarray, and the **max** total seen so far in this subarray. If the value of **current** ever drops below zero, **current** is reset to zero, indicating the start of a new subarray. If the value of **current** is ever greater than **max**, **current** becomes the new value of **max**. This process is demonstrated with the array of values from Figure 2.

Array	3	-4	6	0	4	0	3	-1	3
Current	3	0	6	6	10	10	13	12	15
Max	3		6				13		**15**

Figure 3: Kadane's algorithm to find the maximum subarray value

By this process, Kadane's algorithm easily identifies the maximum subarray value of fifteen for this array.

Cite Unseen Positive Subarray Algorithm

Cite Unseen expands on Kadane's algorithm to locate all significant word sequences for a given URL. It maintains the numeric array representation of the URL and tracks all positive subarrays by including a few additional variables and calculations.

Let **i** be the number of the current subarray, **score[i]** be the value of the current subarray (replacing **max**), and **start[i]** and **end[i]** be the start and end locations of the current subarray. The new algorithm recognizes the **start** of each subarray as the first cell in the array after **current** falls below zero, while the **end** is the cell where the **score** of this subarray was last set equal to **current**.

Array	3	-4	6	0	4	0	3	-1	3
Current	3	0	6	6	10	10	13	12	15
Score	**3**		6				13		**15**

Figure 4: Algorithm to find positive subarray values and positions

This figure shows two subarrays of positive value; one at [1, 1] with a value of three and one at [3, 9] with a value of fifteen. Highlighting these matches in the suspect text shows an identical match with Example 4.

> ***Plagiarism is not*** *exactly a crime, but* ***it is a serious ethical offense in academia and industry. Cases of plagiarism*** *sometimes* ***constitute copyright infringement.***

Example 7: Matching text with positive subarrays highlighted

Although successful here, one pass of the algorithm is not always sufficient to find all positive subarrays. It is necessary to run the algorithm repeatedly until all significant subarrays are found. Consider the following hypothetical array of values after processing by the algorithm.

Array	4	-3	2	-6	10	-4	6	-4	2	0	1
Current	4	1	3	0	10	6	12	8	10	10	11
Score	**4**				10		**12**				

Figure 5: Hypothetical array after processing

It immediately becomes clear with this example that several positive values are not included in either of the two identified subarrays. This happens because processing for each subarray ends only when **current** drops below zero. This is necessary because

while **current** is above zero, a future value could increase the max **score** of the current subarray. Thus, subarrays can be missed after a max **score** has already been found but prior to reaching a point where **current** drops below zero. In order to correct for this problem, the algorithm must be run repeatedly across the same array, skipping any subarrays already calculated in the prior passes, and completing only when no subarrays remain.

Array	4	-3	2	-6	10	-4	6	-4	2	0	1
Current		0	2	0				0	2	0	3
Score			**2**						2		**3**

Figure 6: Hypothetical array after second pass, skipping known subarrays

Combining these results with the results of the first pass in Figure 5 correctly identifies all four subarrays and their scores.

Array	4	-3	2	-6	10	-4	6	-4	2	0	1
Score	**4**		**2**			**12**				**3**	

Figure 7: Combining the results of both passes

In this example, two passes is enough to identify all subarrays. However, scenarios exist where additional passes are necessary. In the worst case, each pass requires approximately $l/2i$ calculations, where i is the current iteration and l is the length of the starting array. Over the entire array, this results in $2l - 1$ total calculations in the worst case. Thus, despite the additional complexity beyond that of Kadane's algorithm, the positive subarray algorithm remains linear $O(l)$ in time.

Additional Scoring Methods

The effectiveness and utility of a local sequence alignment algorithm is determined by the chosen scoring parameters (Gusfield, 1997). The above example

assumes a base value of $+1$ for each matching *n*-gram and -1 for each missing *n*-gram between a suspect document and a possible source URL. However, these are not necessary the optimal scoring parameters. Cite Unseen consider two additional methods for scoring; weighting of matches at a constant factor and weighting of matches and/or mismatches by relative frequency.

Constant Factor Weighting

The first method simply adjusts the value of matches by a constant factor C, such that each match is worth C while each unmatched *n*-gram maintains a penalty of -1. This method allows for finer control over the value of matches in comparison to mismatches. Testing of various C values is discussed in Chapter 5 .

Weighting by *n*-Gram Frequency

Cite Unseen also explores adjusting the scoring values using the relative frequency of each match or gap. The potential impact of this approach can be seen in the total search results returned by Google for each of the original search trigrams from Table 2: URL match and result rank for each trigram, as shown below:

Trigram	Matches Returned
plagiarism is not	808k
is not exactly	387m
not exactly a	92.1m
exactly a crime	336k
a crime but	18.4m
crime but it	21.2m
but it is	4.29b
it is a	25.3b
is a serious	1.36b
a serious ethical	575k
serious ethical offense	55.7k

Table 4: Trigram search results

According to these results, certain trigrams are extremely common and others much less so. Zipf's law is obvious in this case, as some trigrams are several orders of magnitude more common than others. In particular, the trigrams "but it is" and "it is a" are extremely common, returning billions of matches. Somewhat surprisingly, "is a serious" also returns over a billion results. It would therefore seem that using the total search results returned by each query should provide a straightforward means for identifying significant and insignificant terms.

In fact, weighting query terms by relevance is a well explored problem in the field of information retrieval. In information retrieval, the primary focus is to retrieve the most relevant document based on a user query. One of the most common methods used in information retrieval for query and document weighting is known as term frequency-inverse document frequency.

Term Frequency – Inverse Document Frequency

Term frequency - inverse document frequency (TF-IDF) is a statistical method for determining the relevance between a search query and a document (Ramos, 2003). TF-IDF represents the relative frequency of a word in a specific document compared with the inverse proportion of occurrences of that word over an entire corpus of documents (Ramos, 2003). TF-IDF is used extensively in information retrieval to assist in determining the most relevant documents based on a user query. TF-IDF is the product of term frequency and inverse document frequency, each of which are calculated separately.

Term Frequency

Term frequency is a statistical measure of the relative frequency of a word in a specific document. The most simplistic representation of this is the *natural frequency* of

the term in the document, $tf(t,d) = f(t,d)$, where t represents the number of occurrences of a word in document d. Variations are possible, such as log-based term frequencies, and augmented term frequencies that compensate for the length of the document (Manning, Raghavan, & Schütze, 2009).

Inverse Document Frequency

Inverse document frequency is the inverse proportion of occurrences of a word over an entire corpus of documents (Ramos, 2003). IDF is critical in information retrieval. When a user submits a query with multiple terms, some method for weighting each query term must be introduce in order to identify the most significant terms (Manning et al., 2009).

The inverse document frequency (idf) of a term t is defined as:

$$idf_t = \log \frac{I}{df_t}$$

Equation 4: Inverse Document Frequency

In Equation 4, I is the index size, or number of documents in a corpus, and df is the number of documents in the corpus that contain t (Manning et al., 2009).

Cite Unseen Implementation

TF-IDF is used in information retrieval to retrieve relevant documents to a user query. However, it can be applied in the reverse case, determining the significance of a query that results in a particular document being retrieved.

In the case of Cite Unseen, a submitted document is broken down into queries of *n* words. There is no inherent relevance to each query term from the perspective of the

submitter. However, analysis of the search results returned does provide this information. By using the index size of the given search provider and the total number of results returned for a given query term, the IDF for that term can be calculated.

However, it is important to note that term frequency is difficult to determine from search results alone. Search providers return only a single instance of a particular URL no matter how many times the particular query term may appear in the URL document. It is possible for the term to appear in the description text of the search result more than once, but in general, the term frequency can only be estimated with a value of 1; this is essentially a Boolean true/false value for each term-document match. Assuming a value of 1 for term frequency eliminates this factor from the TF-IDF formula and results in IDF as the only calculation. Thus, in the case of Cite Unseen, TF-IDF is equivalent to the log probability for the term.

Log Probability

Log probability provides a computationally convenient way for expressing and working with probability. Multiplication of probabilities p represents the probability of all included events occurring together, assuming each event is independent of the others. This is equivalent to addition of log probabilities for the same values of p.

Cite Unseen explores using log probability to calculate match weight by defining W_m as follows:

$$W_m = -\log_{\mathrm{I}}(p) \cdot C$$

Equation 5: Log Probability Match Weight

In Equation 5, p is the probability of a match occurring, I is the index size, or number of documents in the corpus, and C is the constant weighting factor previously discussed, such that $0 \geq W_m \geq C$.

Cite Unseen explores a similar weighting method for gaps using the probability p of each unmatched *n*-gram k to determine a value for the overall gap penalty W_g.

$$W_g = -\sum_{k=1}^{q} \log_{\mathrm{I}}(p_k)$$

Equation 6: Log Probability Gap Weight

Thus W_g varies as $0 \geq W_g \geq q$ where q is defined as the number of consecutive unmatched *n*-grams between the given sequence and URL as defined in Equation 3.

The combination of these two weighting schemes creates a simple relation between the likelihood of each match occurring and the weight the algorithm gives to those matches or gaps.

A Note on Index Sizes

Methods for calculating a weighted value for each *n*-gram based on frequency require knowing the index size of the search provider. Unfortunately, with the exception of FAROO Search ("How big is your index?," 2014), none of the search providers tested by Cite Unseen publish their index sizes. There are several reasons for this, not simply corporate discretion. For instance, search engine indexes often contain multiple tiers of indexed pages, which may not all be returned for general results; these pages are indexed, but only appear in search results under specific circumstances (Manning et al., 2009). Thus, no accurate measure of an index size may exist. Search engine index size

estimation is its own area of extensive academic research with no easy solution (Broder et al., 2006).

In order to overcome this limitation, it is necessary to devise a consistent method to estimate index size across all search providers. Searching each provider for a common term, such as “of the” provides a simple option. Accuracy unfortunately cannot be assumed since estimating index size is an area of active research, but the hope is that consistency across all search providers normalizes results. Searching for this common term produces the following information for each search provider examined in this project.

Search API	Estimated Index Size
Bing Search	4.3 bn
Google Custom Search	5.8 bn
FAROO Search	2.0 bn†
Yahoo! BOSS Search	4.3 bn

Table 5: Estimated index sizes

† number provided by ("How big is your index?," 2014)

It is important to note that the search indexes offered by all search providers other than FAROO differ from the search indexes provided directly to users. The reasons for this are unclear, but it is consistent across providers. For instance, searching Google directly for “of the” produces 25 bn results, rather than the 5.8 bn returned by Google Custom Search. It is assumed this is because the direct web searches offered by each provider include sources that are not provided in the search APIs offered.

Identifying Word Sequences for Display

Each URL that appears in the search results for the suspect document is now represented by a series of subarrays of varying scores. These subarrays each represent a

sequence of words in the suspect document with a significant portion of those words matching between the suspect document and the given URL. The final step in identifying areas most likely to be plagiarized is to evaluate all subarrays for each URL and determine if they meet the minimum criteria to be displayed to the user.

Scoring by URL

The natural inclination at this stage is to develop a score for each URL based on the number and significance of each matching subsequence of words in the suspect document. Cite Unseen was initially developed with this method in mind. However, the primary challenge of this method is identifying a logical scoring method for each URL based on the subarray scores found by the Cite Unseen Positive Subarray Algorithm.

