
Authors:
J. Matthew Huculak
Ashlin Richardson

Contributors:
Adele Barclay
Daniel Carter
Tanya Clement
Alex Christie
James Gifford
Adam Hammond
Dean Irvine
Stephen Ross
Jentery Sayers
Susan Schreibman
Katie Tanigawa

Table of Contents

Introduction ..................................................................................................................... 3
Problem Statement .......................................................................................................... 4
Current State of Knowledge .......................................................................................... 5
Contextual Analysis ......................................................................................................... 7
New Modernism ............................................................................................................... 7
Methodology ................................................................................................................... 9
Summary of Testing: ........................................................................................................ 11
Recommendations ........................................................................................................... 12
Explicitness ...................................................................................................................... 12
Siloes ............................................................................................................................... 13
Collaboration .................................................................................................................. 13
Future Versioning ........................................................................................................... 13
Bibliography ................................................................................................................... 14
Annex I: Results .............................................................................................................. 18

Juxta-Commons & Juxta Desktop .................................................................................. 18
Screenshots (short sample data) .................................................................................. 20
Heat Map Visualization .................................................................................................. 20
User Annotation Interactive Visualization ....................................................................... 20
Side by Side Visualization ............................................................................................. 21
Histogram Visualization ................................................................................................. 21
Introduction

This white paper was produced by Dr. J. Matthew Huculak, a Postdoctoral Fellow with the Modernist Versions Project, and Ashlin Richardson, a Computer Science Ph.D. Candidate at the University of Victoria, during the 2012-2013 academic year. Many people associated with the Modernist Versions Project and its partners contributed to initial versions of this paper—all of whom we thank profusely, especially Adele Barclay, Daniel Carter, Alex Christie, Adam Hammond, and especially Katie Tanigawa. We also thank the MVP Board for its help and suggestions on earlier drafts. This includes invaluable advice from Susan Schreibman, Stephen Ross, Jentery Sayers, Dean Irvine, Tanya Clement, and James Gifford.

Huculak has experience working in textual scholarship, and Richardson has many years experience developing software. Richardson was thus tasked to look at the code of each program to reveal, according to expertise, the way each program works.

The paper is a survey of current collation tools for the Modernist Versions Project. It builds upon Hans Walter Gabler’s white paper, “Remarks on Collation,” published online in 2008, and accounts for collation tool development since Gabler’s survey. This paper is not a history of collation tools in the humanities. It is a survey and review of tools that are still being used for collation and visualization of collated texts. We review eight tools that have been developed specifically for textual collation, one tool developed for displaying the Text Encoding Initiative’s parallel segmentation in a web browser, and two version control systems.

List of tools tested in this survey:

- JUXTA-COMMONS + JUXTA DESKTOP
- TUSTEPTXSTEP
- TEI COMPARATOR
- TEXT::TEI::COLLATE
- COLLATEX
- DV COLL
- SIMPLETCT
- HRIT TOOLS (NMERGE)
- VERSIONING MACHINE
- GIT
- SVN (SUBVERSION)
Collation has three meanings in terms of literary criticism: First, in book production, collation is “to assemble sheets or gatherings for binding.” In bibliography, collation is “to analyze and record the number, order, and arrangement of leaves and gatherings in a book.” Finally, in textual criticism, collation is “to compare one text with another to discover textual variation” (Abbott and Williams).

Editors interested in bibliographical and textual studies seek out textual variation in order to discover and annotate differences between texts in order to chart the genealogy of a text and make critical assumptions about a work’s history. Traditionally, this has been a painstaking process. The “Wimbledon Method” of collation is to use one’s fingers to trace the lines in two books simultaneously in order to discover variation. Charlton Hinman invented the “Hinman Collator” in the 1940s by using strobe lights and mirrors to indicate differences between two editions printed using the same plates.

With the advent of computing, collation has taken on a broader meaning as “the general term for the process and function of determining the sorting order of strings of characters. It is a key function in computer systems; whenever a list of strings is presented to users, they are likely to want it in a sorted order so that they can easily and reliably find individual strings” (http://www.unicode.org/reports/tr10/). This “string sorting” can range from DNA sequencing to comparing two or more text files of a given work. Since the 1970s, software has been specifically developed to aid scholars in finding differences between texts in order to formulate critical assumptions about literary heritage.

The Tübingen System of Text Processing tools (TUSTEP) is one of the oldest textual-data processing programs in the humanities, and it was famously used by Hans Walter Gabler to edit his 1984 Synoptic Edition of James Joyce’s Ulysses (http://www.tustep.unituebingen.de/tustep_eng.html). Jerome McGann’s Nineteenth-century Scholarship Online (NINES) developed its own collation program called Juxta in order compare texts as part of that project, which has recently been released as a web-based version.

The Modernist Versions Project is interested in using textual collation tools as part of its workflow for comparing texts. This whitepaper examines twelve textual collation tools in order to account for developments in the field since Gabler’s original investigation.

Scholars need a way to reliably identify differences between texts. Though there is an array of tools developed for collation processing, each tool has been developed with a specific use case in mind. Some tools were developed for specific projects and may not scale to uses outside of that project.

The problems this white paper seeks to answer are: What tools are currently available for textual collation? What are their relative weaknesses and strengths? How can these tools be used by members of the Modernist Versions Project in collation and visualization work?
Current State of Knowledge

The current state of textual-collation software development may be divided into two camps: those tools that support the Text Encoding Initiative, and those that seek to represent variation in non-TEI expressions—whether these expressions be text based or based on new ways of representing variation (see Desmond Schmidt’s “A Data Structure for Representing Multi-Version Texts Online”).

Since many of the collation tools we review were developed for a specific project in mind, most of the tools are limited to simple collation. Larger systems, like TUSTEP, operate on a modular basis, and can accommodate different parts of textual scholarship and publication workflows.

One of the first widely successful textual collation software used by textual critics is the The Tübingen System of Text Processing tools (TUSTEP) developed by Wilhelm Ott at the University of Tübingen. Ott describes TUSTEP as:

a professional toolbox for those academic fields where texts are the object of research. Its potential is illustrated by two examples: (i) typesetting a TEI-lite encoded text, using the TEI tags as formatting instructions; (ii) preparing a critical edition, starting from automatic collation, then semi-automatically selecting the ‘substantial’ variants from the collation results, transforming them into a critical apparatus, and publishing the edition both in print and electronically. (Strategies and Tools)

First implemented in the 1970s (http://tustep.wikispaces.com/TUSTEP), TUSTEP has had a remarkable lifespan as a suite of tools used for over forty years. AT DH2012, Wilhelm Ott and Tobias Ott announced the development of a new tool, TXSTEP, “an XML-based program for scholarly research in the text-based humanities,” built on the “architecture” and “experience” supporting TUSTEP.

Though useful and well-used, TUSTEP’s programming was originally written with an assembly language. It is highly complex and requires training for use, and it does not take advantage of up-to-date standards; this is one reason TXSTEP was developed since it offers “an up-to-date established syntax,” uses a more friendly interface in terms of XML editing, and offers, “to a certain degree, a self-teaching environment.” One challenge that persists for some textual scholars, however, is that, at the time of writing this white paper, the supporting documents are written solely in German. Moreover, the program is still a prototype and has not yet been released for general use.

The newest tool we tested is the JuxtaCommons environment developed by NINES. This web-based, open-source program, was released as a beta in 2012, and offers users the ability to upload different file types and collate and visualize them within the browser. Also under development is the European CollateX environment, which, according to our documentation, collaborates frequently with the Juxta development group. Both these tools offer strong and competing ways of working with textual variation.
Other tools currently available include the TEI Comparator, Text::TEI::Collate, and DV-COLL, which were developed for specific projects (TEI Comparator was used in the Holinshed Project, for example). Finally, there are tools currently under development, like HRIT tools, based on Desmond Schmidt’s nMerge.

Collation remains an important part of textual scholarship and editing, especially as projects seek out ways of displaying multiple texts online. Editing Modernism in Canada, for example, is a multi-year project to publish underrepresented Canadian modernists online. The Project Director, Dean Irvine, has expressed interest in allowing readers to see multiple witnesses of text side-by-side in the browser. The need for tools that identify differences between texts remains as strong as ever as print works are remediated online.

