CA Journal of Cultural Analytics

A BLAST-based, Language-agnostic Text Reuse Algorithm with a MARKUS Implementation and Sequence Alignment Optimized for Large Chinese Corpora

Paul Vierthaler and Mees Gelein

03.18.19

Peer-Reviewed By: Jeffrey Tharsen

Clusters: Data

Article DOI: 10.22148/16.034

Dataverse DOI: 10.7910/DVN/2YYJ2B

Journal ISSN: 2371-4549

Cite: Paul Vierthaler and Mees Gelein, "A BLAST-based, Language-agnostic Text Reuse Algorithm with a MARKUS Implementation and Sequence Alignment Optimized for Large Chinese Corpora," Journal of Cultural Analytics. March 18, 2019.

Until relatively recently, following the life of a phrase or passage from its origin as it is quoted, reused, and remixed through a large corpus of Chinese writing was extremely difficult.[1] Yet identifying how and when a document is appropriating materials from earlier works is critical in many domains. Establishing the source of a given sequence of words is not only important in industry (particularly in

[1] The suite of tools described in this article can be found at https://github.com/vierth/chinesetextreuse and Mees Gelein's MARKUS implementation is at https://dh.chinese-empires.eu/markus/. I wrote all the scripts using libraries found in the Anaconda 5 distribution of Python 3.6.3 (www.anaconda.com). Additionally, I used the python Levenshtein library for quick measurements of string similarity: David Necas, Antti Haapala, et. al. *Python Levenshtein*, https://pypi.org/project/python-Levenshtein/, accessed June 20, 2018. Visualizations were made using D3JS (Mike Bostock, *D3js*, https://github.com/d3/d3/wiki, accessed June 20, 2018) and Adobe Illustrator. More information can be found in the GitHub repository.

patent law), journalism, and academia (to detect and prevent plagiarism), but it is also valuable as an interpretive mechanism for those who study literary or historical documents. Developing consistent ways of tracing information movement through a corpus of documents helps scholars understand information networks, intellectual trends, and quotation practices.

In the case of Chinese studies, scholars have long been interested in identifying the sources of appropriated text, as recycling textual material without clear signposting was not unusual in imperial Chinese literature. The adaptation and reuse of older materials was a key stylistic choice made by many authors who assumed that readers would understand the connection to an earlier set of materials. A very famous example is the novel *Plum in the Golden Vase*, which borrows materials from a vast array of earlier novels, poetry, drama, and historical works without ever explicitly citing its sources.[2] In the specific cultural context in which a document was originally written, this works. Yet modern readers will often miss these connections. Additionally, once philologists have identified instances of obvious reuse (a new edition of a popular novel) and obscure quotations (an unattributed line of 400 year-old poetry in the middle of a short story), it is then necessary to study the minute variations in how the text transforms across time as authors moved it from one document to another.

Scholars have long conducted painstaking research to identify source materials and track variations within the quotes themselves with impressive results. Yet the arduous nature of this work, and increased access to comprehensive open-source textual corpora such as the *Chinese Text Project* and the *Kanseki Repository*, has fueled efforts to approach this problem algorithmically. Simple search and find operations open many possibilities, but because text sequences can undergo many transformations as they are shared between documents, a more flexible approach is necessary. Searching for a direct quote misses many potential instances of reuse where authors or scribes introduced even minor changes. Similarly, this approach lacks exploratory capability: it depends on the scholar having a priori knowledge of textual appropriation and will not unearth previously unknown instances of quotation. As such, it very useful to develop exploratory algorithms that find similar sequences of characters.[3]

Developing computational methods to identify text reuse is a popular issue, and computer scientists and digital humanists have developed many algorithms and

[2] For a detailed rundown of its many sources, see Patrick Hanaan, "Sources of the Ch'in Ping Mei," *Asia Major* 10, no. 1 (1963): 23-67.

[3] The algorithms I will be describing here do not speak to the directionality of quotation (if text *A* quotes text *B*, if *B* quotes *A*, or if there is an unknown third text being quoted). It is up to the scholar to determine likely directionality using outside information.

tools to facilitate this practice in their own research. For example, David Smith, Ryan Cordell, and Abigail Mullen developed a method for detecting text reuse by identifying newspaper articles which contain a high incidence of shared sequences of words (*n*-grams) and then running a local sequence alignment algorithm on the resulting documents.[4]Marco Buchler of the Electronic Text Reuse Acquisition Project at Gottingen University has also developed a highly sophisticated tool, TRACER, to detect both direct textual reuse and recycled ideas.[5] In Chinese Studies, scholars like Donald Sturgeon and Jeff Tharsen have also been developing tools. Sturgeon has developed two distinct approaches; a highly accurate algorithm that depends on domain knowledge of the corpus at hand,[6]and a more flexible but less accurate *n*-gram shingling approach now deployed in the text tools section of the *Chinese Text Project*. This tool allows users to specify a number of characters (*n*), and then simply highlights cases where a *n*-grams of the set length appears in the compared documents.[7] Tharsen, meanwhile, has been developing *Intertext* at the University of Chicago, which identifies intertextuality in up to five documents.[8]

While all of the methods mentioned above have their strengths, they have proven inadequate for my own use-case: identifying text reuse in a large collection of long prose Chinese works from the 15th to the 19th centuries. The Smith, *et. al.* algorithm performs remarkably well when looking at shorter documents such as newspaper articles, but the Smith-Waterman local alignment algorithm they use to test for intertextuality is prohibitively slow when dealing with novel-length (or even chapter-length) textual objects. TRACER, while sophisticated and language agnostic, does not easily handle the hundreds of thousands of pairwise document comparisons necessary to compare all documents in my research corpus against each other.[9] Sturgeon's *n*-gram shingling approach, and Tharsen's comparison algorithm in Intertext, would perform well at this task, but they are only currently

[4]David Smith, Ryan Cordell, and Abigail Mullen, "Computational methods for uncovering reprinted texts in antebellum newspapers," *American Literary History* 27, no. 3 (2015).

[5]Marco Buchler, *TRACER*, https://www.etrap.eu/research/tracer/, accessed June 7, 2018.

[6]As Sturgeon works with an early corpus, he is able to take a few normalization steps, such as removing character radicals and accounting for variant character forms, that are difficult to perform when working with late imperial materials. Donald Sturgeon, "Unsupervised identification of text reuse in early Chinese literature," *Digital Scholarship in the Humanities* (November 2017), https://doi.org/10.1093/llc/fqx024.

[7]Sturgeon presented the *n*-gram shingling model at the DHAsia Summit in 2018. Donald Sturgeon, "Citation Practice in Pre-modern China," *DHAsia Summit Talk*, April 28, 2018.

[8]Jeff Tharsen, *Intertext*, http://edoc.uchicago.edu/textccr/textconcordancer.php, accessed June 20, 2018. He has also been working on a distributed version of this running on a high performance cluster.