The most basic option is to simply sum all subarray scores for each URL to obtain a single score for each URL. However, this method has notable failings. Consider two example URLs; the first with three short subsequences after analysis by the algorithm, and the second with one long subsequence. The total lengths of these sequences per URL may be identical, but it is clear the single longer sequence from the second URL is likely to be the more significant. Thus, simple summation of sequence scores per URL is not a viable option.

Scoring by Sequence

Another approach is to consider each individual subarray and its corresponding word sequence as a unique entity and identify suspicious areas purely by sequences. This is the approach ultimately used by Cite Unseen.

Scoring by individual sequence has several advantages in practice. First, it avoids the problem of scoring an entire URL that may consist of several sequences of varying

values. Second, it allows for efficiently eliminating duplicate sequences that may exist between several URLs. This is typical as webpages often duplicate one another. By eliminating duplicate sequences, processing efficiency is increased. Lastly, since each sequence is scored on its own merits, this method addresses the case of plagiarists who may use disparate sources for plagiarism; for instance, copying from two or three related Wikipedia pages, rather than one single page.

In order to score by individual sequence, all sequences from all URLs are combined into one single set of sequences. Any duplication is eliminated, but tracked; if a sequence is associated with more than one URL, each of those URLs is maintained in a list associated with that sequence. Not all sequences of positive value indicate significant matches. Sequences that fall below a certain scoring threshold are eliminated from the set. Each remaining sequence maintains an associated "best" URL, which is defined as the URL that has the highest total similarity with the suspect document. For instance, if the same sequence appears in two URLs, and one URL has a total number of sixty matches with the entire suspect document while the other has only twenty matches, the first URL will be labeled as the best URL for that sequence. This URL is then used to represent that sequence to the user during presentation of results.

Presentation of Results

The final step outputs results to the user. Cite Unseen operates with series of default parameters easily controlled by the user at time of submission. These parameters, as well as the user experience and interface, are explored in more detail in the next chapter. The determination of the optimal value for each parameter is discussed in detail in Chapter 5 .

Chapter 3 User Interface

This chapter focuses on the Cite Unseen user interface. The first section provides a brief overview of the command line interface. The second section provides an in-depth walkthrough of the web interface.

Command Line Interface

Cite Unseen can be run from the command line on any computer that supports Java applications. The command line interface provides a simple method to run and test Cite Unseen on particular documents. The command line interface supports several flags, which are displayed by running the application with the '-?' help flag. Options control various features such as importing or exporting search caches for a document, choosing a particular search provider, controlling various scoring variables, and controlling the maximum connection and connection rate to the search provider.

Web Interface

The primary means to use Cite Unseen is through the web interface. The web interface runs via JavaServer Pages and a Java servlet. A screenshot of the Cite Unseen web interface summarizes the display and features of the system.

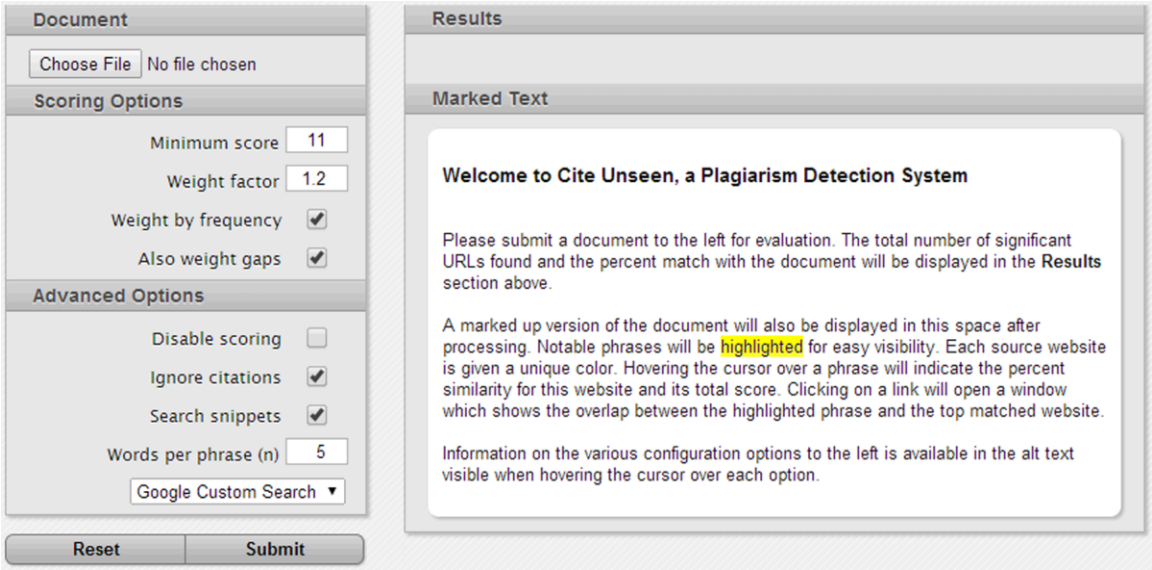

Figure 8: The Cite Unseen web interface

This interface provides quick access to several core features of Cite Unseen. The document submission field is the first and most important field in the system. Here the user can choose a document from their computer to analyze. The user can adjust the various parameters outlined in Chapter 2 such as score desired, the weight factor C, and whether to weight matches and gaps by their frequency.

Under Advanced Options, the user can disable scoring entirely, which disables the Cite Unseen processor and simply returns basic similarity matches with any found websites. The user can also choose to filter out citations and quotes from a submitted document, whether or not to include search result description parsing, and the value of n to use. The final option selects which supported search provider to use for searching.

Once a document has been submitted and processing is completed, the results are displayed to the user.

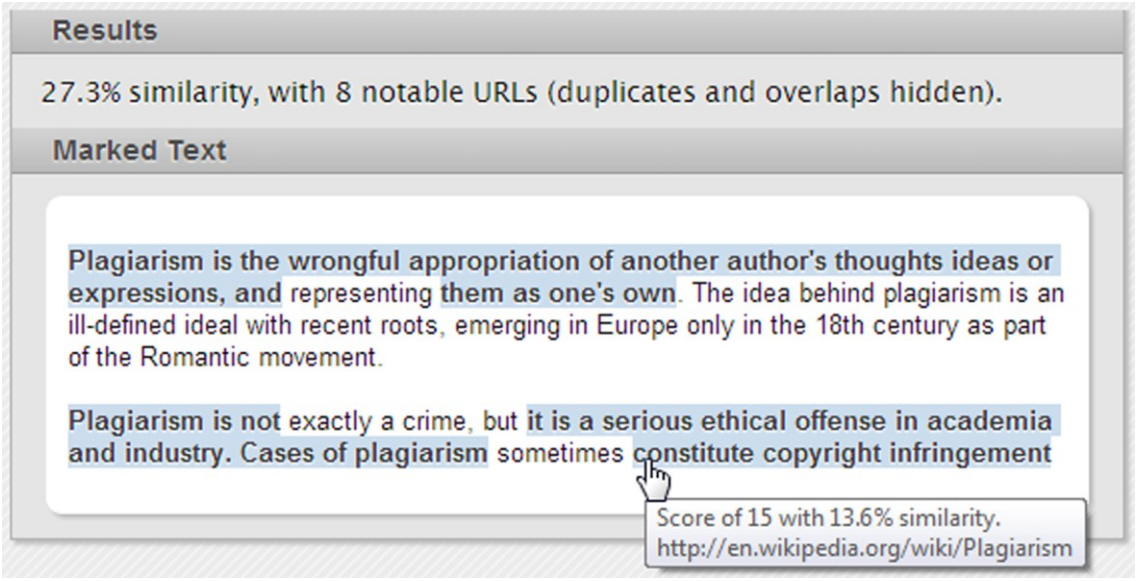

Figure 9: Displaying results to the user

This figure displays the results seen by the user after document processing has completed. The user has the option of hiding the options panel on the left to maximize the readability of the processed document. The options panel has already been hidden in this example.

The Results section displays the overall similarity between all matches and the submitted document, as well as the number of notable websites detected. The text area highlights sequences with scores above the chosen threshold. Each sequence is highlighted in a color unique to the source website of that sequence. Multiple sequences from the same website are displayed in the same color. This highlighting makes it visually obvious to the user when several copied sections of the document all link back to the same source.

Hovering the cursor over a particular highlighted section provides the user with information about that section, including all websites that matched that sequence, the total score of the sequence, and the similarity between the particular sequence and the

submitted document. If the user clicks on a particular sequence, a window pops up to display the highlighted sequence within the context of the source page.

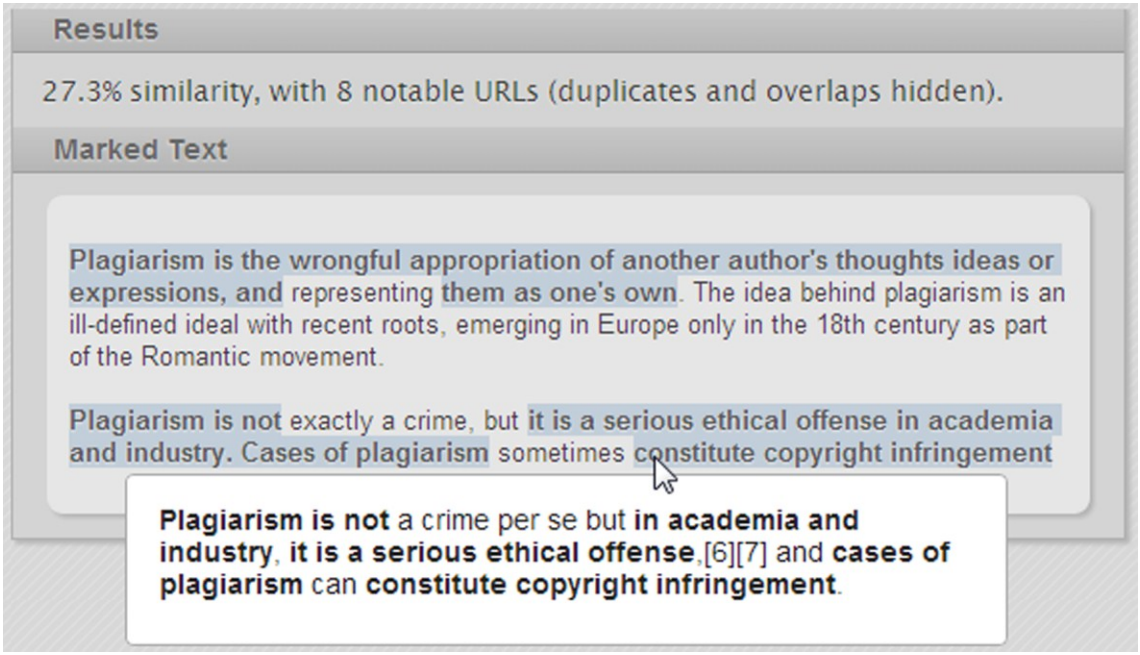

Figure 10: Displaying copied text in context with the original

Figure 10 demonstrates the matching of the text from Example 2 with the source text from Example 1. Displaying the copied text in the context of the original source provides a convenient method for the user to verify that the copied text came from a source of interest and helps in quickly eliminating false positive matches. It also helps the user determine if other areas of the document were copied and possibly missed by Cite Unseen.