Finally, there are developments in the field in terms of “versioning.” Matthew Kirschenbaum explores the role of software versioning in artistic literary production in his chapter “Save as” in Mechanisms. The tools tested in this survey are largely concerned with works that exist in paper form and have been digitally remediated into text files. Since our primary area of interest is modernism, we limited ourselves to these types of tools since the material with which we work is print-based. This survey is not exhaustive. A larger survey of collation, authorship, and publication tools (following the work of Kirschenbaum) is warranted.
Contextual Analysis

Modernist scholars face particular difficulties when collating textual versions of a work. The primary challenge arises from the complicated transatlantic copyright situation of modernist production, as well as the author’s own interests in the editorial process. Modernist authors, more so than their predecessors, tended to be more engaged in the editorial and publication process of their work. George Bornstein’s book-length study, *Representing Modernist Texts: Editing as Interpretation* (1991), explores this phenomenon in more detail. Bornstein notes that modernists’ editorial labor was often their own art. Their effort to control the process of textual production involved not only authority over the text itself but also determination of the form in which it appeared to the public and influence over institutions of transmission, whether magazines, anthologies, or entire publishing houses. (2)

The “institutions of transmission” had undergone rapid technological and social change around the turn of the twentieth century with the rise of the periodical press and mass-production in printing. For example, William Butler Yeats not only published in large circulating newspapers in Dublin, he started his own little magazines in Dublin and eventually collaborated with his sister at the Cuala Press in bringing out editions and different versions of his work. Bornstein argues that this editorial control, combined with copyright laws, created a situation where the “protean” nature of modernist writing—one which reveals multiple versions of a work—was concretized by the necessity of securing copyright in the United States. Bornstein notes, Despite the early adroitness of Yeats, Pound, or even Eliot in manipulating editors and publishers (and they themselves were editors, of course), the modernists eventually settled into long-term copyright arrangements with their publishers: in America, if you wanted to read Yeats, you turned to Macmillan; if Moore, to Viking; if Pound or Williams, to New Directions; if Eliot, to Harcourt Brace for the poetry and Farrar Strauss for the prose; and in England for Yeats again to Macmillan and for many of the other poets to Faber, where Eliot himself served as editor and director. The result was to “freeze” the principal texts in the form distributed by those publishers, to the loss of all earlier forms, and if writers like Yeats continued their revisions, the latest ones were apt to shove aside all the others. Readers of Moore's "Poetry,” Pound's *Cantos*, or Yeats's *Collected Poems*, for example, were simply furnished with the latest form of those poems, with little or no indication that earlier forms even existed, let alone what they were. *(Material Modernism 39)*

The illusion of fixity was further reinforced by the New Critical obsession with authorial intention and the “well-wrought urn,” concepts supported by market realities rather than the actual artistic process of creation. As Bornstein notes, by returning to the multiple sites of publication, we do “merely” create “a series of variants but, more importantly, [we create] a physical enactment of the process of transmission of modernist poetry” (35).

New Modernism

Bornstein’s focus on studying all versions of a given work is very much in the spirit of the rise of the “new modernist studies” (expressed by Douglas Mao and Rebecca Walkowitz in *Bad*...
Modernisms, 2006), which articulates a need to open criticism to protean nature of modernist production in all of its forms. By examining modernism “in its original sites of production and in the continually shifting physicality of its texts and transmissions results in alternative constructions very different from current ones. Such views emphasize historical contingency, multiple versions, and the material features of the text itself” (Material Modernism 1). To collate multiple versions of a text is both an act of recovery and a way of constructing new meaning in our understanding of textual transmission.

The Modernist Versions Project is in a unique position to do this type of work since it is located in Canada. In Canada, a work enters the public domain fifty-years after the death of the author. This means that Canadian scholars may digitally reproduce and store material beyond the 1923 cut-off date in the United States as long as the material has entered the public domain in Canada. This allows us to do the acts of recovery outlined by Bornstein above, in the spirit of new modernist studies, as we attempt to recover the “protean” nature of modernist work by revealing and publishing multiple versions of the text. The classic example used by Bornstein to show the importance of collating modernist works is that of Marianne Moore and her poem “Poetry.” The poem underwent three major revisions over time: it got smaller every time it was published. By collating Moore’s work (as well as that of other moderns), scholars may reveal,

the significance of the existence of such multiple versions: by displaying her poem in such radically different material forms, Moore creates not merely a series of variants but, more importantly, a physical enactment of the process of transmission of modernist poetry. (Material Modernism 35)

In this context, we undertook this current study to identify collation programs currently available to textual scholarship. We needed to know what the programs were, how the operated, and what type of collation and analysis we could perform with these tools.
Methodology

Ashlin Richardson (Computer Science doctoral student) was provided with two witnesses of Joseph Conrad’s *Nostromo* in XML format (TEI-P5), prepared by Katie Tanigawa, and .txt files of *Ulysses*, prepared by Huculak, to use as constants in testing.

Richardson, a computer scientist, and Huculak, a literary scholar, realized they had dissimilar conceptual notions of collation due to the term’s use in their respective fields. Richardson noted that the act of collation must remain distinct from the act of classification.

- **Collation**: “the assembly of written information into a standard order.” We emphasize that Collation is the ordering of information; in contrast
- **Classification** (a related but distinct task) is “concerned with arranging information into logical categories” (Wikipedia).

According to the discussion of textual collation and computing systems presented in Hans Walter Gabler’s “Notes on Collation,” some fundamental activities requisite to textual collation are as follows:

- Grouping documents by their likenesses and identities
- Decomposing texts into smaller features (fragments)
- Searching based on image features and/or markup
- Registration of minute differences (and also similarities) between texts (and the parallel visualization between multiple documents)
- The output of “lemmatized apparatus lists”; i.e., automatically generated lists of coordinations between the above differences (and similarities).
- Construction of paths through collections of text materials by observing correspondences and/or correlations between features—we should expect the result of collation to be the principal (dominant/most significant/most comprehensive/ most persistent) of the above paths (robustness to highly divergent versions of text). The result is a sequence of “reference points” which constitute the progression of the “base text”; these reference points are used as explicit references for textual comparison between versions.
- Contextual visualization of all of the above: several methods of interleaving the coordinated information are possible (including side by side or columnar representations, horizontal line-by-line representations). Such vertical or horizontal representations may be elaborated through version-by-version stacking of the constituent features or, alternatively, by sequential agglutination - a representation of the collation result in a “sequential run-on” format. Agglutination refers to the activity of copying and merging of constituent features of the different textual versions, with respect to the reference points that are representative of the collation result.
- Interactive modification of the collation result allowing user feedback to improve the result, based on human modification of results. Such modification should allow textual regrouping operations, and/or modification of the “reference points” arrived at by collation.
• Representation/markup of ancillary information such as page numbering, layout, formatting, and/or illustration is also advantageous.
• Special treatment of, and the ability to select different processing options for: variations in white-space and other special characters, different types of character sets/textual encodings is requested.

Based on these collation activities, Huculak and Richardson developed the following questions to ask of the software under consideration:
• What collation tools and/or textual analysis tools are presently available in 2013?
• What do the collation tools available have in common? How do they differ? (A detailed “meta-collation” analysis should be performed, to ascertain the commonalities and differences of the collation tools that are available.)
• How many input data formats should collation/textual analysis tools support?
• What internal representations for textual data are supported?
• Specifically, what visualization methods are available for the visual representation of collation results?
• To enhance, support, and add flexibility/robustness to collation functionality, what visualization methods/algorithms are available for textual analysis in general?
• What visualization methods are available (or described) for user interaction with collation/textual analysis results, in particular, the user feedback and modification of collation results (through regrouping and/or modification of groups of text, and/or the reference points determined by collation)?
• How much user intervention should be necessary? What are the limits in terms of how automated collation/textual analysis might be?
• Users of software developed by the textual analysis community do not always possess a high degree of algorithmic fluency. Accordingly, the user documentation for textual analysis tools may not substantially reveal the “under-the-hood” details of the collation and/or other textual analysis algorithms involved. Furthermore, fully technical documentation may not be provided. Specifically, which algorithms are involved in contemporary textual analysis tools? Understanding of the underlying algorithms is extremely important, in order to be aware of the strengths and limitations of the approach.
• The algorithmic details should ideally be explained so that the strengths and limitations may be communicated to a non-specialist. Accordingly, can we present the algorithmic details simply, in plain English, in order to relate the strengths and limitations to the casual user?