[9]This is the case both in terms of RAM and in terms of processing time. TRACER also operates on the sentence level, and the documents I am working with do not have clear sentence boundaries, so each document is a "sentence."

designed to work on a few texts at a time (and *n*-gram shingling would be a bit noisy for my purposes). Sturgeon's semi-supervised algorithm requires extensive tuning that is very time consuming. Furthermore, as is sometimes the case for research conducted by computer scientists, digital humanities scholars, and developers working in industry, the code researchers develop is always not readily available or open-source.

In this article, I present a methodology (and share the code that implements it) that provides fast intertextuality extraction from long documents at the corpus level and aligns the extracted quotes. I also discuss the MARKUS platform's recent introduction of a version of this intertextuality algorithm.[10] I approach the problem from a slightly different angle than the scholars presented above, though one with significant homologies to Smith, *et. al.*'s approach, optimized to trace text reuse and align instances of reused text as a reading aid for scholars who are interested in both macro-level distant reading and micro-level philological analysis in corpora containing documents of highly variable length. To do this, I have relied on innovations developed in bioinformatics. I first identify sequences with high levels of similarity using an approach based on the algorithm used by the Basic Local Alignment Search tool (BLAST), a DNA sequence alignment tool.[11] I then align the results using the Needleman-Wunsch global sequence alignment algorithm (described in detail later). The MARKUS implementation provides scholars a quick and easy method to find cases where two documents share text. Soon it will also allow comparisons between multiple documents.

A Scalable Text Reuse Algorithm

Some of the earliest and best developments in the field of similar sequence identification and alignment has been done in bioinformatics, and this work is directly applicable to identifying text reuse in human-produced documents. Scientists developed BLAST to compare nucleotide sequences and identify regions of high homology in DNA. Most BLAST implementations run on FASTA files, which represent the long strings of Gs, Ts, Cs, and As in DNA as text. While BLAST operates on a "language" with a vocabulary of four, it is easy to generalize the concept to a language with an arbitrarily large vocabulary by expanding the four-

[10] Mees Gelein has implemented this, with guidance from myself (and based on conversations with Jeff Tharsen and Brent Ho).

[11] Stephen F Altschul, et. al., "Basic Local Alignment Search Tool" *Journal of Molecular Biology* 215 (October 1990): 403-10.

character set to a dynamically generated set based on the contents of the corpus.[12] The algorithm itself, as I've implemented it, is also completely language agnostic. It immediately works at the character level in many languages (English, French, Japanese, etc.) by breaking input texts into character tokens.[13] With a language-specific tokenizer, it can be customized to work at the word level as well.

This algorithm operates on a few set parameters that specify what constitutes meaningful homology; a minimum matching length, and a minimum similarity as calculated by Levenshtein edit distance.[14]Appropriate values for these thresholds depend on the application in question, but in late imperial Chinese prose, repeated sequences shorter than eight characters tend to constitute mostly set phrases and idioms, number sequences, and other information that is not analytically useful. A high similarity threshold is best for working with prose documents, and I have had success with 80 to 85 percent similarity.[15] Those who work with poetry may find lower similarities more useful.

Once the thresholds are set, the algorithm itself is simple:

1. Break the two texts to be compared into "query words,"[16] or "seeds," of overlapping *n*-grams. *N* itself is arbitrary; four-character seeds seem to work well for Chinese.[17]
2. Record where every seed occurs in both texts in an index.
3. Find seeds that occur in both texts.[18]
4. Go to the seed location in both texts.
5. Expand the sequences one character at a time and measure their similarity.
6. Once similarity falls below the set threshold, return the match if it is above the set length.

[12]BLAST has an advantage in that there can be partial matches, where Gs are more structurally similar to As than to Ts, so the algorithm accounts for this. In the case of Chinese, this is also true, but creating these relationships involves significant overhead (Sturgeon's approach does take such relationships into account, though not for the purposes of a scoring algorithm).

[13]Languages written from right to left like Arabic and Hebrew may require some adjustment.

[14]Vladimir Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals," *Soviet Physics-Doklady* 10, no 8. (1966): 707-710. For fast calculation, I use the python-Levenshtein library, which wraps fast C code.

[15]I consider an edit distance of two for a ten-character phrase to be 80 percent similar ((1 - edit distance)/length).

[16]As they were called in Altschul, 404.

[17]This is a balance between speed and accuracy. A seed of one would ensure all matches are found, but this is computationally expensive. A seed of eight would execute quickly, but would miss any shared sequence that did not start with eight identical characters. Longer seeds work better for English and other alphabetic languages.

[18]Indexing the seeds and then finding the intersection of the two seed lists represents a significant speed increase when compared to searching for the seeds from one text in the other text.

This algorithm returns all sequences that meet the set criteria, but instead of returning the raw results as soon as the sequences fall below the threshold, I back the match up to the last point similarity increased (or remained at 100 percent). To ensure that the algorithm doesn't return duplicate information, I ignore seeds that have already been mapped to each other as part of a quote. I also implement several small speed optimizations. For example, when I am using four-character seeds and a minimum length of ten, I do not calculate the interim similarity of the five to nine-character sequences, but immediately look at the ten-character sequence starting at the seed index. If the sequence is above the similarity threshold I keep running the algorithm. Otherwise, I move to the next matching seed. When matching sequences are over one hundred characters long, I also only use the last one hundred characters to calculate the similarity metric.[19]

It is necessary to do some post-search curation to remove uninteresting results and one can approach this either automatically or in the context of domain-specific knowledge. For example, the vast majority of matches that are returned when comparing Ming and Qing dynasty novels with each other aresome variant of "to see/hear what happens next, see the next chapter." For example, each chapter of *Plum in the Golden Vase* ends with something like "If you want to know what happens next, please read the next chapter (畢竟未知後來何如且聽下回分解)." This is further complicated by the fact that in many digital editions, each chapter begins with "Chapter Number X (第 X 回)." If the algorithm is working on the full text (rather than individual chapters), then just one occurrence of this might be returned 100 times as a 16-19 character long 80+ percent match when compared against another 100-chapter novel that uses similar phrasing, as differing chapter numbers are often not enough to push the match below the similarity threshold.

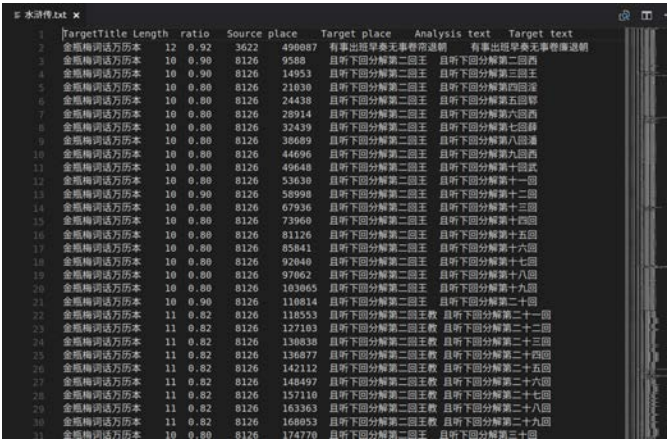
This algorithm can produce significant numbers of matches that need to be filtered out if one is not interested in such formulaic phrases. While I am particularly interested in what these phrases reveal about the structure of Chinese genres, I remove these when exploring questions about information transmission.[20] There are multiple possible ways to filter the results, but the process I use is very simple: in cases where a short phrase is detected more than a certain number of times, I remove it from the results (along with all phrases with a set similar-

[19] This has the dual benefit of increasing speed, as the distance metric can be time consuming to calculate, and ensuring that the returned quotes do not contain significant mismatch at the end when they are extremely long.