The user can click anywhere outside the pop-up window to return to the main interface. If the user clicks on the bolded text in the pop-up window, the source webpage will open in a new window so the user can further investigate any matches.

Chapter 4 Design and Implementation

This chapter provides an overview of the programmatic design of Cite Unseen, the various technologies used, and the implementation of the Cite Unseen system. The first section details the various programming languages and technologies used to develop Cite Unseen. The second section discusses the public Java libraries used to augment the Cite Unseen system. The final sections describe the core classes and methods of the system.

Languages and Technologies

The core package of Cite Unseen is written in Java. The web interface is built using JavaServer Pages, HTML, CSS, and JQuery. The application runs via a command line interface or through a hosted webpage, interfacing with a Java servlet. For testing and development, Cite Unseen was run on an Apache Tomcat webserver in a cloud hosted CentOS Linux environment.

Dependencies

Cite Unseen interfaces with several public Java libraries. Each of these libraries and the role they play in the Cite Unseen process is detailed below.

Apache Lucene

Apache Lucene is an open source Java project focused primarily on text indexing and searching ("Apache Lucene," 2014). Cite Unseen interfaces with Apache Lucene through its Tokenizer and Analyzer interfaces, which are used to analyze and tokenize

text efficiently and intelligently. Cite Unseen analyzes submitted text, breaks it into *n*-grams and later converts these *n*-grams back into the original format of the submitted document with the assistance of these Lucene interfaces.

Apache Tika

Apache Tika is an open source Java toolkit for detecting and extracting text from a wide range of document formats ("Apache Tika," 2014). Cite Unseen interfaces with Tika in order to parse submitted documents into text strings for further processing. Apache Tika provides a fast and efficient means to extract text data from submitted documents.

Apache Commons HttpClient

The HttpClient library provides an alternative to the native Java Http libraries for basic http connections between client and server ("Apache HttpComponents," 2014). Cite Unseen uses the HttpClient library to build and submit GET requests for each *n*-gram query, as well as to manage the maximum concurrent connections made between Cite Unseen and the search provider.

Google Gson

Google Gson is a Java library that serializes and deserializes Java Objects using JavaScript Object Notation (JSON) ("Google Code," 2014). Gson is used by Cite Unseen to decode search results returned from each provider's search API and convert these search results into Cite Unseen SearchResult objects.

Core Classes and Methods

This section details the core classes within Cite Unseen and the methods that drive some of the more critical features as described in Chapter 2 The overall process of Cite Unseen and how each of the core classes plays a role is best illustrated by this core code snippet taken from SubmissionHandler.java:

```
// Tokenize our submitted text
TokenizedText submittedText = new SourceText(submission, n);

// Get search results
SearchEngine engine = new GoogleSearch();
Map<String, SearchResult> searchResults = engine.search(submittedText);

// Use our search results to find all word sequences of any interest
Processor processor = Processor.build().setMinimumScore(11);
Set<SourceFragment> sequences = processor.process(searchResults,
submittedText);

// Mark our sequences by their matching URLs in HTML format
Map<String, String> results = PageBuilder.asHTML(sequences,
submittedText);

return results;
```

Figure 11: Code sample of the Cite Unseen process

The remainder of this section is broken down by class type and presented generally in order of the appearance of each class in the processes outlined in Figure 11 and Chapter 2 . The first section details the TokenizedText interface, which encompasses classes that represent text within Cite Unseen and all methods for retrieving that text in various forms. The second section details the SearchEngine abstract class, which defines the basic methods shared by all search engine implementations. The third section details the Processor class and its essential methods, which perform the analysis necessary to identify locations of potentially plagiarized text. The final section briefly details remaining major classes and their functions.

The TokenizedText Interface

The TokenizedText interface provides a set of common methods for manipulating representations of text within Cite Unseen. This text may be the entirety of a submitted document, a fragment of that document, or a sequence of *n*-gram matches based on a particular source URL. TokenizedText is implemented by two classes within Cite Unseen; SourceText and SourceFragment. SourceFragment is also extended by several subclasses; NGram, URLResult, and Sequence.

TokenizedText requires implementation of various methods that allow for comparison of various TokenizedText objects. For instance, this interface allows a sequence of matched words to be compared to the original source document, determining the exact overlap location in the original submission.

The SourceText Class

The SourceText class represents a document submission. SourceText internally uses the Apache Tika and Lucene libraries to convert a submitted document of any supported format into a tokenized string of format-less *n*-grams. The value of *n* is defined at time of instantiation.

SourceText is responsible for providing various representations of the submitted document and answering questions about the content of that document. For instance, the document text can be provided in original format or a tokenized *n*-gram format. SourceText can also indicate if a given *n*-gram is contained within the document, where an *n*-gram is located, both in position compared to all other *n*-grams and absolute position in the originally formatted document. SourceText uses hash maps to store all text data.

Hash map storage allows fast lookup of document location or locations for any *n*-gram, or the *n*-gram for any location, in amortized $O(1)$ time.

The SourceFragment Class

The SourceFragment class represents all text fragments; that is, a sequence of tokens that are ordered but not necessarily contiguous. The SourceFragment class is extended by three subclasses; NGram, URLResult, and Sequence.

NGram

The NGram class is the most basic subclass of the SourceFragment class. It represents a single *n*-gram and its position or positions within the SourceText, where *n* is defined by the SourceText object that instantiated the NGram.

URLResult

The URLResult class represents the word sequence, array values, and duplicate URLs that are determined during the processes outlined in the Chapter 2 section Converting URL Matches to Numeric Arrays. Each URLResult also contains the URL link to the web document it represents and a set of all Sequences that share this URL.

Sequence

The Sequence class represents a set of proximal *n*-grams that share one or more common URL sources. Each Sequence is created and scored based on the processes detailed in the Chapter 2 section Locating Significant Subarrays. In final presentation to the user, the words represented by each Sequence are highlighted in the suspect document to indicate areas of possible plagiarism.

The SearchEngine Abstract Class

The SearchEngine class is an abstract class that provides fundamental, shared methods for interfacing with search provider APIs. This process allows for new search engine support to be added to Cite Unseen with great ease. The only information needed during implementation is the specific parameters of that engine, such as its query URL format and authentication methods, and instructions for how to parse the provider's SERPs. SERP parsing is handled by the abstract class SerpParser, which is extended to support whatever particular format a search provider API requires. The search providers currently supported by Cite Unseen use the JSON format, and so each implements the SerpParser extension JsonSerpParser. However, additional formats such as XML could be easily added by extending SerpParser in a new subclass.

Cite Unseen currently extends SearchEngine in five subclasses; GoogleSearch, YahooSearch, BingSearch, FAROOSearch, and OfflineSearch. OfflineSearch is primarily a test class for returning results from static, offline corpora. The use of this class is discussed in more detail in Chapter 5 .

Once instantiated, the specific SearchEngine implementation is passed a SourceText object representing the submitted document and returns a set of SearchResult objects. This set of SearchResult objects represents all search results returned by searches on the content of the submitted document. The SearchEngine class relies primarily on two other classes in order to produce and present these search results; SearchThread and SearchResult.

The SearchThread Class

The SearchThread class is a nested, internal class of SearchEngine that implements Callable. Implementation of Callable allows for multi-threaded searching as discussed in the Chapter 2 section Maximizing Search Speed. SearchThreads concurrently build a hash map of query strings to SERP strings using the Java Future interface. This class also interfaces with Apache Commons HttpClient to perform GET requests with the designated search provider.

The SearchResult Class

The SearchResult class represents the final search result for each query, after all SERP string parsing and other optimization is complete. This process is discussed in the Chapter 2 section Parsing Search Results. Each SearchResult object consists of a query term, the total number of results returned as a long value, and a set of matching URLs.

The Processor Class

The Processor class handles the bulk of the processing described in Chapter 2 . Processor is passed the set of SearchResult objects produced by SearchEngine, as well as the SourceText that represents the submitted document. Processor returns a set of Sequences that represent the most likely areas in SourceText to contain plagiarism, based on the processing parameters. These processing parameters can be specified during instantiation, most notably the minimum sequence score required and whether to consider the frequency of an *n*-gram when scoring a sequence. If no parameters are specified, Processor users default values determined to be optimal during testing and analysis. More detail on these default values is found in Chapter 5 .

Other Classes of Note

Cite Unseen uses many additional classes. The most notable are presented in alphabetical order here.

The ClickHandler Class

The ClickHandler class is a Java servlet class that intercepts user mouse clicks on highlighted sections in the web interface. The ClickHandler class is responsible for downloading a copy of the URL document associated with the clicked section and parsing the content of that URL to identify areas of overlap. The ClickHandler class then passes this highlighted text back to the web interface for display to the user, as in Figure 10.

By downloading source URL data only in the case of a direct user request, Cite Unseen minimizes unnecessary downloading and processing of possible source URLs.

The Dev Class

Dev is a utility class that contains various troubleshooting and diagnostic code. It also contains code for caching search results and retrieving cached data. In a production environment, the Dev class would serve no role within Cite Unseen.

The Main and SubmissionHandler Classes

These classes are responsible for interfacing with the user. The Main class provides a command line interface for Cite Unseen. The SubmissionHandler class runs as a Java servlet and interfaces with the user through JavaServer Pages.

The PageBuilder Class

PageBuilder contains only static methods. These methods are responsible for presenting a set of Sequences in a particular display format. Cite Unseen presents these results in HTML format. PageBuilder is responsible for querying SourceText for the original, formatted text of the submission, then highlighting the relevant Sequences via HTML. PageBuilder returns a string representation of this HTML for output to the user. This HTML format is displayed in detail in Chapter 3 .

The Test and BatchTest Classes

The Test class is a utility class for testing and analyzing the performance and results of Cite Unseen. The Test class works in concert with the BatchTest class to perform bulk processing of documents in succession and handle output of that data. The Test class and BatchTest class are the sources of all data presented in Chapter 5 .

The BatchTest class is built specifically to analyze passed Processor objects. Processor objects are constructed with various settings and passed to BatchTest. BatchTest then runs a batch of documents through the passed Processor and outputs the results. These results are represented in the form of FileTest subclass objects and are output in CSV format for viewing and analysis.

The Test class also provides methods for constructing offline indexes. Creating offline indexes allows for offline searching and processing, as well as controlled testing with source and suspect documents of known parameters. Artificially constructed offline indexes provide a means for deep analysis of the Cite Unseen methodology presented in Chapter 2 and the design and implementation outlined in this chapter.

Chapter 5 Results and Analysis

This chapter details the results and performance of Cite Unseen and its methodology as defined in Chapter 2 . The first section covers time complexity expectations, testing and results. The second section describes the testing parameters; the corpus used for testing and the scoring metrics used to analyze test results. The final section discusses and analyzes the test results and compares the optimal Cite Unseen settings to the baseline results in order to determine the absolute benefit of the Cite Unseen approach.

Time Complexity Testing

Throughout the development of Cite Unseen, efforts were made to maintain linear $O(t)$ performance, where t is the number of unique n-grams in a submitted document. Linear time is achieved for most settings of Cite Unseen, the exception being frequency weighting of all terms. In the case of frequency weight of all terms, evidence suggests Cite Unseen runs in linearithmic $O(t \log t)$ time. The mathematical case for run time and results verifying these expectations follow below.