After our testing of tools developed for literary scholarship is complete, we propose to ask the following questions in order to broaden our understanding of the field of collation across disciplines:
• What is the relationship between the algorithms (incl., e.g., protein sequence analysis, gene sequence analysis, spectral network analysis) used by the genetics/genomics/proteomics and/or mass spectrometry communities for data sequence alignment and analysis, and algorithms used for collation/textual analysis?
• Furthermore, above and beyond analysis of network structure within textual collections, what algorithms are available to make inferences about the network-based processes
(e.g., social phenomena) underlying the evolution and development of those textual works?

• Especially for collections of text that are extremely large (e.g., the internet, an entire library, or other large collections) we should expect the facilitation of robust collation functionality to be computationally expensive. Are textual analysis tools available within the distributed computing paradigm?

• What algorithms and visualization methods applicable to collation/textual analysis are available from the information technology and information theoretic disciplines (computer science, mathematics, data visualization, semantic analysis)? Fully hierarchical segmentation algorithms are just one example of methods to vastly enhance the visualization and analysis of collections of textual works (particularly those that are vast and/or divergent).

• What algorithms/methods are available to augment the collation/textual analysis tools available, with the aim of substantially improving and/or increasing the robustness of conventional collation tools, particularly for large, highly divergent and/or diverse collections of text) semantic analysis and/or other forms of situational awareness?

Summary of Testing:

• Programming Language: In what language/framework is the software programmed?
• Open Source: Is the software open source?
• Input: What input formats are accepted?
• Output: What output formats are generated?
• Interface: What type of interface does the program use (GUI, Command line, other)?
• Collation Algorithm: What algorithm is used (if provided)?
• Visualizations: How are results displayed?
• Web Service: Does application have API or other hook?
• Collation Type: Does the program use horizontal or vertical collation? (or both)
• Number of Files: How many files can be collated at once?
• Data Manipulation after Processing (ignore white space, etc.): Does the program allow for user input/manipulation during or after the collation process?
• Documentation: Why type of documentation accompanies the software?
• Installation Requirements: What are the hardware requirements for installation?
• Tool History: What is the history of the tool?
• Problems: What problems did we encounter during testing?
• Recommended: Do we recommend this tool for our specific use?
• Technical Experience: Roughly, how much user experience needed to install and run the software?
• Preprocessing Needed?: Does the user have to mark up text before processing?
• Features: What makes the software distinct?
Recommendations

Tools

JuxtaCommons provided the most robust, user-friendly environment in which to do collation. It is the ideal tool for beginners to become acquainted with collation since it allows users to upload various file formats and includes “point-and-click” functionality. Results can be visualized in the browser and shared via the web. JuxtaCommons also seems to be the most actively developed of the tools surveyed, and its team seems willing to adopt recommendations from its user-base. For example, a version of the Versioning Machine has been incorporated into the program so that viewers can choose VM as a visualization tool. JuxtaCommons also allows users to share data, which is quite useful when working across institutions. As an all-around tool for collation and visualization, JuxtaCommons provides the most advanced and easy-to-learn framework of the tools surveyed.

For general use, including work in the classroom, JuxtaCommons provides an easy way to collate texts and produce TEI P5 parallel segmentation.

In terms of documentation, JuxtaCommons could be more explicit in terms of the types of algorithms used, like the CollateX project, with which the Juxta group seems to collaborate. Also, when collating longer texts, JuxtaCommons takes some time to produce visualizations. We recommend that JuxtaCommons adopt a “status bar” so that users can track the progress of their work as it is collated.

CollateX also offers quite a robust framework in which to do collation work, but it requires more technical knowhow to operate and install. Its major strength is that it allows users to change collation algorithms, and it is extensively documented.

There are a few promising tools under development for collation, including the work of Desmond Schmidt at the AustESE project, and TEI :: TEXT :: COLLATE, whose clean code impressed Richardson as computer scientist. Richardson recommends that this program be supported since “The presentation of the code shows that it has good logical structure, and is highly modular, and entirely self-contained” (see page 38).

Observations

Explicitness

Our research shows that many programs lack explicitness in documentation in how they operate. The exception to this observation is the CollateX program, which has explicitly published which algorithms are used for collation. CollateX also takes the time to explain collation algorithms to the non-specialist. Moreover, CollateX allows users to change collation algorithms before processing. We recommend that all tool developers adopt this transparent approach to collation software.
**Siloes**
We also note that many projects have been developed in siloed environments. This is logical since most collation programs have been programmed with a specific project in mind. It may be useful to the community that we develop a common repository of collation software. This way, different projects can build on the successes and failures of previous projects in order to create more robust software.

**Collaboration**
One promising development we witnessed during our research was the collaboration between CollateX and JuxtaCommons in the development of their respective projects. Though each project has its own strengths, the developers of each project communicate with one another in order to build better products for the community at large. We hope this type of collaboration continues as these larger projects continue to develop.

Another point of collaboration we found useful was that between the English department and the Computer Science department. Huculak is familiar with the needs of textual critics and collation, and Richardson is a developer for other types of collation algorithms dealing with images. There is a lot of collation development work being done right now in terms of protein sequence analysis and genetics. Richardson suggests that the processes of collation in protein analysis are strikingly similar to that required by textual critics (both need to collate strings of letters). Richardson suggests that greater collaboration occurs between the Humanities and Computer Science since their interests overlap in this area.

**Future Versioning**
This survey examines currently existing collation tools and codex-based processing. Matthew Kirschenbaum is theorizing the work of version control systems *(see Mechanisms)*. A new study of born-digital version control systems and collating born digital apparatuses might be useful to the community.
Bibliography


<http://multiversiondocs.blogspot.ca/>.


Annex I: Results

Juxta-Commons & Juxta Desktop

www.juxtacommons.org

Programming Language: Java

Open Source: Yes. Source code available for web services API version and desktop version:
  • https://github.com/performant-software/juxta-service
  • https://github.com/performant-software/juxta-desktop

Input: Plain text, XML, HTML, DOCX, OpenOffice, EPUB, PDF files, Rossetti Archive Markup (RAM) XML, TEI XML, and TEI-P5 Parallel Segmentation formatted files

Output: HTML, XML, TEI-P5 (incl. Parallel Segmentation), web URL to results hosted online, results sent via email, results shared via iframe embed (facebook embed for social sharing of results coming soon).

Interface:
  • Desktop: GUI application
  • Online: Graphical web service

Collation Algorithm: For JuxtaCommons the collation algorithm is derived from CollateX (interedition.eu). Juxta (the desktop application) has its own collation algorithm (Schmidt) this is based on the algorithm presented in the monograph (Heckel).

Visualizations: several helpful visualizations (interactive annotation, heat map, histogram, parallel segmentation) and others are available. Please see the sample screen shots provided below.

Web Service: Web services API hooks are available (see above).

Collation Type: two-column side by side (“parallel segmentation”).

Number of Files: 2-15 (which seems ample for practical purposes).

Data Manipulation after Processing (ignore white space, etc.): a couple of options are available.

Documentation: detailed. Documentation is provided with substantial detail, the system takes almost no effort to use in order to produce collation results, and straightforward visualizations
are available (described at [http://juxtacommmons.org/guide](http://juxtacommmons.org/guide)). The collation algorithm is given some limited description within the documentation [http://juxtacommmons.org/tech info](http://juxtacommmons.org/tech_info) but

Further algorithm information is available here:


- And, detailed information should be available from the source code. Additional documentation and discussion available from developer group
  - [https://groups.google.com/forum/#!forum/juxta-dev](https://groups.google.com/forum/#!forum/juxta-dev)

- The collation algorithm itself is closely related to CollateX (from Interedition project).
  - [http://collatex.net/about/](http://collatex.net/about/)
  - [http://gregor.middell.net/collatex/api/collate](http://gregor.middell.net/collatex/api/collate)

**Installation Requirements:** Web based client is offered. Thus, no installation is required.

**Tool History:** Developed by Applied Research in Patacriticism group at U. Virginia (Jerome McGann), Nick Laiacona of Performant Software, and Duane Gran, and Bethany Nowviskie.