[20] As I explain below, I remove phrases that occur more than forty times from the results (as well as phrases that are highly similar to the removed quotes), for example. Forty works well for comparing two novels pairwise, but for comparing hundreds of texts, the minimum threshold may need to be set much higher. The compile and filter results script does this automatically.

ity). I do this fully automatically, but it would be simple to return representative examples of the frequent quotes so a scholar could curate the deletion.[21]

This algorithm very quickly identifies similar sequences of characters. On a relatively powerful consumer computer with an Intel Core i7-6700 manufactured in late 2015/2016, it takes around 2 seconds to identify all matches between the two novels *Water Margin* and *Plum in the Golden Vase*, each of which are approximately 700,000 characters long. Most of the processing time is spent in creating the indexes. Before any filtering, the algorithm identifies 10,275 sequences of text that are at least ten characters long and eighty percent similar (the average length of these sequences is 12.7 characters and average similarity is 83.9 percent).[22]Figure 1 shows an example of the raw text output from the algorithm:



	TargetTitle	Length	ratio	Source place	Target place	Analysis text	Target text
2	金瓶梅词话万历本	12	0.92	3622	490087	有事出班早奏无事卷帘退朝	有事出班早奏无事卷廉退朝
3	金瓶梅词话万历本	10	0.90	8126	9588	且听下回分解第二回王	且听下回分解第二回西
4	金瓶梅词话万历本	10	0.90	8126	14953	且听下回分解第二回王	且听下回分解第三回王
5	金瓶梅词话万历本	10	0.80	8126	21030	且听下回分解第二回王	且听下回分解第四回淫
6	金瓶梅词话万历本	10	0.80	8126	24438	且听下回分解第二回王	且听下回分解第五回郓
7	金瓶梅词话万历本	10	0.80	8126	28914	且听下回分解第二回王	且听下回分解第六回西
8	金瓶梅词话万历本	10	0.80	8126	32439	且听下回分解第二回王	且听下回分解第七回薛
9	金瓶梅词话万历本	10	0.80	8126	38689	且听下回分解第二回王	且听下回分解第八回潘
10	金瓶梅词话万历本	10	0.80	8126	44696	且听下回分解第二回王	且听下回分解第九回西
11	金瓶梅词话万历本	10	0.80	8126	49648	且听下回分解第二回王	且听下回分解第十回武
12	金瓶梅词话万历本	10	0.80	8126	53630	且听下回分解第二回王	且听下回分解第十一回
13	金瓶梅词话万历本	10	0.90	8126	58998	且听下回分解第二回王	且听下回分解第十二回
14	金瓶梅词话万历本	10	0.80	8126	67936	且听下回分解第二回王	且听下回分解第十三回
15	金瓶梅词话万历本	10	0.80	8126	73960	且听下回分解第二回王	且听下回分解第十四回
16	金瓶梅词话万历本	10	0.80	8126	81126	且听下回分解第二回王	且听下回分解第十五回
17	金瓶梅词话万历本	10	0.80	8126	85841	且听下回分解第二回王	且听下回分解第十六回
18	金瓶梅词话万历本	10	0.80	8126	92040	且听下回分解第二回王	且听下回分解第十七回
19	金瓶梅词话万历本	10	0.80	8126	97062	且听下回分解第二回王	且听下回分解第十八回
20	金瓶梅词话万历本	10	0.80	8126	103065	且听下回分解第二回王	且听下回分解第十九回
21	金瓶梅词话万历本	10	0.90	8126	110814	且听下回分解第二回王	且听下回分解第二十回
22	金瓶梅词话万历本	11	0.82	8126	118553	且听下回分解第二回王教	且听下回分解第二十一回
23	金瓶梅词话万历本	11	0.82	8126	127103	且听下回分解第二回王教	且听下回分解第二十二回
24	金瓶梅词话万历本	11	0.82	8126	130838	且听下回分解第二回王教	且听下回分解第二十三回
25	金瓶梅词话万历本	11	0.82	8126	136877	且听下回分解第二回王教	且听下回分解第二十四回
26	金瓶梅词话万历本	11	0.82	8126	142112	且听下回分解第二回王教	且听下回分解第二十五回
27	金瓶梅词话万历本	11	0.82	8126	148497	且听下回分解第二回王教	且听下回分解第二十六回
28	金瓶梅词话万历本	11	0.82	8126	157110	且听下回分解第二回王教	且听下回分解第二十七回
29	金瓶梅词话万历本	11	0.82	8126	163363	且听下回分解第二回王教	且听下回分解第二十八回
30	金瓶梅词话万历本	11	0.82	8126	168053	且听下回分解第二回王教	且听下回分解第二十九回
31	金瓶梅词话万历本	10	0.80	8126	174770	且听下回分解第二回王	且听下回分解第三十回

Figure 1. Unprocessed output file as produced by the intertextuality algorithm. Here we can see the unfiltered results of detected shared text between the Plum in the Golden Vase (listed as TargetTitle) and the Water Margin (the name of the file). There is significant repetition and these results mostly represent noise. Produced by detect_intertexuality.py

After filtering any sequence that is forty or fewer characters and is detected at least forty times (and phrases that are at least 60 percent similar to those), 895 match-

[21]I initially carefully controlled the deletion, but after working with the results for a while, I rarely found any quotes were being deleted that I wished to maintain in the analysis.

[22]These figures were derived by comparing the *cihua* edition of *Plum in the Golden Vase* found at the now defunct *Daizhige* website againstthe 120 chapter edition of the *Water Margin* from WikiSource. You can find a copy of the *Plum* edition in question in the DataVerse repository at http://dx.doi.org/10.7910/DVN/4ZVSKA in the Chinese corpus under the filename 金瓶梅词话万历本 _ 兰陵笑笑生 _ 明 _ 小说.txt. The *Water Margine* can be found at https://zh.wikisource.org/wiki/%E6%B0%B4%E6%BB%B8%E5%82%B3_(120%E5%9B%9E%E6%9C%AC)/%E7%AC%AC120%E5%9B%9E.

ing sequences remain (averaging 31 characters and 86 percent similar). Figure 2 shows the significantly more diverse nature of the filtered output:

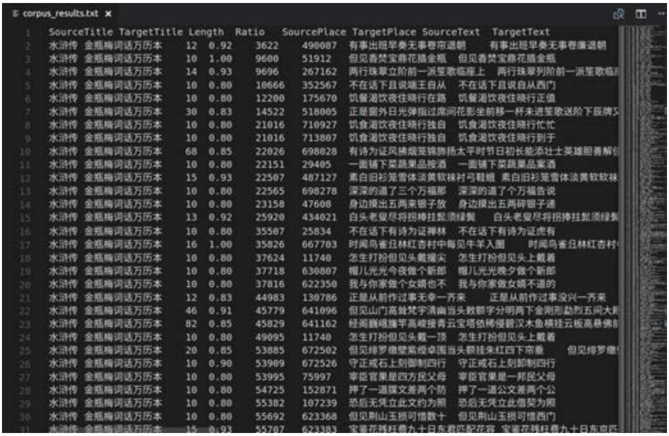


Figure 2. Filtered results file as output by the intertextuality algorithm. Now all the results reside in a single file where the SourceText (the Water Margin) and the TargetText (the Plum in the Golden Vase) are both represented. The results are significantly denser and exhibit much less repetition. Produced by compile_and_filter_results.py

Corpus-level text reuse

Document-level comparisons using this algorithm are relatively quick, but processing slows considerably when doing pairwise comparisons between all documents in even a semi-large corpus. Assuming each comparison takes an average of two seconds, comparing 1,000 documents against each other would take around 280 hours if done naively.[23] However, I can rely both on the structure of dataset, as well as the knowledge that creating the indices is where most of the overhead is, to significantly speed this process up. The single most important thing is pre-calculating an index for each text, as this is the primary bottleneck. To do so, I create a unique identification number for every seed in the corpus and save only index values for seeds that appear in at least two documents.[24] Addi-

[23] This involves 500,500 comparisons.

[24] The indexing method I use is relatively disk and memory intensive, but it is optimized for the highest speed possible given the amount of RAM in my machine (64GB of RAM aimed at processing corpora of around 150 million characters/1000 documents). For a description of several more

tionally, I can simultaneously make as many comparisons as my computer has threads. This means instead of doing one comparison at a time, I can do as many as eight (given that my CPU has four cores). After all the optimizations, I can reduce the comparison time to around 12 seconds to compare the *Plum in the Golden Vase* against 974 other documents. To do pairwise comparisons among all documents in a 975-document, 157 million-character corpus, it takes 17 minutes.[25]

This exhaustive level of comparison produces a wealth of data that can be used to generate insights into corpus-level repetition. It also identifies text similarity in places that had not been noticed in the past. The algorithm outputs a file which contains information on all of the matches: the documents in which the matches appear, where they appear, their length, similarity, and the sequences themselves. Simply combing through the intertextuality results of two documents that do not have significant intertextuality is relatively straightforward and can be done without the aid of visualization tools, but dealing with thousands of documents is more difficult because of the inevitable size of the results. This means a heuristic approach must be taken in order to explore the results. This can be done in a variety of ways. First, one can simply look at the longest results, which will be significantly rarer. It is unusual to find highly similar sequences of more than 200 characters in a row (roughly a full page of copied material). Filtering out repetitive phrases also helps to winnow down on the amount of results that one needs to wade through. Alternatively, one can turn to network visualizations to rapidly understand the connections among documents.[26]

In Figures 3a and 3b below, you can see the network of results when comparing a corpus of late imperial prose documents written in the Ming and Qing Dynasties in China. Each node represents a document and each edge represents some amount of shared text. I calculate the edge weight by factoring the length of identified sequences by their similarity score and then summing the adjusted scores (so a 20-character sequence that is 90 percent the same will contribute 18 points to the score). This approach lets me see a broad picture of which documents

efficient approaches, and the inspiration for my implementation, see Samuel Huston, Alistair Moffat, and W. Bruce Croft, "Efficient Indexing of Repeated *n*-Grams,"(2010). I have also taken cues from Smith, *et. al.*, who use a version of Huston's two-pass hash algorithm.

[25] This assumes a pre-indexed corpus. The indexing process takes around 90 seconds. The *Plum in the Golden Vase* is one of the longer documents in the corpus, so takes longer than the average document. On average it takes about 1 to 2 seconds to compare a document with the rest of the corpus. In an actual research context, I have used this tool to compare 3,400 chapters from 500 books in around 5 minutes.

[26] To facilitate this, I have provided code that transforms the returned results into a Gephi-compatible edge list. Donald Sturgeon employs a similar networked approach in "Unsupervised Identification of Text Reuse in early Chinese Literature."

are most closely connected with which other documents. Some documents have edges with a very high score and are usually different editions of the same work (as shown in Figure 1b), while others are only loosely connected by a single ten-character idiom.

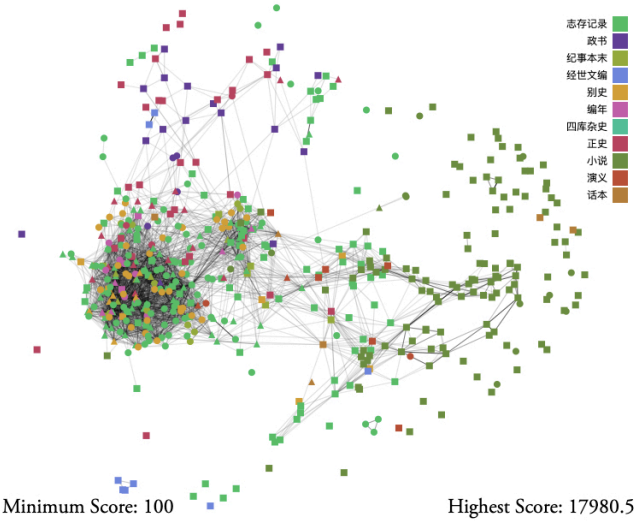


Figure 3a. Network of Intertextuality among 11 genres of late imperial Chinese Prose. Documents are chapters from works which mention either the late Ming eunuch Wei Zhongxian (squares), the late Ming general Shi Kefa (circles) or both (triangles). Edges are limited to connections with a score of at least 100. Screenshot from the dynamic visualization appearing at www.pvierth.com/pca/

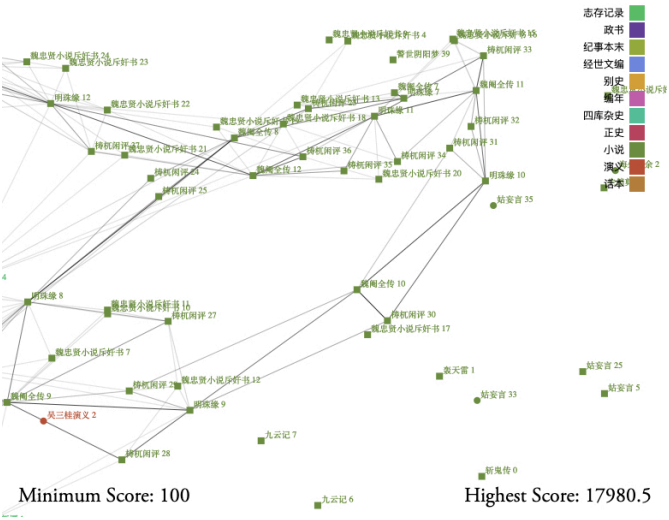


Figure 3b. Detail of network with text titles and chapter numbers displayed. The loop here is formed by different editions of the same novel.