Expectations

As described in Chapter 2 the initial t n-gram queries results in a set of m unique search results. For each URL in m, a numeric array of positive and negative weights is constructed. Due to the fixed number of search results per n-gram obtained during the searching phase, each n-gram in t is associated with a fixed number of URLs in m. Thus,

the mean matched *n*-grams per URL is equal to $\frac{t}{m}$. Therefore, if only matches are visited for each URL, the overall time complexity is $O\left(m\frac{t}{m}\right)$, or simply $O(t)$.

This relationship is also true for the sequence alignment algorithm defined in the Cite Unseen Positive Subarray Algorithm section of Chapter 2 . The algorithm is demonstrated to be linear in time to the length l of the given array. This algorithm must be run for each matching URL. Thus, if the set of all matching URLs has size m, the overall run time of for this phase is $O(lm)$. However, as l is also a function of the number of matches per URL in m, this step also remains linear in time.

Alternately, if the weight of each missed match is individually calculated, the time complexity increases because now all t of the *n*-grams must be visited. As mentioned in Chapter 2 the construction of a numeric array is $O(t)$ for each URL in m if each *n*-gram weight is calculated individually. This results in a total run time of $O(mt)$ if missed matches are also visited. Cite Unseen uses this approach when matches and gaps are weighted by individual *n*-gram frequency.

However, it is possible that $O(mt)$ is not quadratic, but linearithmic $O(t \log t)$. It is reasonable to assume that the larger the value of t, the higher the likelihood of duplication in search results. This relationship results in a reduced rate of increase of m as t increases, implying that $O(m) = O(\log t)$.

Testing and Results

In order to empirically verify the time complexity of Cite Unseen, performance testing was run on a series of files of varying lengths. Run times were then analyzed.

Each test was run excluding online searching in order to control for the variability of search provider response.

All testing was done using an n value of 3. Testing was performed on 187 files ranging in length from 2 trigrams to 12,720 trigrams. An average of 250 trigrams per page was used to estimate the number of pages per second, with the maximum page length tested being approximately 51 pages. Documents were either of Microsoft Word (doc or docx) format or plain text files. All testing was done on a Windows 7 64-bit system with 8 GB of RAM and an Intel i5-2410M CPU at 2.3 GHz. Each test was run ten times and the mean times used.

Three settings of Cite Unseen were tested for time performance. The first is the basic configuration with no frequency weighting. The second weights only matches by frequency. The third weights both matches and missed matches by frequency. Testing of constant factor C values and snippet searching was not done, as these are both algorithmically constant in time.

The initial test was run using the basic settings of Cite Unseen, with no frequency weighting included.

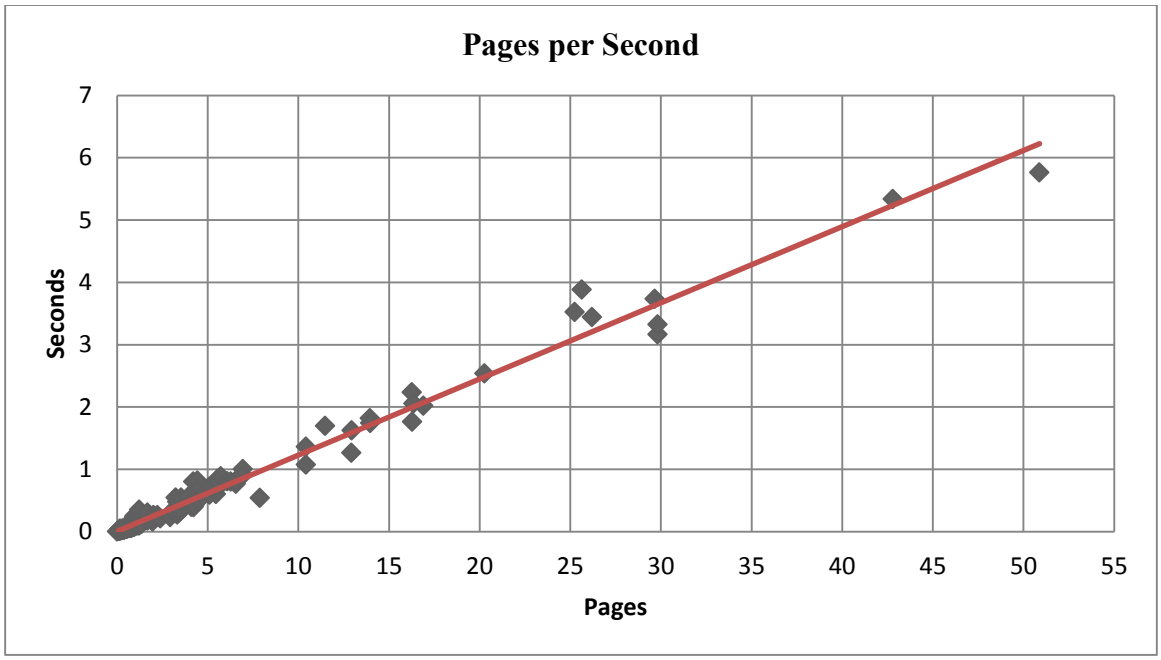

Figure 12: Time complexity

Testing strongly indicates $O(t)$ run time for the system, as expected. Testing was then repeated using frequency weighting for only matched terms.

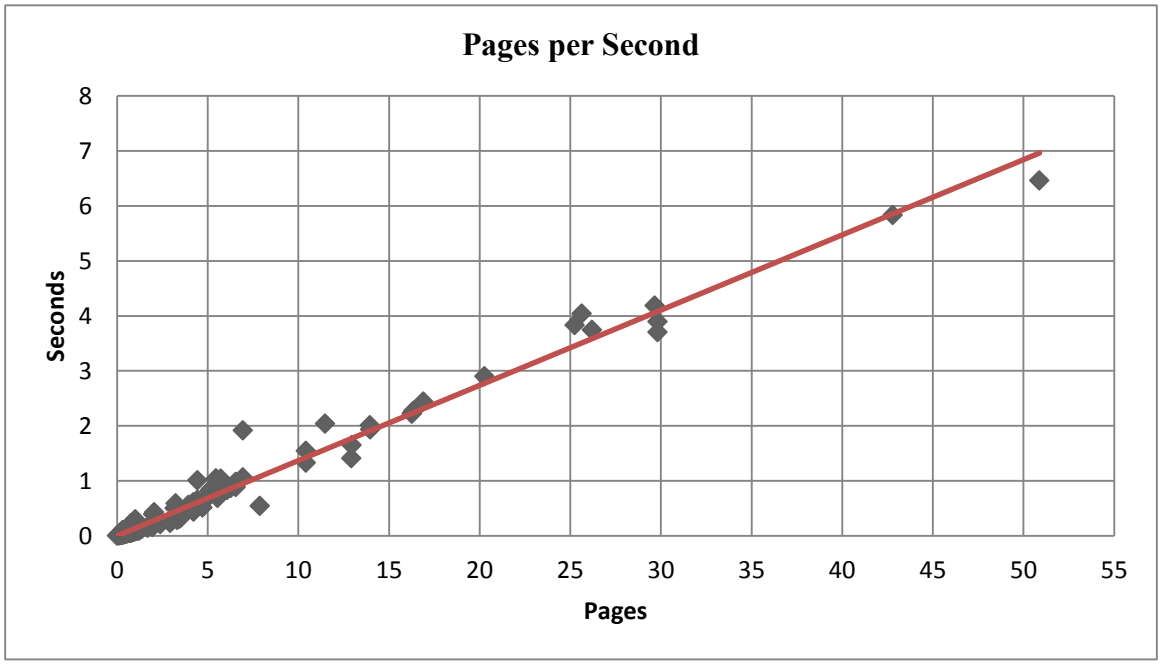

Figure 13: Time complexity with frequency weighting of matches

Testing with frequency weighting of matched terms indicates a slight performance penalty, but still appears linear in time. In both cases, documents of roughly 10 pages are completed in under 2 seconds.

Final testing was then performed with frequency weighting of all terms.

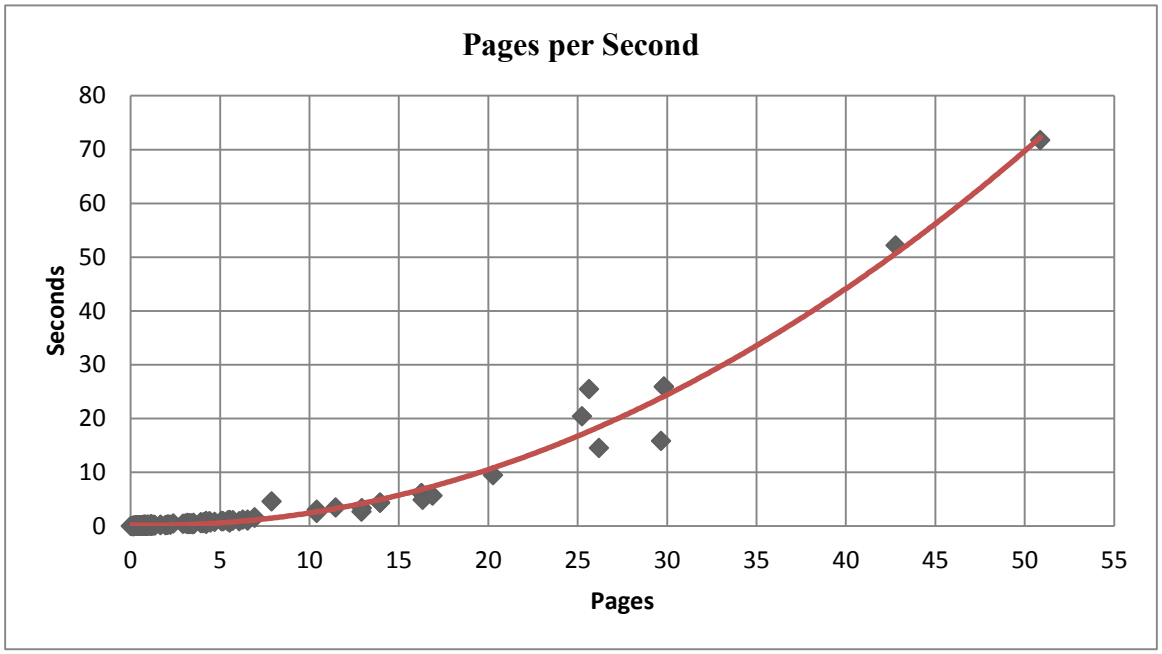

Figure 14: Time complexity with frequency weighting of all terms

Test results indicate a linearithmic $O(t \log t)$ trend, also as expected. However, the performance penalty is substantial for longer documents in comparison to the earlier tests. Nonetheless, a 10 page paper is still processed in under 3 seconds.

A final test was run measuring search results vs. *n*-gram count to see if a logarithmic trend was visible.