**Problems:** None; Though some time is needed for longer texts to process. There is not a “status bar” to indicate the progress of collation.

**Recommended:** Yes

**Technical Experience:** not required. For developer-level functionality, software development experience required.

**Preprocessing Needed:** No.

**Features:** helpful settings relating to the treatment of punctuation and capitalization
Screenshots (short sample data)

Heat Map Visualization

User Annotation Interactive Visualization

Pencil icon in witness list denotes annotations are present on document.

Click to insert annotation on this region across all witnesses.

Click to insert annotation on the base annotation.
**Side by Side Visualization**

![Side by Side Visualization]

**Histogram Visualization**

![Histogram Visualization]
**TEI XML element visualization**

**Ulysses Sample data: JuxtaCommons**
Outputs from the *Ulysses* test files are conveniently made available online as part of the web service. In order to see the (full) output produced using the *Ulysses* test set, please see the links below.

**Ulysses Text File Output – Heat Maps**
Base Witness: Ulysses_1922_telemachiadplus1
[http://juxtacommons.org/shares/ImlfNR](http://juxtacommons.org/shares/ImlfNR)

Base Witness: Ulysses_Gabler_Telemachiad1_MASTER
[http://juxtacommons.org/shares/AXVpAk](http://juxtacommons.org/shares/AXVpAk)

**Ulysses Text File Output – Side by Side Comparison**
[http://juxtacommons.org/shares/WJ7rQm](http://juxtacommons.org/shares/WJ7rQm)
"Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressinggown, ungirded, was sustained gently behind him on the mild morning air. He held the bowl aloft and intoned: Introibo ad altare Dei. Halted, he peered down the dark winding stairs and called out coarsely: Come up, Kinch! Come up, you fearful Jesuit! Solemnly he came forward and mounted the round gunrest. He faced about and blessed gravely thrice the tower, the surrounding land and the awaking mountains. Then, catching sight of Stephen Dedalus, he bent towards him and made rapid crosses in the air, gurgling in his throat and shaking his head. Stephen Dedalus, displeased and sleepy, leaned his arms on the top of the staircase and looked coldly at the shaking gurgling face that blessed him, equine in its length, and at the light untoned hair, grained and hued like pale oak."
Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressing-gown, ungirdled, was sustained gently behind him by the mild morning air. He held the bowl aloft and intoned:

Introibo ad altare Dei.

Halted, he peered down the dark winding stairs and called up coarsely:

-Come up, Kinch. Come up, you fearful Jesuit.

Solemnly he came forward and mounted the round gunrest. He faced about and blessed gravely thrice the tower, the surrounding country and the awaking mountains. Then, catching sight of Stephen Dedalus, he bent towards him and made rapid crosses in the air, gurgling in his throat and shaking his head. Stephen Dedalus, displeased and sleepy, leaned his arms on the top of the staircase and looked coldly at the shaking gurgling face that blessed him, equine in its
Side-by-Side Comparison

**Comparison Set**

- Ulysses Gabler Telemachiad1_MASTER.txt
- Ulysses 1922 Telemachiadplus1.txt

**Collation Histogram**

Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressing gown, ungirded, was sustained gently behind him by the mild morning air. He held the bowl aloft and intoned:

> Introibo ad altare Dei.

Stately, plump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressing gown, ungirded, was sustained gently behind him by the mild morning air. He held the bowl aloft and intoned:

> Introibo ad altare Dei.

Halted, he peered down the dark winding stairs and called out coarsely:

> Come up, Kinch! Come up, you fearful Jesuit!

Solemnly he came forward and mounted the round gunnest. He faced about and blessed gravely thrice the tower, the surrounding land and the awaking mountains. Then, catching sight of Stephen Dedalus, he bent towards him and made rapid crosses in the air, gurgling in his throat and shaking his head. Stephen Dedalus, displeased and sleepy, leaned his arms on the top of the staircase and looked coldly at the shaking gurgling face.

Solemnly he came forward and mounted the round gunnest. He faced about and blessed gravely thrice the tower, the surrounding country and the awaking mountains. Then, catching sight of Stephen Dedalus, he bent towards him and made rapid crosses in the air, gurgling in his throat and shaking his head. Stephen Dedalus, displeased and sleepy, leaned his arms on the top of the staircase and looked coldly at the shaking gurgling face.
TUSTEP

Programming Language: Fortran, C

Open Source: Yes: http://www.tustep.uni-tuebingen.de/down/accept.php
http://www.tustep.uni-tuebingen.de/tustep_eng.html
http://www.tustep.uni-tuebingen.de/tustep_eng11.html

Input: Plain text (ASCII), Magnetic Tape Cassette Format

Output: Plain text, Magnetic Tape Cassette Format

Interface: A command line and scripting language interface. To get an idea for the proprietary technical opacity of the programming/software paradigms involved, a sample screenshot of a text-processing script is provided (please see the following section on TXSTEP).

Collation Algorithm: Has a “collation” function, developed as a module, although many parameters are required for this function, which are provided in a list with alphabetical index. The collation function does not calculate differences between versions (the user must supply the differences in a separate file, which can be generated using a separate script, which is included in TUSTEP). Not readily apparent from the English documentation whether a “collation sample” script is available.

Visualizations: None (text only interface).

Web Service: According to http://www.tustep.uni-tuebingen.de/tustep_eng.html, TuStep is available from the TextGrid environment (a Web service). However, at http://www.textgrid.de/, TuStep is listed in none of the following:

- TextGrid handbook https://textgrid.sub.uni-goettingen.de/fileadmin/dokumentation/user-manual-2.pdf,
- Technical documentation https://dev2.dariah.eu/wiki/display/TextGrid/Main+Page#MainPage-TechnicalDocumentation,
- Online help https://dev2.dariah.eu/wiki/display/TextGrid/Main+Page.

Interestingly, under Philological tools, at

- https://dev2.dariah.eu/wiki/display/TextGrid/Philological+Tools, collateX (collatex.sourceforge.net) is listed (CollateX is developed as part of the EU-funded Interedition project http://www.interedition.eu/, via http://collatex.net/about/, open source development is available through github: https://github.com/interedition/collatex). The latter version is actively maintained, whereas the http://collatex.sourceforge.net/team-list.html version has not been updated since 2011.

Collation Type: line-by-line

Number of Files: Several
Data Manipulation after Processing (ignore white space, etc.): In the 612-page manual, numerous programming options for processing and text formatting are available.

Documentation:
Detailed documentation is available in German, e.g.,
- http://www.tustep.uni-tuebingen.de/tustep_eng.html
- http://tustep.wikispaces.com/TUSTEP-Wiki
- Documentation of limited scope is available in English. This does not include examples.
- http://www.tustep.uni-tuebingen.de/pdf/hdb93_eng.pdf

Installation Requirements:
- Fortran compiler and C compiler must be installed.
- Path to Fortran Compiler and C compiler must be provided to the installation program.
- Windows or Linux instructions are given for installation (in German).
- Examples for use included with the installation are limited and/or have limited documentation.

Tool History: The system has been developed at the university of Tubingen since the 1960s.

Problems: Many issues (see below).

Recommended: No.

Technical Experience: An advanced level of technical experience is required to run the software. To perform collation, apart from an understanding of the scripting environment (which is covered in an 612 page English manual from 1993), the user needs to understand the command syntax to load/import/convert data files. Furthermore, the required “COMPARE” and “COLLATE” commands each have technical options which are more than several in number, and probably too complicated for the non-specialist.

Preprocessing Needed: As indicated by the English language handbook, the amount of user intervention/preprocessing needed in the differencing command “COMPARE” (required as input to the command “COLLATE”) is unclear, as the document for “COMPARE” states “the program can handle omissions and insertions of any length, provided these are identified and indicated by the user”. This suggests that the automatic collation functionality is complicated and, user intervention/mark-up will be required to indicate correspondences between versions.

Features:
- Good text manipulation functions are available. This does not necessarily serve any advantage over other programming environments (such as, Python).
- Flexibility to change character set.
- Pattern matching functionality.
- Good flexibility of indexing features.
• Collation algorithm is present, although the documentation suggests it is outdated (in comparison to newer languages).
• Generation of word frequency lists is available (probably easier to compute this in a modern programming environment, such as Python).