This figure shows connections macro-scale connections, but the act of condensing quotes into a single edge significantly flattens the complex relationships between individual texts. The multiplicity of connections between two documents is often significantly more interesting than its aggregate similarity score would seem to indicate. To account for this, it is often useful to only look at several documents at a time with *all* of their edges visible. The easiest way to display all edges between documents depends on the number of documents one is working with. For two documents, a simple approach is to abstractly represent each text as a bar that is sized according to the relative length of the document. By then using lines to connect sections where intertextuality occurs, as in Figure 4a, we can apprehend the extent of sharing. When working with more than two documents, a circular chord diagram like the one provided by the D3JS javascript package is very useful.[27] Figure 4a, below, shows all 895 quotes shared between the *Plum in the Golden Vase* and the *Water Margin*.

[27] The idea for using a chord diagram comes from Mees Gelein's MARKUS's implementation of the intertextuality algorithm. The chord diagram itself was created using D3JS. While the alignment algorithm can be run on the entire corpus, it is easiest to run it only on documents of interest (ideally identified via network analysis) because aligning the millions of results the reuse algorithm may identify can take significant time.

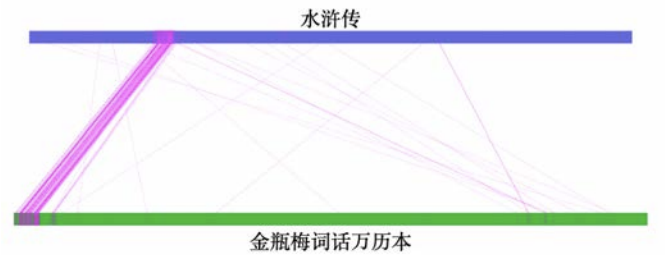


Figure 4a. Intertextuality between the Water Margin (top, purple)and the Plum in the Golden Vase (bottom, green). Most of the intertextuality comes at the beginning of the Plum in the Golden Vase, but it is scattered throughout the work.

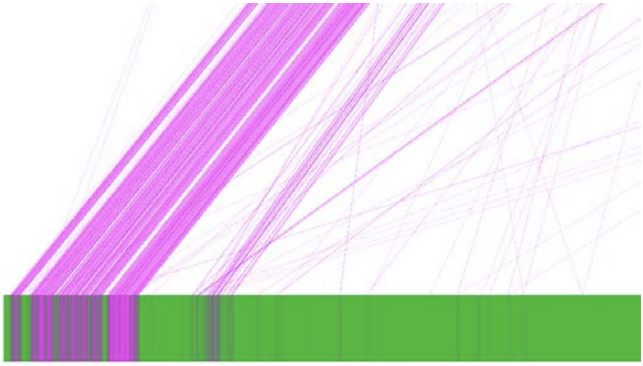


Figure 4b. Detail of intertextuality at the beginning of the Plum in the Golden Vase.

While the relationships are compressed into a single edge in a network visualization, here they can help a scholar rapidly understand the extent to which these two works are directly textually related. The first part of the *Plum* is copied extensively from a section in the middle of the *Water Margin*. The algorithm captures this intertextuality as hundreds of tightly packed quotes that are tens to thousands of characters long. In actuality, this is one long, edited instance of copying that extends across the first six chapters of the *Plum in the Golden Vase*. The *Plum*'s anonymous author transformed a story from the *Water Margin* and used it as the genesis of his own novel. Even though this sequence can be conceived of as a singular chunk of text, it is captured as intermittent quotations, an artifact of only measuring the similarity between the last 100 characters of each

sequence.[28] Beyond capturing this early extensive sharing, it also picks up upon the other fragmentary sharing that occurs throughout the book. While the complicated intertextual relationships between the *Plum* and the *Water Margin* are well understood, this algorithm ensures that a scholar can exhaustively identify every instance of sharing regardless of the works involved (within the limits of the set similarity thresholds).

In the chord diagram in Figure 5, you can see another example of how quoted materials connect multiple chapters from novels and historical documents about the Eunuch Wei Zhongxian written in the mid-seventeenth century. Scholars like Han Li have looked closely at the textual history of some of theseworks, the copying among them, and how this was used to create coherent (or not sometimes not so coherent) narratives, but the extent and distribution of textual similarity is readily evident and easy to track in the diagram.[29]

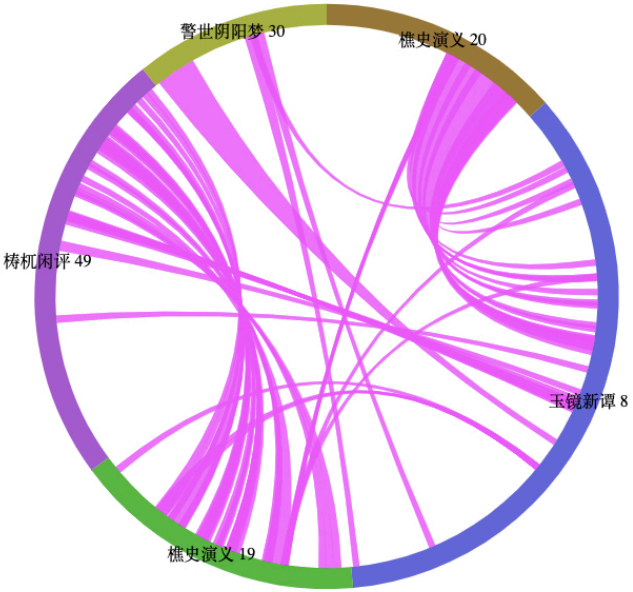


Figure 5. Intertextuality among five chapters from books about Wei Zhongxian.

[28] You can disable this behavior in the accompanying script, at the cost of significantly increased compute times and quotes with lots of noise at the end in the case of very extensive sharing.

[29] Han Li, "News, Public Opinions and History: Fiction on Current Events in Seventeenth-Century China" (Ph.D., University of California, Irvine, 2009).

Brown: Historical Romance of the Woodcutters Chapter 20, Blue: New Discussion of the Jade Mirror, Green: Woodcutters, chapter 19, Purple: Idle Tales of the Taowu Beast. Here, there are long stretches of text that are remixed across each document.

Clearly these five chapters share and remix significant textual information, and studying these patterns of sharing and editing helps us understand how information can be co-opted by different authors across multiple genres of text.