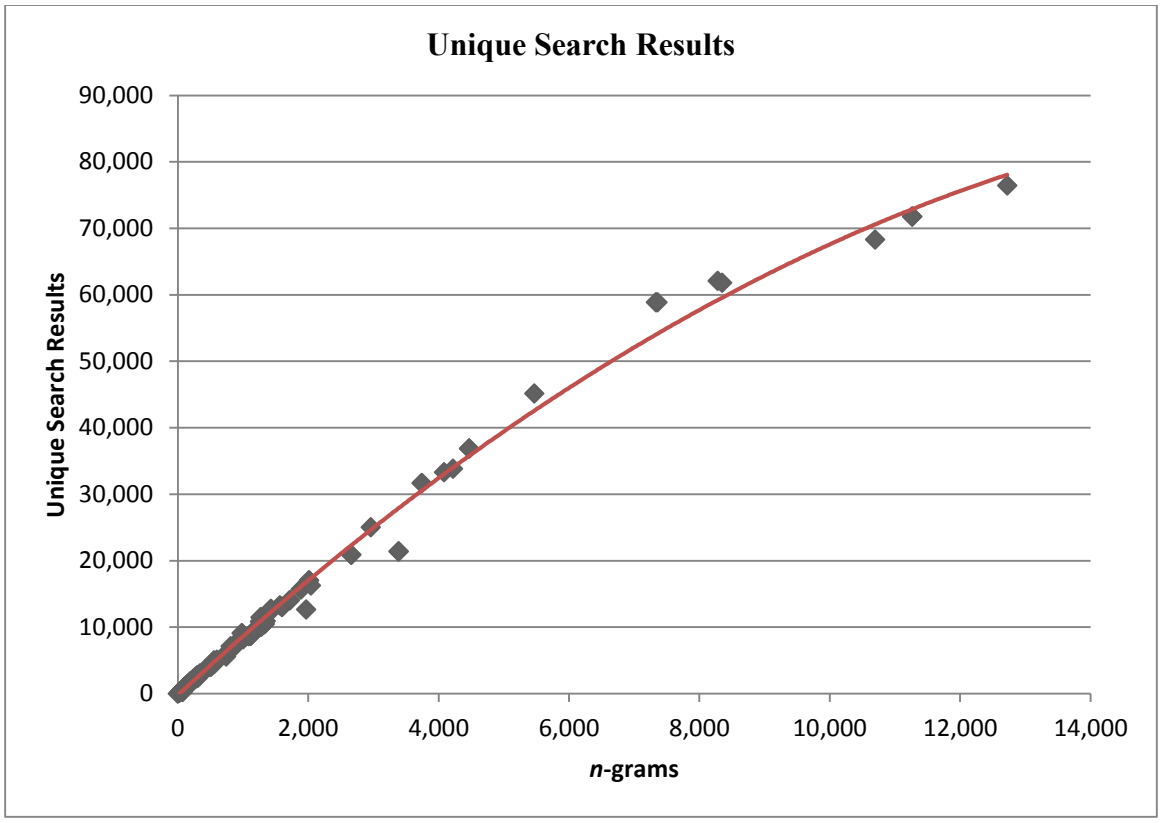

Figure 15: Unique search results by number of *n*-grams

While the curve does indicate a logarithmic trend, it does not appear to have a substantial effect until a very large number of *n*-grams, roughly equivalent to 40 or more document pages.

With time complexity established for each of the various scoring settings, testing then determined the effectiveness of each setting of Cite Unseen.

Testing Parameters

All testing of Cite Unseen was performed using the Corpus of Plagiarised Short Answers (Clough & Stevenson, 2011). Use of this corpus provided a controlled testing environment for both offline and online testing. Scoring metrics of *precision, recall,* and *F*-measure were used to quantify the effectiveness of Cite Unseen.

Corpus of Plagiarised Short Answers

Development and analysis of plagiarism detection methods necessarily requires extensive testing. However, due to the deceptive nature of plagiarism, construction of plagiarism detection corpora faces several challenges not encountered in the construction of corpora for other fields (Clough & Stevenson, 2011). Because of these challenges, no standardized test medium has been developed, though several artificially created corpora do exist (Clough & Stevenson, 2011). Typically, these corpora are created through automatic means, randomly moving words or inserting phrases between documents (Clough & Stevenson, 2011). However, such examples do not truly represent real-world plagiarism. More importantly, most artificially created corpora do not copy from internet sources, making them unsuitable for testing Cite Unseen.

The Corpus of Plagiarised Short Answers (COPSA) is a corpus of artificially plagiarized documents developed specifically to capture the types of plagiarism not usually included in other plagiarism detection corpora (Clough & Stevenson, 2011). Rather than produce documents through automated means, the researchers simulated plagiarism by recruiting several students to purposefully plagiarize answers to five designated questions.

Each of the five questions corresponded to a particular topic in computer science. Students were asked to answer each question with a different level of plagiarism, using the corresponding Wikipedia article on that topic as a source. The level of plagiarism varies between documents and is categorized at four levels; near copy, light revision, heavy revision, and non-plagiarism. The corpus consists of ninety-five short answers of 200-300 words and the five original Wikipedia articles.

Scoring Metrics

A standard set of metrics is used to determine the effectiveness of Cite Unseen in detecting plagiarism. These metrics are *precision*, *recall*, and *F-measure*. Each metric is described below. In all cases, a *true positive* (TP) indicates a returned document that is a source of plagiarism, a *false positive* (FP) is a returned document that is not a plagiarism source, and a *false negative* (FN) is a plagiarism source that exists but was not returned.

Precision

Precision, or positive predictive value, represents the fraction of possible source documents returned that are valid plagiarism sources. Precision is defined as:

$$Precision = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + |\mathrm{FP}|}$$

Equation 7: Precision

If no documents are returned, precision is defined with a value of 1.

Recall

Recall, or specificity, represents the fraction of valid sources that are found. Recall is defined as:

$$Recall = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + |\mathrm{FN}|}$$

Equation 8: Recall

If there are no sources to be found, recall is defined with a value of 1.

F-measure

The F-measure is the harmonic mean of precision and recall. The F-measure encapsulates both precision and recall, representing them as a single score and is a useful measure of how well information is retrieved. The F-measure is defined as:

$$F_\beta = \left(1 + \beta^2\right) \cdot \frac{Precision \cdot Recall}{(\beta^2 \cdot Precision) + Recall}$$

Equation 9: F-measure

The value of β varies depending on the weight placed on precision versus recall. A higher value for β results in more emphasis being placed on recall, while a lower value emphasizes precision. Typically used values for β are 1, 2, and 0.5. The standard implementation of F-measure uses a value of 1 to equally emphasize precision and recall. For testing of Cite Unseen, F_1 and F_2 are both used. F_2 is included as it is reasonable to assume that locating a plagiarism source is more critical to a human reviewer than sifting through a small number of false positives (Potthast et al., 2013). However, the F_1 score is still considered the primary metric, with the F_2 score highlighted as a secondary measure of effectiveness.

Test Design

Each Cite Unseen test is designed to test a different feature of the Cite Unseen system. The effectiveness of each feature is then evaluated by the precision, recall, and F-measure of that feature test. The primary feature tested is the local sequence alignment detection and scoring algorithm. Additional tests are performed to measure the benefits of URL description text searching, constant factor weighting of matches, and frequency

weighting of matches and mismatches. A determination of similarity with no features of Cite Unseen enabled is used as a baseline for comparison for each test. All tests are run with n values of 1 through 5. Optimal results for each test are determined primarily by highest F_1 measure, followed by highest F_2 measure, and finally by lowest sequence alignment scoring threshold.

Although the COPSA corpus is based on internet sources, Wikipedia is frequently updated and there is little chance that the articles of today are identical to the articles as they were at the time the corpus was created in 2009. As a result, accepting only the original Wikipedia pages as the only correct source documents is not likely a viable method for testing Cite Unseen with the COPSA corpus.

To circumvent this issue and test the effectiveness of Cite Unseen, two sets of tests were designed. The first set runs entirely offline, processing the COPSA corpus against only the original source Wikipedia articles. The second set of tests is performed entirely online with certain allowances to compensate for the revisions in each Wikipedia article.

Offline Test Design

The offline tests search against only the five source articles in their original formats. Searching against only the original source articles creates a useful baseline for comparing the various features of Cite Unseen and determining their effectiveness. To support offline searching, Cite Unseen implements an OfflineSearch extension of the SearchEngine class described in Chapter 4 . A small index of the original five documents was created for searching to support this test.

An initial test is first run to establish the similarity, precision, recall, and F-measure with no features of Cite Unseen enabled. Then the test is run with the sequence alignment algorithm enable and results are compared with the baseline.

Online Test Design

Because copying from Wikipedia is a common occurrence, there is some likelihood that the articles as they were at the time of COPSA's creation still exist in some form on the internet. Thus, searching online is reasonable if it includes a wider range of sources than just the original five Wikipedia articles. To determine the amount of overlap between the original Wikipedia articles, searches were performed on each of the original articles and the returned sources quantified. As an example, below are the results with $n = 3$ and scoring at a value of 10.

	Similarity with	
Original	**Current Version**	**All Sources**
Taska	4.9%	97.4%
Taskb	49.0%	98.7%
Taskc	89.5%	97.5%
Taskd	0.4%	93.4%
Taske	19.4%	82.0%

Table 6: Similarity between current and original Wikipedia articles

As can be seen, there is a wide range of variance in the level of editing in each article since the creation of the corpus. However, the overall similarity between the original articles and all current internet sources is still very high. Thus, to account for the varying level of differences between the original and current Wikipedia articles, a wider range of sources must be allowed.

This wider range of sources is defined as follows. *Primary sources* are defined as sources that rank the highest for each area of the document. Primary sources are identical

to the sources displayed to the user after a manual submission through Cite Unseen. *Secondary sources* are defined as all other matching sources that exceeded the designated scoring threshold, but were not displayed due to overlap with a higher scoring source.

In the case of a match during testing, a match is considered *true positive* for the purposes of calculating precision if the match is included in the primary or secondary sources for the corresponding original source Wikipedia article.

In order to calculate recall accurately, a more involved approach is necessary. Since each document in the corpus corresponds to no more than one source Wikipedia article, a match with *any* source in the primary sources is considered a match with the original. A match with any secondary source is weighted at half value for the purposes of recall. Therefore, for each tested document, recall will have a value of 1.0 or 0.5; 1.0 in the case of one or more matches with a primary source, and 0.5 in the case of one or more matches with a secondary source, but no matches with a primary source.

After establishment of primary and secondary sources for each original source Wikipedia article, an initial test is run to establish the similarity, precision, recall, and F-measure with no features of Cite Unseen enabled. Each feature is then tested and compared with this baseline.

Additional Test Designs

After testing of offline and online performance, final tests will be performed to analyze additional features of the Cite Unseen system. First, weighting of matches by a constant factor C is tested. Next, the weighting of matches by frequency discussed in the section Weighting by *n-Gram* Frequency is tested and results analyzed. Finally, all options are combined to determine the optimal settings for plagiarism detection. All these

options are tested using the same parameters as described in the Online Test Design section above.

Test Results

The below sections summarize the test results from each test described in the Test Design section. First, offline test results are presented, followed by online tests results. Finally, results and analysis of the additional options of the Cite Unseen system are presented.