Note: these features are only readily available to the TUSTEP/TUSCRIPT specialist.

Comments:
In computing terms, this is an archaic system, most of which consists of features that any standard operating system should be responsible for (e.g., executing a sequence of commands, querying the time, opening, closing, reading and writing text files, copying and pasting text, indexing, sorting, operations for manipulating data on magnetic cassette tapes).

Does have collation utility, but this is extremely difficult to use (requires that the user learn the TUSTEP language, which, again, is mostly representative of functionality present in any normal operating system).

Given the requisite investment in terms of programming learning curve, a user would have a vastly more fruitful experience to learn any modern high-level computing language, e.g., a scripting language such as PYTHON, rather than learn TUSTEP, because little additional functionality is gained from TUSTEP. TUSTEP lacks visualization (which modern high-level computing languages can readily offer). In conclusion, we suggest that use or further testing of this software package would be counterproductive.
**TXSTEP**

**Programming Language:** XML, TUSTEP scripts. Since TXSTEP is a really a front-end to TUSTEP, the programming languages involved are XML, TUSTEP, C, and Fortran.

**Open Source:** Yes. Prototype available for download from [http://www.tustep.uni-tuebingen.de/down/txstep_1205/](http://www.tustep.uni-tuebingen.de/down/txstep_1205/)

**Input:** plain text, XML, HTML, TUSTEP, RTF, PS, Word, and Excel formats.

**Output:** plain text, XML, HTML, TUSTEP, RTF, PS, Word, and Excel formats.

**Interface:** XML script input (below, left). Whether this will prove to be self-explanatory than the TUSTEP/TUSCRIPT scripting environment, remains to be seen.

![Figure 1: XML interface (TXSTEP)](image1.jpg)

![Figure 2: TUSTEP/TUSCRIPT interface](image2.jpg)

These screenshots were taken from [http://www.itug.de/2010/Wuerzburg/wue2010.html](http://www.itug.de/2010/Wuerzburg/wue2010.html).

**Collation Algorithm:** TUSTEP collation algorithm.

**Visualizations:** text-based output. No visual representation.

**Web Service:** web-services not provided. Strictly speaking, TXSTEP is an XML-based API/hook for TUSTEP. Not configured or designed for web-deployment.

**Collation Type:** line-by-line.

**Number of Files:** several (as is subject to the limitations of TUSTEP, as above).

**Data Manipulation after Processing (ignore white space, etc.):** the extremely limited English documentation available does not convey anything about the extent to which text-manipulation
functions from TUSTEP are made available in the XML interface, however, the German documentation indicates that detailed text-manipulation functions are available.

**Documentation:** English documentation is lacking. The main “current” project page is:
- [http://www.tustep.uni-tuebingen.de/txstep_eng.html](http://www.tustep.uni-tuebingen.de/txstep_eng.html)
- [http://www.tustep.uni-tuebingen.de/tustep_eng.html](http://www.tustep.uni-tuebingen.de/tustep_eng.html)

The “current” TXSTEP project page is listed at the above site. Installation instructions for Windows environment are provided:
- [http://www.tustep.uni-tuebingen.de/down/txstep_1205/readme_eng.txt](http://www.tustep.uni-tuebingen.de/down/txstep_1205/readme_eng.txt)

However, it is not clear how to satisfy all the dependencies (the installation of freely available command-line C and Fortran compilers are typically much simpler for Unix/Linux or Mac OS environments). For the prototype, rather limited documentation is available in German.

**Installation Requirements:** requires installation and configuration of:
- C compiler, such as the open/free compiler “gcc” available from gcc.gnu.org,
- A Fortran Compiler, e.g., the open/free Fortran compiler from gcc.gnu.org/fortran/ provided as part of the “gcc” family of utilities.
- TUSTEP from [http://www.tustep.uni-tuebingen.de/tustep_eng.html](http://www.tustep.uni-tuebingen.de/tustep_eng.html)
- An XML editor such as oXygen [http://www.oxygenxml.com](http://www.oxygenxml.com) is suggested/recommended (not a free product). Single user academic license for oXygen is $99 ($297 for academic single-user floating, classroom license $809, academic department license $3428, academic site license $8,380). If oXygen is not available (for which instructions are available), German instructions are available to execute TUSTEP software from the Windows Command Line.

**Tool History:**
TXSTEP was conceived in 2009 by Tobias Ott, an academic at Stuttgart Media University (also CEO of pagina GmbH in Tubingen) and presented at 2010 TUSTEP workshop in January 2010 at Trier University. The first developer’s workshop took place in May 2010 at Stuttgart Media University. The first prototype and status report were presented at the 2010 annual meeting of the International TUSTEP User Group (ITUG). The integration of further functions and functionality remains unclear. Currently, very little documentation is available for the prototype, so an evaluation of the features is not substantially possible.

**Problems:**
- Despite claims from the English introduction found on [http://tustep.wikispaces.com/TXSTEP](http://tustep.wikispaces.com/TXSTEP), the interface is not (yet) self-explanatory.
- Although the XML-based interface is an English-based scripting language, there is no English-based description of it.
- It is not yet clear how the XML-based interface corresponds to TUSTEP or what features of TUSTEP are integrated so far.
- So far, German language proficiency is required to conduct the demonstrations provided.
- Complete German documentation is not available either. E.g., the prototype being distributed does not contain a complete explanation of the file structure, or a substantially
complete description of the components, or their interrelationships. A simple listing of the XML-based examples is given in German.

- Although sample XML files are made available, they are sparsely documented (via comments made in German). Proper documentation of the examples is not present.
- Taking the claim of integration into oXygen and other XML editors into the context of other information, such as the state of available documentation, it seems likely that the proprietary editor oXygen will be, in effect, a de facto requirement (if the user is to realize the proposed advantages of TXSTEP).
- While an honest and genuine effort to bring a 35+ year old historical platform based on legacy-code (i.e., TUSTEP) to modernity the number of dependencies is quite large, the basis on the legacy-code structure remains.
- Current project status, available from http://www.tustep.uni-tuebingen.de/txstep_eng.html, has not been updated since July 2012.
- According to the above link, it is not clear that the indication (to release an updated version, incl. corrections, following the DH2012 meeting in July 2012) was carried out.

**Recommended:** Not yet. We hope that we may be able to recommend the software for users, as the flexibility of text processing offered by the system seems promising, only if improvements to the prototype become available, including adequate and sufficient documentation in English that substantially validating the developers’ goals to create a self-evident interface. Claims at http://tustep.wikispaces.com/TUSTEP and elsewhere asserting that TUSCRIPT (the TUSTEP scripting language) is a modern scripting environment have not been validated (the very existence of the TXSTEP interface rules out this possibility. Indeed, from http://rosettacode.org/wiki/Category:TUSCRIPT_examples_needing_attention the admission that the examples for TUSCRIPT “may be incorrect, poorly-written…” or otherwise “…unsatisfactory”, suggests that the stated modernization goals represented by TXSTEP may be made with great difficulty, if it is so deeply rooted in a technically dated predecessor.

**Technical Experience:** Unless substantial documentation is made available, currently an expert level of technical experience is required to install and operate the examples. For native German speakers who can follow the limited documentation, the level of technical experience is somewhat less.

**Preprocessing Needed:** Not required, at least for the text-processing examples supplied. Although sample scripts are provided to compute TEI-compliant differencing, it is not clear from the documentation whether a text-collation example was provided with the prototype, or if instructions were given on how to carry out a collation procedure.

**Features:** As in TUSTEP, there is the potential for a number of beneficial features:

- Good text manipulation.
- Flexibility to change character set.
- Pattern matching functionality.
- Flexibility of indexing features.
- Collation algorithm is present.
- Generation of word frequency lists.
Note: it is not clear whether all of these features from TUSTEP are in fact implemented in TXSTEP. Again, none of these features are necessarily advantageous (see the comment on TUSTEP features, above).
**TEI Comparator**

**Programming Language:** java web application built using the Google Web Toolkit API

**Open Source:** Yes

**Input:** XML (designed to work with XML conforming to TEI guidelines, will work with any XML file).

**Output:** TEI (including user annotations)

**Interface:** Command Line

**Collation Algorithm:** Shingle Cloud Algorithm

**Visualizations:** Yes, for user annotation functionality only.

**Web Service:** Yes, but you have to set it up yourself (configuration is difficult).

**Collation Type:** Heat Map.

**Number of Files:** Compare two editions only

**Data Manipulation after Processing (ignore white space, etc.):** Yes, in GUI


**Installation Requirements:** Difficult installation process involving setup for: Database, web server (manual editing of Configuration files).