The Alignment Algorithm

Aligning the shared sequences viewable in Figures 4 and 5 above can help us make more effective use of this visualization approach for both large-scale analysisand detailed close reading by identifying precisely how and where the texts differ in their use of similar language sequences. There are a wide variety of sequence alignment algorithms, many of which were developed in bioinformatics, just like the BLAST algorithm, and they each suit different purposes. Smith, et. al. use the Smith-Waterman local alignment algorithm to compare their documents to identify and align local areas of similarity.[30] But because the BLAST-like algorithm returns sequences already optimized for similarity, I can use the Needleman-Wunsch global alignment algorithm to align them.[31]

Like Smith-Waterman, Needleman-Wunsch is a "dynamic programming" algorithm and involves creating a scoring matrix. Essentially, every possible alignment between the two sequences is scored, and the one with the highest score is returned. To do this, I give a positive score for two matching characters (let's say +1), a negative score when they are mismatched (let's say -1), and a negative score when I need to introduce a gap (-1). These scores can all be adjusted depending on what I want to prioritize. For example, I can discourage gaps by giving them a lower score (like -2 instead of -1). Figure 6 illustrates the process of creating the matrix. The matrix is formed by spreading out one sequence along the columns of a matrix, and the other is spread along the rows. The upper-left handcorner is seeded with a zero, and then the top row and first column are filled out by adding the gap score in each box (Figure 6a). I calculate scores for the rest of the matrix by looking at each box iteratively. First look at the score to the upper left of the

[30] T.F. Smith and M.S. Waterman, "Identification of Common Molecular Subsequences," *Journal of Molecular Biology* 147, no. 1 (1981): 195-197.

[31] Saul Needleman and Christian Wunsch. "A General Method Applicable to the Search for Similarities in the Amino Acid Sequence of Two Proteins." *Journal of Molecular Biology* 48, no. 3 (1970): 443-453.

box. If the characters represented by this box match, then I add the match score (+1) to whatever is in the upper left. If it is a mismatch, I add the mismatch score (-1) and remember this number. Then I check the score above and add the gap score (-1) and the score to the left and add the gap score (-1) (Figure 6b). Then I simply write down the highest score and move to the next empty box that has scored boxes both above and to the left (Figure 6c-6d). Once the entire matrix is filled out (Figure 6e), I start in the bottom right-hand corner of the matrix and follow the highest scores backwards (Figure 6f). Diagonal movement means the two characters represented here should be aligned with each other, if I move up or to the left, a gap should be inserted in either one or the other of the sequences. This returns an optimal global alignment between the sequences (though there may be more than one optimal alignment).

		H	E	L	L	L	O
	0	-1	-2	-3	-4	-5	-6
H	-1						
A	-2						
L	-3						
L	-4						
O	-5						

Figure 6a. Primed Matrix. To begin the scoring, characters in the first string are spread along the columns and the second are spread along the rows. We place a zero in the upper left hand corner and then add gap scores across the top row and leftmost column.

The matrix as filled in Figure 6a represents two clearly non-optimal alignments (in fact, these are the least optimal alignments):

HELLLO HELLLO

HALLO HALLO

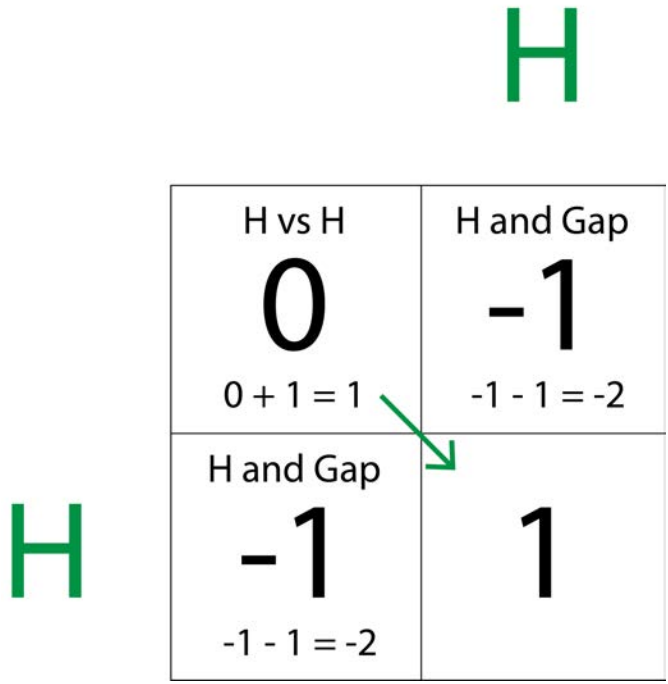


Figure 6b. The first score. Here we are determining the score of the box in the bottom right corner (asking if these two Hs should be aligned). Movement through the matrix determines how the alignment is formed. Diagonal movement through the matrix represents alignment between the two characters being considered. We check if H matches H and award the match a score (+1) so we take the score in the box on the upper left and add the match score (0 + 1) to get 1. It is also possible to add gaps in an alignment, and vertical and horizontal movement through the matrix represents this process. We add gap scores (-1) to the boxes above and to the left to get -2. We pick the maximum of these three scores (1) and place it in the box on the bottom right.

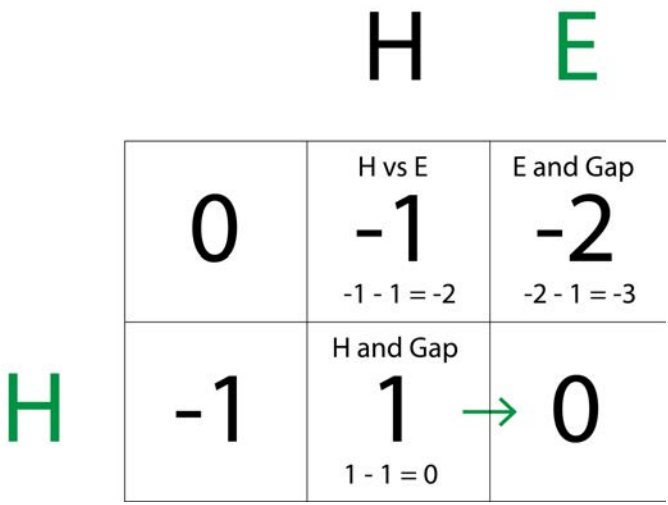


Figure 6c. The second score. We follow the same process for the next box in the matrix. Here, H does not match with E, so the diagonal score (-1 -1) is -2. The score from the top is even worse at -3 (-2 - 1). The score from the left, however, is zero (1 - 1), which is the highest score and so gets recorded.

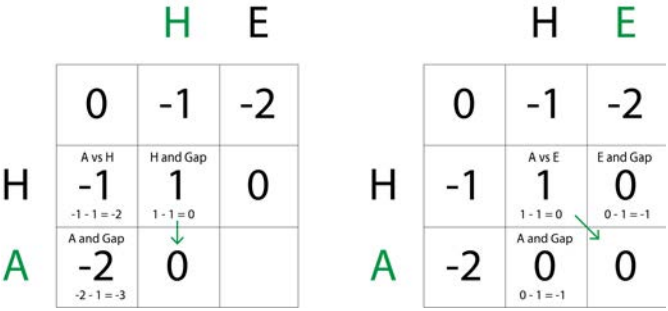


Figure 6d. The third and fourth scores. This process of filling out the matrix continues until all boxes within the matrix are complete.