Offline Test Results

The initial test runs with no features of Cite Unseen enabled, in order to establish a baseline for comparison. The highest values for each metric are highlighted.

n	Total Files	Similarity	Precision	Recall	F_1	F_2
1	100	0.83	0.26	1.00	0.41	0.64
2	100	0.49	0.28	1.00	0.44	0.66
3	100	0.35	0.47	0.99	0.64	0.81
4	100	0.30	0.74	0.98	0.84	0.92
5	**100**	**0.27**	**0.95**	**0.97**	**0.96**	**0.97**

Table 7: Baseline results for offline testing

It can be immediately seen that recall is generally quite high. This result follows from the fact that the index consists only of the original five documents and so the chances of missing an original source is relatively low. However, precision varies substantially, and does not reach an acceptable level until n of 4 or higher.

Testing then continues for each value of n and for a scoring threshold ranging from 1 to 20. For sake of brevity, the results only for n = 3 and scores 1 through 10 are shown below as an example.

Score	Total Files	Similarity	Precision	Recall	F_1	F_2
1	100	0.35	0.46	0.99	0.63	0.80
2	100	0.35	0.46	0.99	0.63	0.80
3	100	0.35	0.46	0.99	0.63	0.80
4	100	0.33	0.69	0.98	0.81	0.90
5	100	0.32	0.83	0.97	0.89	0.94
6	100	0.32	0.87	0.97	0.92	0.95
7	100	0.32	0.95	0.97	0.96	0.97
8	**100**	**0.31**	**0.97**	**0.97**	**0.97**	**0.97**
9	100	0.31	0.98	0.95	0.96	0.96
10	100	0.31	0.98	0.94	0.96	0.95

Table 8: Example of offline test run for $n = 3$

As can be seen, a score value of 8 returns the best results. In fact, when compared with the baseline data for $n = 3$ in Table 7, a considerable improvement can be seen. Precision increases by 0.5, recall decreases by only 0.02, F_1 increases by 0.33 and F_2 increases by 0.16.

The best results for each value of n are summarized below.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	20	0.61	0.64	0.99	0.78	0.89
2	**15**	**0.36**	**0.99**	**0.97**	**0.98**	**0.97**
3	8	0.31	0.97	0.97	0.97	0.97
4	7	0.29	0.97	0.96	0.96	0.96
5	5	0.27	0.95	0.97	0.96	0.97

Table 9: Summary of offline results for n of 1 through 5

The values of $n = 2$ with a score of 15 is the optimal setting. However, $n = 3$ and a score of 8 is likely a more practical combination, as a higher score will result in an increased chance of missing short phrases.

Finally, the overall increase for each value of n is shown to summarize the net benefit of the Cite Unseen sequence alignment algorithm.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	20	-0.22	+0.38	-0.01	+0.37	+0.25
2	15	-0.13	+0.71	-0.03	+0.54	+0.31
3	8	-0.04	+0.50	-0.02	+0.33	+0.16
4	7	-0.01	+0.23	-0.02	+0.12	+0.04
5	5	0.00	0.00	0.00	0.00	0.00

Table 10: Delta of sequence alignment impact

Even in this simple example it is readily apparent that the sequence alignment method has substantial benefits beyond the basic similarity method described in Equation 2: Similarity. However, online testing provides a much more realistic view of these improvements.

Online Test Results

Online testing follows a similar methodology to the offline testing above, allowing for the modification as described in the Online Test Design section. However, one small modification was made to the corpus to ensure accurate test results.

During testing, it was noticed that several of the corpus documents were substantially copied from web sources unrelated to the original Wikipedia articles. This was not relevant in offline testing, as the documents were rightly categorized as plagiarized or not plagiarized in the majority of cases. However, in online testing, it was necessary to eliminate these documents to avoid false positives. These documents would match very highly with sources unrelated to the Wikipedia sources and skew results. In total, twelve documents were eliminated from the corpus during this stage, leaving eighty-eight documents for testing.

As in offline testing, the initial test was run with no features of Cite Unseen enabled, in order to establish a baseline for comparison. The highest values for each metric are highlighted here.

n	Total Files	Similarity	Precision	Recall	F_1	F_2
1	88	1.00	0.00	0.31	0.00	0.00
2	88	1.00	0.00	0.31	0.00	0.00
3	88	0.97	0.01	0.98	0.02	0.05
4	88	0.86	0.08	0.99	0.15	0.30
5	88	0.68	0.19	0.99	0.32	0.54

Table 11: Baseline results for online testing

It is readily apparent that online matching is a substantially more challenging task. While recall remains fairly high for values of $n = 3$, precision is extremely low in all cases and virtually non-existent for low values of n. This demonstrates that while it is possible to locate a correct source easily enough, actually recognizing it as correct among the false positives and presenting it to the user is a much more difficult proposition.

Summary Text Processing

The test is then repeated with search description text parsing as discussed in the section Utilizing URL Summary Text to Improve Search Results. The net benefit of parsing search descriptions is highlighted here.

n	Total Files	Similarity	Precision	Recall	F_1	F_2
1	88	1.00	0.02	0.85	0.04	0.09
2	88	1.00	0.03	0.95	0.06	0.13
3	88	0.97	0.15	0.99	0.26	0.47
4	88	0.86	0.23	0.98	0.37	0.59
5	88	0.69	0.27	0.98	0.42	0.64

Table 12: Results after description text parsing

n	Similarity	Precision	Recall	F_1	F_2
1	0.00	+0.02	+0.54	+0.04	+0.09
2	0.00	+0.03	+0.64	+0.06	+0.13
3	0.00	+0.14	+0.01	+0.24	+0.42
4	0.00	+0.15	-0.01	+0.22	+0.29
5	+0.01	+0.08	-0.01	+0.10	+0.10

Table 13: Delta of description text parsing

The benefit of additional parsing of search description text is quite apparent and easily justifies the additional processing time required for this feature. Although it does not increase precision numbers to levels that would be adequate on their own, it does provide a better foundation for later processing. In particular, it provides a larger number of matched *n*-grams for each source returned, which allows easier identification of significant sequences.

The data presented in Table 12 is now used as the baseline for all further testing of Cite Unseen in order to distinguish between description text benefits and the benefits of additional processing.

Sequence Alignment

As in the case of offline testing, all tests are run with values of n of 1 to 5 and in a scoring range of 1 through 20. Again, the results for n = 3 with scores of 1 through 20 are presented here as an example.

Score	Total Files	Similarity	Precision	Recall	F1	F2
1	88	0.97	0.07	0.99	0.13	0.27
2	88	0.97	0.07	0.99	0.13	0.27
3	88	0.97	0.07	0.99	0.13	0.27
4	88	0.81	0.13	0.99	0.23	0.43
5	88	0.61	0.25	0.98	0.40	0.62
6	88	0.49	0.36	0.98	0.53	0.73
7	88	0.44	0.50	0.98	0.66	0.82
8	88	0.41	0.66	0.98	0.79	0.89
9	88	0.40	0.72	0.96	0.82	0.90
10	88	0.39	0.77	0.95	0.85	0.91
11	88	0.39	0.78	0.93	0.85	0.90
12	88	0.38	0.83	0.90	0.86	0.89
13	88	0.37	0.89	0.89	0.89	0.89
14	88	0.37	0.88	0.89	0.88	0.89
15	**88**	**0.36**	**0.91**	**0.89**	**0.90**	**0.89**
16	88	0.36	0.91	0.89	0.90	0.89
17	88	0.35	0.90	0.88	0.89	0.88
18	88	0.35	0.90	0.86	0.88	0.87
19	88	0.35	0.90	0.85	0.87	0.86
20	88	0.34	0.91	0.84	0.87	0.85

Table 14: Example of online test run for $n = 3$

In this case, a much higher score of 15 is required to reach the maximum impact of the sequence alignment algorithm. Such a high score comes at a cost to recall, lowering it from 0.99 to 0.89, but has an enormous impact on precision, raising it from 0.15 to 0.91. Both F_1 and F_2 also rise substantially as a result of this increase. This large difference clearly demonstrates how well the algorithm performs in filtering out false positives and ensuring substantially more relevant results are presented to the user.

The impact of the algorithm is demonstrated very well in graphical form.

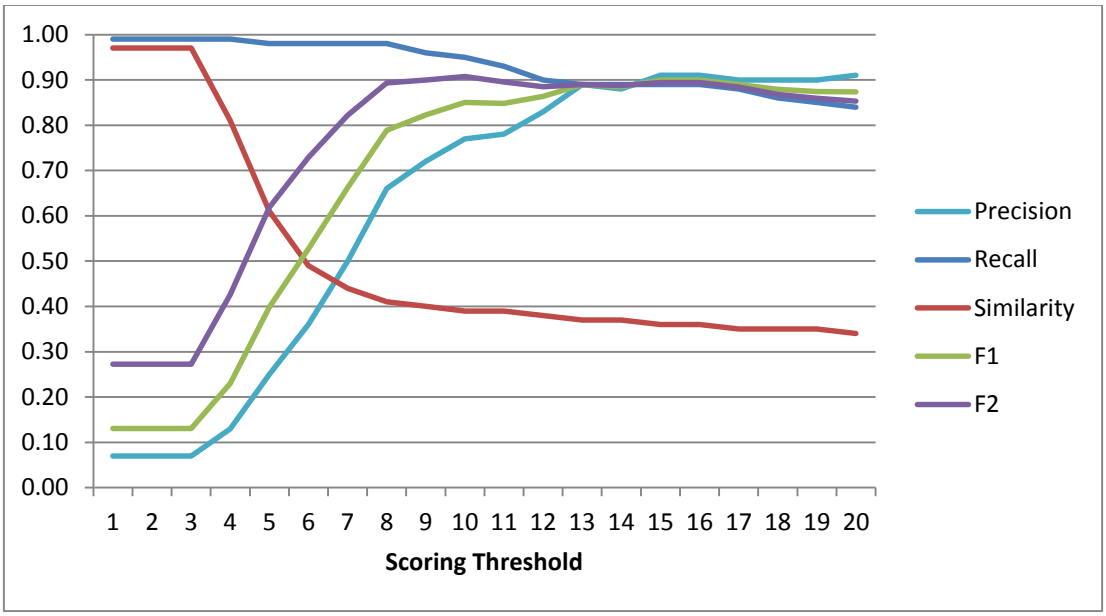

Figure 16: Graph of online test run for $n = 3$

This visual representation shows the rapid convergence of all metrics around the score of 13, and also the substantial decrease in similarity as the scoring rises, seemingly approaching equilibrium around 0.30. The best results for each value of n are summarized below.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	10	0.06	0.79	0.60	0.68	0.63
2	12	0.24	0.87	0.91	0.89	0.90
3	15	0.36	0.91	0.89	0.90	0.89
4	**16**	**0.39**	**0.89**	**0.91**	**0.90**	**0.91**
5	13	0.39	0.86	0.92	0.89	0.91

Table 15: Summary of online results for n of 1 through 5

In the online testing, $n = 4$ with a score of 16 is the optimal setting. This result challenges the notion that $n = 3$ is the optimal setting for plagiarism detection in the online environment. Although the difference is marginal, it does demonstrate that trigrams are not always the most obvious choice for detection.