**Tool History:** Developed for the Holinshed Project

**Problems:** Security issues (see below)

**Recommended:** No

**Technical Experience:** High (see below)

**Preprocessing Needed?** Yes

**Problems:**
- Database-backed. The requirement of the database feature requires installation of a database, incl. a web server (this process is complicated, requiring numerous configuration steps, and potentially leaves the machine open to vulnerabilities/exploits).
• Algorithm is not explicitly described.

Features:

• Has a good wiki-style documentation [here](#) that is substantial and meaningful. While the details on the alignment algorithm called the *Shingle Cloud Algorithm*, are scarce on the WIKI, the details are presented clearly in the developer paper (please see references), which is an important reference (that should be consulted for references to other alignment approaches).

• The algorithm is “fast”, but it requires pre-processing. First and foremost, user annotation/markup is required as pre-processing.

• The algorithm is run on a multi-step, command line basis.

• For the Hollinshead Project, the parameters for the command line processes were determined using statistical methods (which were not made available as part of the project).

• The graphical interface is only available for visualization/user annotation.

• Algorithm is run on a multi-step, command line basis.

• The matching operates on a coarse level, so lots of user feedback may be required.

• The description of the applicability of the algorithm is unsystematic: it “should work with any XML as long as the units being compared are of a similar paragraph-like size” [here](#).

• Requires computer science degree to edit several configuration file settings, and run (too many) different tools: 1) ant buildscript 2) mysql 3) apache tomcat. Setup requires manually editing file names, and configuration of each tool by the command line.

• Web interface allows user to confirm/delete proposed matches, create matches, and annotate individual items (or annotate the link present between two items that match).

Conclusion: The installation of the software requires an advanced degree of technical user intervention, the web-based/database requirement introduces security/user access issues, and the matching must be pre-annotated. As is evident from the screen capture below, the software has similar (spectral-visualization of differences) functionality to JuxtaCommons, but the matching requires user intervention.

JuxtaCommons has eclipsed this software in functionality.
Screenshots: Short data sample

Showing a match

Proposing more matches

Confirming a match
Browsing for Manual Navigation Matching
This Lut prooued a right worthie prince; a worthie prince; amending the lawes of the realme that were defective, abolishing euill customes and manners vfed amongst his people, and repairing old Cities and townes which were decayed, but specially he helpt molt to beautifie and to enlarge with buildings the Citie of Tropyant, which he had paffed with a strong wall made of lime and stone. London enclosed with a wall. 

in the better paper fortified with divers firece towres; and in the west part of the same wall he erected a strong gate, which he commanded to be called after his name, Lut's gate, and fo unto this day it is called Lutgate, the only chauued in yonge pronuciated as in the word. In the same citie also he forsooke for the more part, he humifie caufed buildinges to be made between London Stone & Lutgate, humifie for humifie not faire from the land gate a faire palace, which is the Fifth of Londons palace.
Text::TEI::Collate

Programming Language: Java.

Open Source: Yes. Available from CPAN and GITHUB:
- [https://github.com/tla/ncritic/tree/master/cpan/Text-TEI-Collate](https://github.com/tla/ncritic/tree/master/cpan/Text-TEI-Collate)

Input: JAVA strings, Plain text files, JSON format

Output: JAVA strings, Plain text files, JSON format, TEI-parallel segmentation format.

Interface: Command line (Java-based programming API). Text interface (STDIN/STDOUT).

Collation Algorithm: A collation algorithm is provided, although it does not handle transpositions (permutations) according to [http://cpansearch.perl.org/src/AURUM/Text-TEI-Collate-2.1/TODO](http://cpansearch.perl.org/src/AURUM/Text-TEI-Collate-2.1/TODO). Thus, the collation algorithm is not recommended for use.

Visualizations: Does not provide visualization (see: [https://github.com/tla/ncritic/blob/master/cpan/Text-TEI-Collate/doc/guide_to_scripts.txt](https://github.com/tla/ncritic/blob/master/cpan/Text-TEI-Collate/doc/guide_to_scripts.txt)), output is provided in TEI-parallel segmentation format for visualization in other software, e.g., phylogenetic/dendritic style visualization made available in, e.g., [http://mobyle.pasteur.fr/cgi-bin/MobylePortal/portal.py?form=pars](http://mobyle.pasteur.fr/cgi-bin/MobylePortal/portal.py?form=pars)

Web Service: Not made available as a web service. However, being Java-based, it would not be difficult to operate it from a conventional web server.

Collation Type: Vertical (line-by-line) collation/differencing.

Number of Files: Multiple (not limited by the software).

Data Manipulation after Processing: Unknown

Documentation: Detailed documentation is present in terms of substantial comments and/or examples within the source code. Expert level of technical proficiency may be required to follow these. Some auto-documentation is produced from the embedded source-code comments, e.g.,

Installation Requirements: Java Standard Development Kit (SDK).

Tool History: developed by Tara Andrews of the Oxford History Department: [http://byzstud.history.ox.ac.uk/~senior_members/andrews_tara.html](http://byzstud.history.ox.ac.uk/~senior_members/andrews_tara.html). It was used successfully to edit the Armenian-language Chronicle of Matthew of Odessa.

Problems:
• Documentation is limited, although this poses no problem to the intermediate/advanced Java developer, because the exposition of the source code is logical and coherent.
• Collation algorithm is not yet current.

**Recommended:** Not recommended for use at this time, but is highly recommended for follow up of further developments, as they are made available.

**Technical Experience:** Intermediate Java development experience (or greater).

**Preprocessing Needed:** No.

**Features:**
- TEI-compliant output
- Also, GraphML support (uses hierarchical/graph-based formalism).

**Comments:**
From http://cpansearch.perl.org/src/AURUM/Text-TEI-Collate-2.1/README, this is a scholarly text collation program, which has been used successfully for editing the Armenian-language Chronicle of Matthew of Edessa. Its documentation and test suite is half-done, its functionality is mostly there but not complete, and there are some improvements to the algorithm that could be made.

The code was developed by a one-person project and is also a work in progress. Admittedly, the collation algorithm available is not current, nor is substantial documentation available (thus the code is obscure to anyone other than a Java specialist) so we do not generally recommend this project for use at this time. However, there are some noteworthy aspects:

• Modern languages and programming paradigms are used.
• The author is coherent in her exposition (despite the incomplete nature of the project).
• The presentation of the code shows that it has good logical structure, and is highly modular, and entirely self-contained.
• The author shows clearly the need for hierarchical/dendritic/phylogenetic style visualization (i.e., uses a graph-based programming formalism).
• The author shows clear aware of the present deficiencies of the work.

Based on the above, we heartily recommend following up on the algorithmic developments here in the future. This could make a highly useful finished product, when the (stated) deficiencies are addressed, and substantial documentation is provided.
**CollateX**

**Programming Language:** Java.

**Open Source:** Yes. Available from:
- Interedition on GitHub: [https://github.com/interedition/collatex](https://github.com/interedition/collatex)
- Main development site: [http://collatex.net](http://collatex.net) (last updated 2013).
- Old development site: [https://launchpad.net/collatex](https://launchpad.net/collatex) (last updated July 2010).

**Input:** plain text, JSON, XML

**Output:** JSON, TEI P5, GraphML, GraphViz DOT, SVG vector graphics.

**Interface:** Command Line Interface. Web interface available (see below).

**Collation Algorithms:**
1. Dekker. The most mature algorithm offered by CollateX, the Dekker algorithm aligns an arbitrary number of text versions, optimizes local alignment of partial token sequences (phrases), and performs transposition detection.
2. The Needleman-Wunsch. This well-known global alignment algorithm broadly applied in Bioinformatics and Social Sciences searches for an optimal alignment (based on a score function which penalizes gaps) of an arbitrary number of versions.
3. **MEDITE.** CollateX features an experimental implementation of the algorithm of Bourdalliet et. al. Like the Dekker algorithm, this algorithm takes transpositions into account.