		H	E	L	L	L	O
	0	-1	-2	-3	-4	-5	-6
H	-1	1	0	-1	-2	-3	-4
A	-2	0	0	-1	-2	-3	-4
L	-3	-1	-1	1	0	-1	-2
L	-4	-2	-2	0	2	1	0
O	-5	-3	-3	-1	1	1	2

Figure 6e. The completed matrix with all scores filled in.

		H	E	L	L	L	O
	0	-1	-2	-3	-4	-5	-6
H	-1	1	0	-1	-2	-3	-4
A	-2	0	0	-1	-2	-3	-4
L	-3	-1	-1	1	0	-1	-2
L	-4	-2	-2	0	2	1	0
O	-5	-3	-3	-1	1	1	2

Figure 6f. Returning the optimal alignment. Beginning in cyan box on the bottom right (as we are using the Needleman-Wunsch algorithm), we trace the highest scores (highlighted in magenta) back through the matrix until we arrive at the yellow box in the upper left-hand corner. For all diagonal moves, we align the characters at this intersection. All horizontal and vertical moves represent inserted gaps. Here, there are actually two optimal alignments, where one could either move from the cyan box to the 1 on the left and then diagonal to the 2, or diagonal to the 1 (in green) and then left to the 2.

The matrix in Figure 6f produces the following two equivalent optimal alignments:

```
HELLLO          HELLLO

HAL LO          HALL O
```

This process enables quick and detailed philological analysis in a manner difficult to achieve in the past. If I highlight every inserted, deleted, or mutated character, I can rapidly assess the differences between the two sequences and start to look for patterns in edits. A word of caution, however, as this algorithm will pick up on every minute difference between the two documents. This includes typos that were introduced in the digitization process, though this problem is ameliorated with very high-quality digital editions. Figure 7 shows the output of the alignment algorithm, in which mismatched ends have been trimmed and spaces representing the places in which gaps have been introduced to align each sequence.

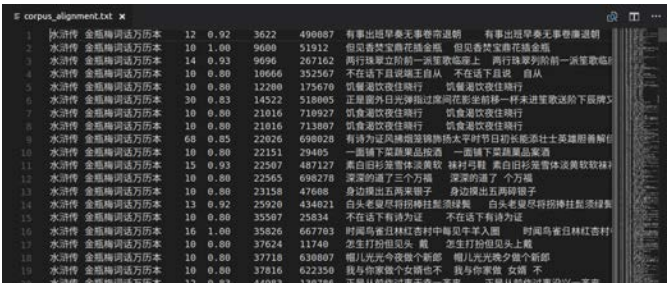


Figure 7. Aligned quotes shared between the Water Margin and the Plum in the Golden Vase.Note that the length and similarity scores have not been updated here. This is largely because by introducing spaces, these measures become less meaningful.

The results of the intertextuality algorithm combined with the sequence alignment algorithm can be presented in any number of ways, but they can be difficult to interpret when viewed as a flat file like in Figure 7 above. We can return to the intertextuality diagrams show in Figures 4 and 5 to introduce a new innovation that allows us to see the aligned quotes themselves by simply clicking on one of the edges. Using these diagrams as interfaces, one can retrieve the aligned and matching quotes by simply clicking on an edge. The quote is then displayed and each mismatched, inserted, or deleted character is highlighted for easy identification.[32]

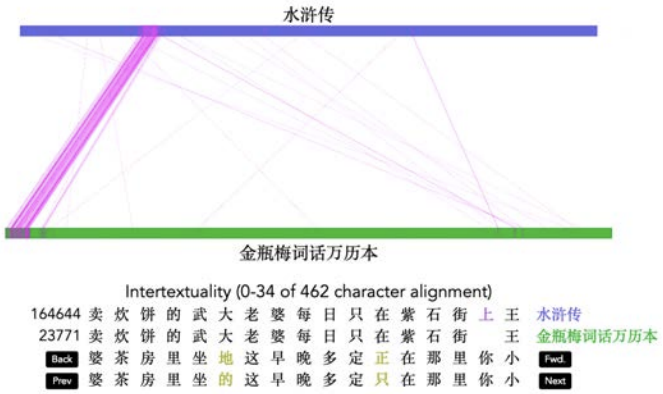


Figure 8. Aligned quote shared between the Water Margin in blue and the Plum in the Golden Vase in green. Inserted characters are highlighted in purple,and edited characters are highlighted in yellow.

Using an interface like the one shown in figure 8, we can rapidly see the editorial choices the author of the *Plum in the Golden Vase* made when adapting text from the *Water Margin*.In this case, there are several minor vocabulary changes (*zuodi* 坐地 becomes *zuode* 坐的, both essentially meaning "to sit," and *zhengzai* 正在 "continuously there" becomes *zhizai* 只在 "just/only there") and a prepositional character (*shang* 上 "tobe on") is deleted. Such tweaks are common throughout

[32]You can generate a similar figure yourself after aligning the documents you are interested by running the generate chord vis script and then opening the viz.html file. This basic GUI allows the user to explore connections between multiple documents and read the aligned quotes.

the shared sections of text. Like in this particular case, these changes do not always alter the fundamental meaning of the passage, but they can tell us a lot about the editorial practices that created the *cihua* edition of the *Plum in the Golden Vase*. This example also makes the need for good digital copies clear; many of these changes *could* be artifacts of the digitization process.[33]

A similar process exposes an instance of shared text between the unofficial history *Jade Mirror* and the historical romance the *Woodcutters* in the chord diagram shown in figure 9:

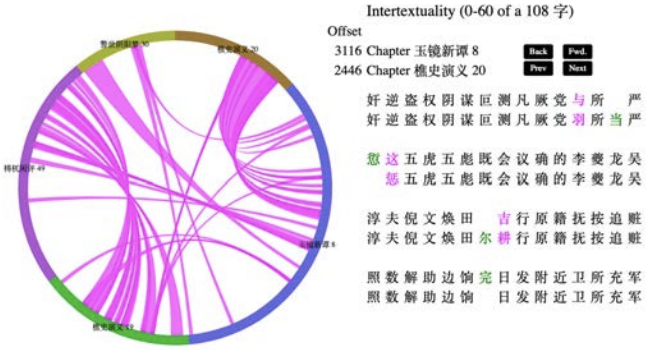


Figure 9. Sample visualization of all intertextuality existing between five different chapters from books about Wei Zhongxian. The displayed text is an instance of intertextuality between Chapter 8 of the Jade Mirrorand Chapter 20 of the Woodcutters).

It is possible to use the text reuse algorithm in conjunction with the sequence alignment algorithm to help conduct detailed philological research. The suite of algorithms are open and free to use, but do require some technical knowledge of Python to use. In light of this, I now turn to the MARKUS platform.