Finally, the overall increase for each value of n is shown to summarize the net benefit of the Cite Unseen algorithm in the online environment.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	10	-0.94	+0.77	-0.25	+0.64	+0.54
2	12	-0.76	+0.84	-0.04	+0.83	+0.77
3	15	-0.61	+0.76	-0.10	+0.64	+0.43
4	16	-0.47	+0.66	-0.07	+0.53	+0.31
5	13	-0.30	+0.59	-0.06	+0.47	+0.27

Table 16: Delta demonstrating sequence alignment impact

The most substantial benefit is seen for $n = 2$, but all values of n greatly improve with only minimal loss of recall. In addition, scores seem to be somewhat consistently optimized in the low to mid teens, unlike in the offline environment where the benefit of the algorithm dropped off as n increased.

However, the high score requirement will have a notable penalty on missing more highly paraphrased or obfuscated text. Therefore, other options for maintaining the success of the algorithm while lowering the scoring requirement are investigated.

Additional Test Results

This section outlines the results of testing various additional features of the Cite Unseen system beyond the sequence alignment algorithm tested above. In particular, three options were tested; variations on the value of C as defined in the section Constant Factor Weighting, weighting of matches by the method defined in Equation 5: Log Probability Match Weight, weighting of gaps by the method defined in Equation 6: Log Probability Gap Weight, and lastly a combination of each of these methods.

To determine the impact of each option, a series of tests as outlined in the Online Test Design were run with various frequency weighting schemes. All tests were

performed on scores of 1 through 20 and n values of 1 through 5. All test results are compared with the results of Table 15 as a baseline.

Constant Factor Weighting

The first test attempts to isolate an optimal value of C from the section Constant Factor Weighting. Values of 0.1 to 2.00 were tested in increments of 0.1. Results are summarized below.

n	C	Score	Similarity	Precision	Recall	F_1	F_2
1	1.0	10	0.06	0.79	0.60	0.68	0.63
2	1.2	14	0.25	0.86	0.92	0.89	0.91
3	1.0	15	0.36	0.91	0.89	0.90	0.89
4	1.3	15	0.40	0.86	0.93	0.89	0.92
5	1.3	13	0.40	0.83	0.94	0.88	0.92

Table 17: Summary of *C* factor results for *n* of 1 through 5

n	Score	Similarity	Precision	Recall	F_1	F_2
1	0	0.00	0.00	0.00	0.00	0.00
2	+2	+0.01	-0.01	+0.01	0.00	+0.01
3	0	0.00	0.00	0.00	0.00	0.00
4	-1	+0.01	-0.03	+0.02	-0.01	+0.01
5	0	+0.01	-0.03	+0.02	-0.01	+0.01

Table 18: Delta demonstrating *C* factor impact

Weighting matches over mismatches by a constant factor has surprisingly little impact on overall performance. Most notably, it results in a slightly penalty to precision and slight gain in recall. These results indicate weighting by a constant factor is unlikely to have much benefit in plagiarism detection.

Match Frequency Weighting

The next set of tests examines the effects of frequency weighting of matches only. Frequency weights are based on the log probability of each match with $C = 1$ as defined in Equation 5. Results of n = 3 are displayed below in graph form.

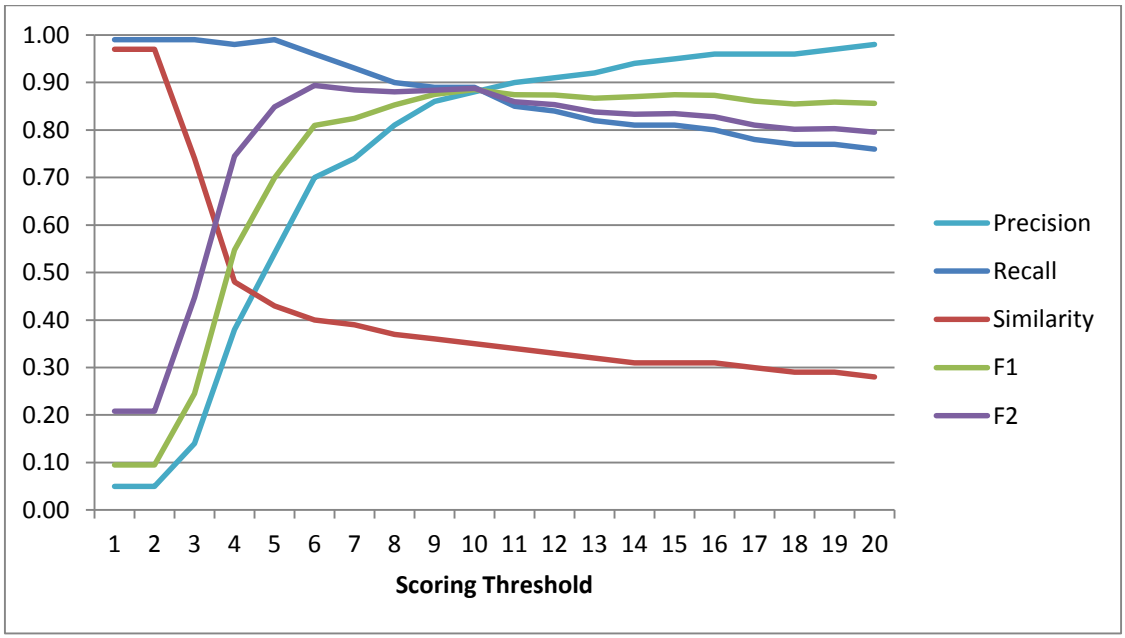

Figure 17: Graph of frequency weighting test for n = 3

Comparison of Figure 17 to Figure 16 visually demonstrates the benefits of frequency matching. The convergence of various metrics occurs earlier and then diverges more quickly resulting in a more obvious indicator of the optimal score.

Results for all values of n are outlined below and summarized by the best results per value of n.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	1	0.04	0.61	0.53	0.57	0.54
2	5	0.23	0.84	0.90	0.87	0.89
3	10	0.35	0.88	0.89	0.88	0.89
4	13	0.39	0.90	0.90	0.90	0.90
5	10	0.40	0.85	0.94	0.89	0.92

Table 19: Summary of frequency weighting results for *n* of 1 through 5

These results are best interpreted by direct comparison to the results in Table 15: Summary of online results for *n* of 1 through 5. This comparison is summarized below.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	-9	-0.02	-0.18	-0.07	-0.11	-0.09
2	-7	-0.01	-0.03	-0.01	-0.02	-0.01
3	-5	-0.01	-0.03	0.00	-0.01	-0.01
4	-3	0.00	+0.01	-0.01	0.00	-0.01
5	-3	+0.01	-0.01	+0.02	0.00	+0.01

Table 20: Delta of frequency weighting for matches

The overall impact on F-measure for most values of n is very minimal, while the effect on scoring is substantial. The change is most significant for $n = 2$, lowering the optimal score by 7 while maintaining very good F-measures.

However, it is important to note that by the nature of the frequency weighting implementation, lower scores are to be expected. Because each term is weighted by frequency, more frequent terms contribute less to the overall score of each sequence. The weight value range for matching terms when $C = 1$ is $0 \leq W_m \leq 1$. In the case of $n = 2$, it's likely that the majority of terms are so common, they provide much less than full value in each sequence, resulting in overall lower scores.

This assumption is validated by the fact that lower values of n have correspondingly smaller drops in optimal score. Thus, the goal is to look for improvements in F_1 and F_2 measures as well as drops in optimal score. In the case of

weighting for only matches, a value of $n = 5$ resulted in an improvement by this measure, with an increase of F_2 by 0.1, which is hardly notable. No other values of n saw any improvement.

In order to compensate for this score skewing, tests were then performed weighting both matches and mismatches by the frequency of the terms.

Match and Gap Frequency Weighting

Tests were performed identically to above, but also weighting gaps by the negative log probability of the corresponding missed match, as defined in Equation 6. Results are summarized below.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	4	0.15	0.67	0.48	0.56	0.51
2	13	0.21	0.92	0.81	0.86	0.83
3	12	0.36	0.88	0.89	0.88	0.89
4	13	0.39	0.88	0.92	0.90	0.91
5	12	0.39	0.87	0.94	0.90	0.93

Table 21: Summary of frequency weighting results for n of 1 through 5

n	Score	Similarity	Precision	Recall	F_1	F_2
1	-6	+0.09	-0.12	-0.12	-0.12	-0.12
2	+1	-0.03	+0.05	-0.10	-0.03	-0.07
3	-3	0.00	-0.03	0.00	-0.01	-0.01
4	-3	0.00	-0.01	+0.01	0.00	+0.01
5	-1	0.00	+0.01	+0.02	+0.01	+0.02

Table 22: Delta of frequency weighting for matches and mismatches

When both matches and mismatches are weighted by frequency, the results are only marginally better. Scoring drops less, which is to be expected, but there is also a very minimal improvement in F-measure for n values of 4 and 5. However, this

improvement now indicates that n = 5 with a score of 12 is the most successful setting for plagiarism detection based on the COPSA corpus.

Constant Factor and Frequency Weighting

The final test combines the optimal settings of all previous tests and compares the results to Table 15 to determine if any added benefit is realized. These results are summarized below for values of n of 1 through 5.

n	Match	Mismatch	C	Score	Similarity	Precision	Recall	F_1	F_2
1	normal	normal	1.0	10	0.06	0.79	0.60	0.68	0.63
2	by freq	normal	1.7	7	0.24	0.84	0.93	0.88	0.91
3	by freq	normal	1.3	11	0.36	0.88	0.90	0.89	0.90
4	by freq	by freq	1.2	13	0.40	0.86	0.94	0.90	0.92
5	by freq	by freq	1.2	11	0.40	0.85	0.95	0.90	0.93

Table 23: Summary of optimal results using all features of Cite Unseen

n	Score	Similarity	Precision	Recall	F_1	F_2
1	0	0.00	0.00	0.00	0.00	0.00
2	-5	0.00	-0.03	+0.02	-0.01	+0.01
3	-4	0.00	-0.03	+0.01	-0.01	+0.00
4	-3	+0.01	-0.03	+0.03	-0.00	+0.02
5	-2	+0.01	-0.01	+0.03	+0.01	+0.02

Table 24: Delta of optimal results using all features of Cite Unseen

The results indicate that n = 5 is still the optimal choice when all features are considered. However, combining frequency weighting of both matches and mismatches with a slight increase in C factor to emphasize matches over mismatches creates the optimal detection scenario. This combination results in a slight decrease in optimal score from 12 to 11 while maintaining the same F-measures. However, since raising the value of C also results in higher scores for all matches, the net benefit of a score reduction with a higher C value is more substantial than a simple comparison of the two scores. This

results in shorter word sequences being matched and displayed to the user while still maintaining the high F-measures.

Summary of Results

In order to demonstrate the full effectiveness of Cite Unseen, it is beneficial to compare the optimal final results of Table 23 with the initial baseline of similarity-based matching presented in Table 11.

n	Score	Similarity	Precision	Recall	F_1	F_2
1	10	-0.94	+0.79	+0.29	+0.68	+0.63
2	12	-0.76	+0.87	+0.60	+0.89	+0.90
3	15	-0.61	+0.90	-0.09	+0.88	+0.85
4	16	-0.47	+0.81	-0.08	+0.75	+0.60
5	13	-0.29	+0.67	-0.07	+0.57	+0.37

Table 25: Delta comparison of baseline results and optimal Cite Unseen settings

These results demonstrate the success of the Cite Unseen approach. The precision increases tremendously, which has the added benefit of reducing similarity substantially. By filtering the vast majority of false positives, the true similarity of the document to original sources is determined with much higher accuracy. While recall decreases slightly as a result of processing, the increase in precision is an order of magnitude larger and far outweighs any negatives of decreased recall.