**Visualization:** Graph Data (working with Juxta as model for visualization)

**Web Service:** Yes. Multiple web service implementations are available online. The developer (less than casual user) can also set up/administer.
- One using YUI framework and JavaScript
- Another using PHP and Java

**Type of Collation:** Tokenization / differencing with multiple choices for Collation algorithm (as above) All Collation Algorithms used in CollateX are based on a graph data structure, the “variant graph”, and use a progressive alignment technique.

**Number of Files:** Multiple.

**Data Manipulation after Processing** (ignore white space, etc.): Yes.

**Documentation:** Detailed.
- [http://collatex.net/doc/](http://collatex.net/doc/)
- [http://collatex.net/apidocs/](http://collatex.net/apidocs/)
Installation Requirements: Local installation of Algorithm requires only Java. Requirements of installing your own web service include, depending on the configuration:

- Java, YUI, and Apache Cocoon
- Java and PHP

Multiple implementations of web services are available. To run an existing web service, no installation is required.

http://collatex.net/demo/
http://gregor.middell.net/collatex/api/collate
http://interedition-tools.appspot.com

Tool History: Developed for Interedition project. Can be used as a module within TUSTEP.

Recommended: Yes

Technical Experience: All levels.

Preprocessing Needed: No

Important Features:

- Accompanied by core java library with web services (this can be embedded).
- Simple demo web application is provided.
- Python integration API is provided.
- Graph-based data model.
- Documented for compatibility with Ubuntu Linux, Mac OS X, Windows (XP/Vista/7).

Exceptional Features

- The documentation http://collatex.net/doc/ is exceptionally clear regarding the functionality available, complete with a straightforward description of the data model, and algorithms.
- Have an ongoing relationship with the developers of Juxta. They are working together to have their products conform to TEI standards.
- The Collatex framework supports multiple alignment/collation algorithms.

Other Features

- Substantial GUI not provided
• The second algorithm does not account for the possibility of transposition of text segments. We present the abstract of (which is very important not only historically, but also for the emphasis placed on the relationship between textual and bioinformatics algorithms) as follows:

  o A computer adaptable method for finding similarities in the amino acid sequences of two proteins has been developed. From these findings it is possible to determine whether significant homology exists between the proteins. This information is used to trace their possible evolutionary development.

  o The maximum match is a number dependent upon the similarity of the sequences. One of its definitions is the largest number of amino acids of one protein that can be matched with those of a second protein allowing for all possible interruptions in either of the sequences. While the interruptions give rise to a very large number of comparisons, the method efficiently excludes from consideration those comparisons that cannot contribute to the maximum match.

  o Comparisons are made from the smallest unit of significance, a pair of amino acids, one from each protein. All possible pairs are represented by a two-dimensional array, then, all possible comparisons are represented as pathways through this array. For this maximum match only certain of the possible pathways must be evaluated. A numerical value, one in this case, is assigned to every cell in the array representing like amino acids. The maximum match is the largest number that would result from summing the cell values of every pathway.”

Screenshots
Screenshots of the Collatex.net demo web service are provided. It was not possible to run the demo service with the full *Ulysses* test files, because doing so crashed the service. It was run with portions of the files only. Furthermore, all of the outputs: the graph visualization, alignment table, GraphViz and GraphML graph markup files, and TEI-P5 XML format results, really require further software to properly visualize (hence, e.g., the JuxtaCommons interface).
Collatex.net/demo text input (Ulysses)

Algorithm:

Examples:

Witness #1:

```plaintext
Stately, pump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressing-gown, unriggled, was sustained gently behind him by the mild morning air. He held the bowl aloft and intoned:

—Introibo ad altare Dei.

Heed, he peered down the dark winding stairs and called cut coramly:

—Come up, Kind/! Come up, you fearful Jesu!

Collatex.net/demo output (Ulysses)

Results

Variant Graph

Alignment Table

GraphML

```xml
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1"><g id="graph">
  <g id="nodes">
    <marker id="arrow" viewBox="0 0 10 10" refX="0" refY="0" rotate="0">
      <path d="M0,0 L10,0 L10,10 L0,10 L0,0 z"/>
    </marker>
    <circle cx="50" cy="50" r="10" fill="#00ff00" stroke="#0000ff" stroke-width="2"/>
    <circle cx="100" cy="100" r="10" fill="#ff0000" stroke="#0000ff" stroke-width="2"/>
    <circle cx="150" cy="150" r="10" fill="#0000ff" stroke="#0000ff" stroke-width="2"/>
    <circle cx="200" cy="200" r="10" fill="#0000ff" stroke="#0000ff" stroke-width="2"/>
    <circle cx="250" cy="250" r="10" fill="#0000ff" stroke="#0000ff" stroke-width="2"/>
    <circle cx="300" cy="300" r="10" fill="#0000ff" stroke="#0000ff" stroke-width="2"/>
  </g>
  <g id="edges">
    <line x1="50" y1="50" x2="100" y2="100" stroke="#0000ff" stroke-width="2"/>
    <line x1="100" y1="100" x2="150" y2="150" stroke="#0000ff" stroke-width="2"/>
    <line x1="150" y1="150" x2="200" y2="200" stroke="#0000ff" stroke-width="2"/>
    <line x1="200" y1="200" x2="250" y2="250" stroke="#0000ff" stroke-width="2"/>
    <line x1="250" y1="250" x2="300" y2="300" stroke="#0000ff" stroke-width="2"/>
  </g>
</svg>
```

GraphViz

```text
digraph G {
  v0 [label = ""];
  v1 [label = "Stately, pump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressing-gown, unriggled, was sustained gently behind him. ";]
  v2 [label = ""];
}
```

TEI-PS