Comparativus: MARKUS and Intertextuality (implemented by Mees Gelein)

Recently the online semi-automated markup platform MARKUS implemented a version of intertextuality algorithm that I describe above but with a num-

[33] In this particular case the differences are not due to mistakes, but the grammatical relationship between *di* 地 and *de* 的 make it a prime candidate for a possible typo (made all the more likely by the fact that *de* is an alternate pronunciation for 地).

ber of tweaks designed to allow it to run efficiently client-side in a web browser.[34] Additionally, Mees has adjusted the results of the algorithm to fit the needs of MARKUS users, who are largely interested in using MARKUS as a markup platform for studying a few texts at a time. Gelein has written a full description of the algorithm as implemented in MARKUS, known as *Comparativus* and has made the code itself also open source and available here.[35]

The primary difference between how my corpus-level algorithm works and how *Comparativus* conducts its searches is that *Comparativus* does not implement a minimum matching length, and instead depends on the initial seed to limit results. This means the ideal seed in *Comparativus* is going to probably be longer than the four-characters I set my algorithm at. Additionally, *Comparativus* calculates an index for each text on the fly. This is perfectly reasonable when the number of objects being compared is relatively limited. As *Comparativus* is designed to run in web browsers, Gelein has made concerted efforts to avoid external dependencies, which has led to a few necessary compromises. For example, when determining seeds contained in both texts, *Comparativus* simply loops through the *n*-grams in each index and checks for equality (whereas in the Python implementation, I place the seeds in sets and check the intersection between the sets).

Comparativus seed expansion also operates slightly differently than my code. In my algorithm, I simply expand until I fall below the matching threshold. Gelein awards "strikes" for each time a score falls below 0.8 (the *Comparativus* similarity threshold) and expands to both the left and the right of the original seed (whereas by default, my script only expands to the right, given that the very small seed length I use means it is unlikely leftward expansion would markedly improve the result).[36] Additionally, rather than excluding locations from analysis during processing, Gelein merges duplicate results.

Comparativus is designed to quickly identify and visualize textual overlap in several documents at a time and many of the innovations Gelein has provided are related to dealing with noisy input data and allowing users to smoothly move between the intertextuality and the documents they are marking up within MARKUS. If one is already a user of MARKUS, it is very simple to use the tool. If you already have texts saved to your MARKUS profile, you can simply log

[34] Brent Ho and Hilde De Weerdt, *Markus. Text Analysis and Reading Platform,* http://dh.chinese-empires.eu/beta/, funded by the European Research Council Digging into Data Challenge, accessed June 20, 2018.

[35] Mees Gelein, "*Comparativus* Algorithm," https://github.com/MGelein/comparativus/blob/master/algorithm.md, accessed June 13, 2018. Mees Gelein, *Comparativus*, https://github.com/MGelein/comparativus, accessed June 13, 2018.

[36] At the very most, it would add three characters to the left.

in and then navigate to https://dh.chinese-empires.eu/comparativus/and then select two texts to compare.

Figure 10. Two "juan" from the Feng Menglong short story collection "Tales to Awaken the World" are loaded into the Comparativus module. By clicking the "Compare Texts" button, a user can quickly detect instances of text reuse.

Once the two texts have compared, *Comparativus* provides the user with a glimpse into the similarities between the two documents as shown in Figure 9:

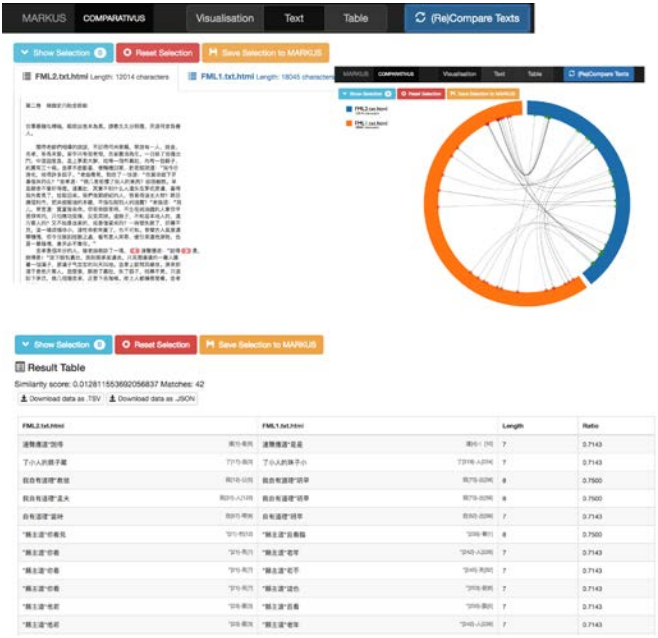


Figure 11. Intertextuality results. The results can be visualized in three ways: the

text itself with the shared words highlighted, a chord diagram view, and a table with all the results. The table can be exported to either tsv or json files.

Comparativus is still under active development and will soon be able to compare multiple texts at once and allow scholars to integrate intertextuality study into their already existing MARKUS workflow. For those who are interested in reuse detection at the corpus level, the code for the algorithms I developed accompany this article. As intertextuality algorithms improve, speed up, and become easier for scholars to use, we are bound to see a flourishing in intertextual studies. This will be increasingly true as high-quality corpora also become more readily available. My hope is the algorithm and code in this paper, *Comparativus*, and other tools being developed by people like Donald Sturgeon and Jeff Tharsen will help scholars more easily unmask the deep connections that bridge their sources.

Coda:

All of the code implementing the algorithms can be found at

http://www.github.com/vierth/chinesetextreuse

(which in spite of the name is indeed language agnostic). I am continuously developing the code to make it easier to use, faster, and more accurate. The following figure shows the workflow that allows you to use the code yourself by running the code associated with each arrow shown in the diagram. You simply need to start with the code and a corpus folder:

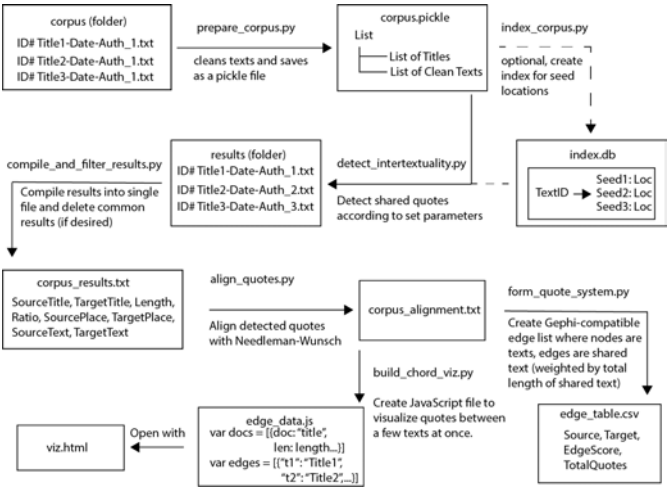


Figure 12. Workflow for using the Text Reuse algorithms.

I am finishing a version controlled from a single script (currently a development branch of the above GitHub repository), and after completing that I will develop a Cython version that should significantly speed the algorithms up.