It is worth noting that the optimal setting required frequency weighting of both matches and mismatches. However, the benefit was extremely small, only increasing F_2 by 0.01 in the best case of $n = 5$. Considering the substantial increase of run time required to perform both match and mismatch weighting, it is unlikely this small increase in results is worthwhile. Therefore, frequency weighting of matches only seems the clearly superior option when considering both time complexity and success of results.

Chapter 6 Summary and Conclusions

This chapter summarizes the Cite Unseen project. The first section provides an overview of the Cite Unseen approach to plagiarism detection and its success in detecting plagiarism. The second section discusses the more substantial problems and challenges encountered during development and how these impact the Cite Unseen approach to plagiarism detection. The final section outlines future research possibilities and areas of potential further development.

Project Overview and Summary

The results and analysis presented in Chapter 5 demonstrate the substantial success of the Cite Unseen approach to plagiarism detection and the completion of all objectives outlined in the Introduction. Utilization of the internet as a public corpus via existing search providers proves to be a viable option for plagiarism detection. Cite Unseen shows that specific plagiarism sources can be successfully located despite the high level of "noise" created by duplication among billions of internet pages.

In addition, the local sequence alignment method discussed in Chapter 2 proves to be very successful at identify plagiarism in text documents, with little performance overhead. Using a relatively standard contemporary computer, Cite Unseen is able to analyze and locate areas of plagiarism in a document of fifty pages in less than ten seconds. In general, the system runs in linear or close to linear time for any reasonable document length.

Problems and Challenges

The most notable challenge to the Cite Unseen system is the reliance on a third-party search provider. Commercial search providers are in business to provide a search service to individual consumers. While many do provide programmatic access to their services through APIs, these APIs are not intended for the level of intense usage required by Cite Useen. These limitations created challenges both in the testing and the development of Cite Unseen.

Search Allowance and Restrictions

Development and testing of the project was stalled several times as a result of conflict with search providers. In general, the type of searching done by Cite Unseen is seen as malicious and extremely unusual by providers, which often resulted in search privilege suspension or throttling. However, in the cases of both Yahoo! and Google, allowances were made for continued development and thanks are owed to both organizations for their support.

Nonetheless, it is very clear that for a system like Cite Unseen to be successful, some long term arrangement would need to be made with a search provider, or an entirely private system would need to be developed.

Access to Indexed Materials

Most search providers offer full access to all indexed material to any user searching through normal means. However, this is not the case when interfacing through a search provider API. Instead, search providers segment each index into a separate service with a separate API. For instance, Google provides access to Web Search and

Books Search through entirely separate APIs. Some materials cannot be accessed via API at all. This compartmentalization creates difficulty in accessing all material easily and impacts the ability of the Cite Unseen system to identify all types of copied text. Cite Unseen is an entirely reasonable plagiarism detection solution provided the system has adequate access to an appropriate index.

Future Research and Development

There are several areas the Cite Unseen project could be expanded in the future. Further testing on score manipulation is warranted, based on the limited usefulness of frequency matching in the current implementation. In addition, the user interface could be further developed to provide better and more user friendly options to assist in locating true plagiarism from false positives.

Scoring and Analysis

The limited usefulness of frequency matching was a surprising result that deserves further investigation. Some additional areas that could be explored include overall sequence weighting by entropy, semantic analysis to identify similar words, and determination of common *n*-gram terms for various values of n in order to minimize search and processing time.

User Interface Improvements

The current Cite Unseen user interface as discussed in Chapter 3 is a fairly basic example and could be greatly improved. In particular, users should have the ability to hide certain matching sequences and unmask overlapped or hidden sequences. Altering displayed sequences should update the total reported similarity accordingly. Further

improvement in citation filtering is also necessary to avoid creating false positives for legitimately cited sources. Scoring settings likely should also be removed, using optimal settings internally in order to simplify the user process.

Streamlining the user process is critical to ensuring the system is a success. Although processing and analysis is critical to any plagiarism detection system, enabling the user to easily identify true plagiarism is the hallmark of a useful plagiarism detection tool. While plagiarism can be identified with great success programmatically, it ultimately requires a human eye to make any final determinations.

References

Apache HttpComponents. (2014). *HttpComponents HttpClient Overview.* Retrieved 8 March, 2014, from http://hc.apache.org/httpcomponents-client-4.3.x/index.html

Apache Lucene. (2014). *Apache Lucene.* Retrieved March 8, 2014, from https://lucene.apache.org/

Apache Tika. (2014). *Apache Tika.* Retrieved March 8, 2014, from https://tika.apache.org/

Barrón-Cedeño, A., & Rosso, P. (2009). On Automatic Plagiarism Detection Based on n-Grams Comparison *Advances in Information Retrieval, 31st European Conference on IR Research, ECIR 2009, Toulouse, France, April 6-9, 2009. Proceedings* (pp. 696-700): Springer Berlin Heidelberg.

Bentley, J. (1984). Programming pearls: algorithm design techniques. *Communications of the ACM, 27*(9), 865-873. doi: 10.1145/358234.381162

Bourdaillet, J., & Ganascia, J.-G. (2006). MEDITE: A Unilingual Textual Aligner. In T. Salakoski, F. Ginter, S. Pyysalo & T. Pahikkala (Eds.), *Advances in Natural Language Processing* (Vol. 4139, pp. 458-469): Springer Berlin Heidelberg.

Broder, A., Fontura, M., Josifovski, V., Kumar, R., Motwani, R., Nabar, S., . . . Xu, Y. (2006). Estimating corpus size via queries. *Proceedings of the 15th ACM international conference on Information and knowledge management*, 594-603. doi: 10.1145/1183614.1183699

Ceska, Z., & Fox, C. (2009). The Influence of Text Pre-processing on Plagiarism Detection. *In Proceedings of the International Conference on Recent Advances in Natural Language Processing 2009*, 55-59.

Clough, P., & Stevenson, M. (2011). Developing A Corpus of Plagiarised Short Answers. *Language Resources and Evaluation: Special Issue on Plagiarism and Authorship Analysis, 45*(1), 5-24.

Francis, W. N., & Kucera, H. (1979). Brown Corpus Manual. *International Computer Archive of Modern and Medieval English.* Retrieved June 17, 2013, from http://icame.uib.no/brown/bcm.html

Gipp, B., & Beel, J. (2010). Citation Based Plagiarism Detection - A New Approach to Identify Plagiarized Work Language Independently. *In Proceedings of the 21th ACM Conference on Hyptertext and Hypermedia*, 273-274.

Google Code. (2014). *A Java library to convert JSON to Java objects and vice-versa.* Retrieved March 8, 2014, from https://code.google.com/p/google-gson/

Grozea, C., Ghel, C., & Popescu, M. (2009). ENCOPLOT: Pairwise Sequence Matching in Linear Time Applied to Plagiarism Detection. *In Proceedings of 3rd PAN workshop. Uncovering Plagiarism, Authorship and Social Software Misuse*, 10-18.

Grozea, C., & Popescu, M. (2010). Encoplot – Performance in the Second International Plagiarism Detection Challenge - Lab Report for PAN at CLEF 2010. In M. Braschler, D. Harman & E. Pianta (Eds.), *CLEF (Notebook Papers/LABs/Workshops)*.

Gusfield, D. (1997). Core String Edits, Alignments, and Dynamic Programming *Algorithms on Strings, Trees and Sequences* (pp. 215-253): Cambridge University Press.

Ha, L. Q., Sicilia-Garcia, E. I., Ming, J., & Smith, F. J. (2002). Extension of Zipf's law to words and phrases. *Proceedings of the 19th international conference on Computational linguistics - Volume 1*, 1-6. doi: 10.3115/1072228.1072345

Habermann, G. (2009, August 5). Title Tag and Meta Description Length for Google, Yahoo, Bing & Ask. Retrieved from http://www.sagerock.com/blog/title-tag-meta-description-length/

Hearle, N. (2003). Statistics - Sentence and Word Length. *Nahoo.* Retrieved March 18, 2014, from http://hearle.nahoo.net/Academic/Maths/Sentence.html

Hölzle, U. (2009). Powering a Google search. Retrieved June 15, 2013, from http://googleblog.blogspot.com/2009/01/powering-google-search.html

How big is your index? (2014). *FAROO Peer-to-Peer Web Search.* Retrieved March 16, 2014, from http://www.faroo.com/hp/p2p/faq.html#size

Lyon, C. (Producer). (2004). The Ferret Copy Detector. [PowerPoint Presentation] Retrieved from http://homepages.stca.herts.ac.uk/~comrcml/NCAF-Ferret.ppt

Lyon, C., Barrett, R., & Malcolm, J. (2006). Plagiarism is Easy, but also Easy To Detect. *Plagiary: Cross-Disciplinary Studies in Plagiarism, Fabrication, and Falsification, 1*, 57-65.

Malcolm, J., & Lane, P. (2008). An approach to detecting article spinning. *In Proceedings of the Third International Conference on Plagiarism*, 1-9.

Manning, C. D., Raghavan, P., & Schütze, H. (2009). Scoring, term weighting & the vector space model *Introduction to Information Retrieval* (Online ed., pp. 109-132): Cambridge University Press.

Meyer zu Eissen, S., & Stein, B. (2006). Intrinsic Plagiarism Detection. *Advances in Information Retrieval, 28th European Conference on IR Research, ECIR 2006*, 565-569.

Needleman, S. B., & Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology, 48*(3), 443-453.

Polyanovsky, V. O., Roytberg, M. A., & Tumanyan, V. G. (2011). Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences. *Algorithms for Molecular Biology, 6*(25). doi: 10.1186/1748-7188-6-25

Potthast, M., Hagen, M., Gollub, T., Tippmann, M., Kiesel, J., Rosso, P., . . . Stein, B. (2013). Overview of the 5th International Competition on Plagiarism Detection. *CLEF 2013 Evaluation Labs and Workshop – Working Notes Papers*.

Potthast, M., Stein, B., Eiselt, A., Barrón-Cedeño, A., & Rosso, P. (2009). Overview of the 1st International Competition on Plagiarism Detection. *In Proceedings of 3rd PAN workshop. Uncovering Plagiarism, Authorship and Social Software Misuse*, 1-9.

Ramos, J. (2003). *Using TF-IDF to Determine Word Relevance in Document Queries.* Paper presented at the First instructional Conference on Machine Learning (iCML-2003), Piscataway, NJ.

Smith, T. F., & Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology, 147*(1), 195-197.

Stein, B., Lipka, N., & Prettenhofer, P. (2011). Intrinsic plagiarism analysis. *Language Resources and Evaluation, 45*(1), 63-82. doi: 10.1007/s10579-010-9115-y

UAX #29: Unicode Text Segmentation. (2013, September 20). *The Unicode Consortium.* from http://unicode.org/reports/tr29/

Xia, X. (2007). Sequence Alignment *Bioinformatics and the Cell: Modern Computational Approaches in Genomics, Proteomics and Transcriptomics* (pp. 24-30): Springer Verlag.