```xml
<rs id="W1">
  Stately, pump Buck Mulligan came from the stairhead, bearing a bowl of lather on which a mirror and a razor lay crossed. A yellow dressing-gown, unriggled, was sustained gently behind him."
</rs>
```

[1]
**DV-COLL**

**Programming Language:** Microsoft Visual Basic (VB)

**Open Source:** No

**Input:** plain text

**Output:** plain text

**Interface:** GUI

**Collation Algorithm:** Unknown

**Visualizations:** Output is displayed in text format

**Web Service:** N/A

**Collation Type:** vertical (line-by-line) collation.

**Number of Files:** Many

**Data Manipulation after Processing (ignore white space, etc.):** N/A

**Documentation:** Substantial documentation including installation instructions can be found at [http://donnevariorum.tamu.edu/resources/down/index.html](http://donnevariorum.tamu.edu/resources/down/index.html)

**Installation Requirements:** software is available for Windows. No other requirements.

**Tool History:** Developed for Donne Variorum by staff at U. Southern Mississippi (the copyright for the software is held by Prof. Gary Stringer at Texas A&M University).

**Problems:** Runs on Windows systems only.

**Recommended:** No

**Technical Experience:** Little

**Preprocessing Needed:** No

**Features:** Minimal but functional collation system: no special features.
Ulysses Sample Data: Screenshots
We tested the DV-Coll on a Windows XP system. The main interface is simple, as follows:

This allows the user to select a list of versions to compare. Here, we selected:

1. Ulysses_1922_telemachiadplus1
2. Ulysses_Gabler_Telemachiad1_MASTER

The MASTER was used as the base for comparison. We pressed “Collate” and were presented with the following options:

As per the instructions, we selected “Word-level” (the DV COLL documentation indicates that “Line-Level” does “not do any actual comparing”).

The collation executed, and we were asked to save an output file:
We include a sample of the text output file here. To interpret this output, there are four possibilities, reproduced here from the DV COLL documentation:

1. **Exact Matches** - If the word in the variant line exactly matches the word in the base text, the program suppresses the word in the variant line, leaving a blank space in the output. This reduces clutter in the collation and allows you to focus on variants.

2. **Alterations** - If the word in the variant line differs in any detail from that in the base line, the variant will be situated directly under the base word and printed out in full.

3. **Insertions** - If a word appears in the variant line that has no counterpart in the base line, it will be interpreted as an insertion, enclosed in braces ({insertion}), and printed out at the appropriate place in the collation.

**Omissions** - If a word that appears in the base line does not appear in the variant line, the omitted word is enclosed in angle brackets (<omission>) and printed directly under the base word. Ulysses Sample Data: Collation Output (Text Format)
bowl of lather on which a mirror and a razor lay crossed. A yellow

intrigue ad altare Del. which << a mirror << and >> a razor << lay >> crossed. >> yellow

dressing-gown, unbuttoned, was sustained gently behind him on the mild

argifried >> was << sustained >> gently >> behind >> dress >> ars >> the >> smile

morning air. He held the bowl aloft and intoned:

intrigue ad altare Del.

and <!ulture! Del.>

Muted, he peered down the dark winding stairs and called out
Muted, up {coarsely} {}

coarsely:

-Come up, Kinch! Come up, you fearful Jesuit!

up -> Kinch! -> Come up, -> you -> fearful -> Jesuit!

Solemnly he came forward and mounted the round gunrest. He faced

He came -> forward >> and >> mounted -> the -> round -> gunrest. >> He -> faced

about and blessed gravely thrice the tower, the surrounding land and the

-Come up, Kinch. Come up, you fearful Jesuit. <<rounding>> <land> and <the>

awaking mountains. Then, catching sight or Stephen Decius, he bent

mountains. >> Then. >> catching >> sight >> of >> Stephen >> Decius >> bent

towards him and made rapid crosses in the air, gurgling in his throat and

dies -> and >> node >> rapid >> crosses >> in >> the >> air >> gurgling >> in >> his >> throat >> and >> shaking >> his >> head. >> Stephen >> Decius >> displeased >> and >> sleepy >> leaned >> his >> arms

on the top of the staircase and looked coldly at the shaking gurgling face

Solemnly he came forward and mounted the round gunrest. at> He faced about and (blessed)

{grovely} {thrice} {the} {tower} {the} {surrounding} {country} {and} {the} {awaking} {mountains.}

{Then.} {catching} {sight} {cf} {Stephen} {Decius} {He} {bent} {towards} {him} {and} {node}

{rapid} {crosses} {in} {the} {air} {gurgling} {in} {his} {throat} {and} {shaking} {his} {head.}

{Stephen} {Decius} {displeased} {and} {sleepy} {leaned} {his} {arms} {on} {the} {top} {of} {the}

{stairscase} {and} {looked} {coldly} at {the} {shaking} {gurgling} {face} {that} {blessed} {him,}

{equally} {in} {its} {length} {and} {its} {light} {unshrouded} {hair} {graines} {and} {were}

{like} {scale} {ok.}
SimpleTCT

Programming Language: Java


Input: RTF, plain text

Output: RTF, plain text

Interface: GUI.

Collation Algorithm: N/A

Visualizations: N/A

Web Service: N/A

Collation Type: N/A

Number of Files: Multiple

Data Manipulation after Processing (ignore white space, etc.): N/A

Documentation (not detailed or detailed):
• Demonstration on Youtube http://youtu.be/r0KMuQj0o0g
• Description on main site http://opendaht.org/SimpleTCT.html

Installation Requirements:
Requires Java Run-time Environment (JRE)

Tool History: Developed by James O’Sullivan and Mary Galvin of University College Cork in June 2012.

Problems: User can select text, apply different tags, annotate these categories, and output the text collection in a linear format, organized according to these “themes” (categories). This may be helpful for manual text comparison, but it is not recommended here (for use, or testing) because the functionality it extremely limited and no processing functionality is provided. Effectively, this represents a manual text-collation environment. Nevertheless, it serves to illustrate clearly some basic (manual/user-intervention) features for textual comparison/analysis, for which it deserves some attention. SimpleTCT also deserves attention for radiating the kind of simplicity of interface that all philological tools should have (JuxtaCommons shows this kind of simplicity as well). We need philological tools that incorporate sophisticated algorithms, while keeping the simple feel of this.
Recommended: No.

Technical Experience: N/A

Preprocessing Needed: N/A. No automatic processing (only user markup).

Features: Compatible with Mac, Unix/Linux, and Windows.

- User intervention/annotation
- Useful for textual annotation, manual visual comparison, and thematic organization.
- Note: not a collation tool.

Screenshot
Here a sample screenshot is shown (taken from the main site) illustrating the application of colored text categories associated with labels (“themes”).
HRIT Tools (nMerge)

Programming Language: C, Java

Open Source: Yes. Available from multiple sites, including:
- https://sites.google.com/a/ctsdh.luc.edu/hrit-intranet/source-code
- http://code.google.com/p/hrit/source/checkout

Input: plain text, XML

Output: plain text, XML

Interface: GUI implementations available in Java and PHP. See https://sites.google.com/a/ctsdh.luc.edu/hrit-intranet/galleries for screenshots.

Collation Algorithm: NMERGE (command line tool) based on the valuable references [19-21]. These are noteworthy for the simplicity, clarity, and detail of information contained, whereby the workings of the algorithms are illuminated with clean exposition. These works also communicate much insightful information (from the perspective of the insider) about other prominent collation algorithm implementations. NMERGE is available from:
- http://code.google.com/p/multiversiondocs/

Visualizations: N/A. Unable to process texts

Web Service: web service implementations are available (including demonstrations):
- https://sites.google.com/a/ctsdh.luc.edu/hrit-intranet/galleries/cortex-restlet
- https://sites.google.com/a/ctsdh.luc.edu/hrit-intranet/galleries/hrit-php-demos
- http://hrit.etl.luc.edu:8080/cortexrestletform/

Collation Type: horizontal and vertical.

Number of Files: Multiple.

Data Manipulation after Processing (ignore white space, etc.): the options available will depend on the front-end implementation.

Documentation: Vast documentation is available. Some of the primary sources are listed here:
- Main Academic site: http://hrit.etl.luc.edu/
- Development site: https://sites.google.com/a/ctsdh.luc.edu/hrit-intranet/home
- Documentation: https://sites.google.com/a/ctsdh.luc.edu/hrit-intranet/documentation
• Developer handbook/manual: https://sites.google.com/a/ctsdh.luc.edu/hrit-intranet/documentation/handbook
• Project Center (at Loyola University Chicago): http://www.luc.edu/ctsdh/

Installation Requirements: Exceptionally simple command line arguments are available for installation. Textbook example of what an easy-to-use command line installation procedure should look like. First, they install the git version control system:

```
brew install git
```

Then download the latest hritserver code:

```
git clone https://github.com/HRIT-Infrastructure/hritserver.git
```

That creates a folder "hritserver" in that directory. Then you should run the installer:

```
cd hritserver
sudo ./install-macosx.sh
```

can easily update to the latest version by typing

```
git pull
```
in the hritserver directory.

Tool History: Funded by NEH Level II Digital Humanities Start-Up grants competition. Created at Loyola University Chicago’s Center for Textual Studies and Digital Humanities in collaboration with Emerging Technologies Laboratory at the Computer Science Department.

Problems: preliminary testing of the web-service based front-end at http://hrit.etl.luc.edu:8080/cortexrestletform/ did not succeed. We added one file successfully to the “Cortex”. Upon attempting to add a second file (below): we were greeted with an “out of Java heap space” error.

![Cortex interface](image.png)

This indicates that there are (somewhat artificial) memory limitations, with respect to the front-end implementation demonstrated there, but this is no reason to become discouraged with this collation implementation.

Recommended: N/A
**Technical Experience:** For the (demo) web service provided above, no technical experience is required. To use the NMERGE tool, software development experience is required.

**Preprocessing Needed:** No.

**Features:**
- Web services are available providing GUI’s, although it is not clear how substantially comprehensive the GUI’s currently available are.
- Online open source environment with collaborative editing/sharing features.
- Markup, annotation, tagging, linking, cross-referencing functionality for coordination of images, texts, and scholarly annotations.
- Compatible with TEI XML format for textual representation.
- Representation and analysis for image formats (incl., Optical Character Recognition (OCR) based functions).
- Features for creating correspondences between textual and image representations of texts.
- Collation is based on the CORTEX data structure, a graph-based data structure that corresponds to the “variant graph” as described in literature from the collateX.net (interedition.eu) community. The CORTEX data structure (and associated algorithms) is based on the Multi-Version Documents (MVD) concept, which is formulated and given detailed elaboration in [19-21], and implemented in the NMERGE tool.
- Any special tags/markup accompanying the text are automatically carried through the collation process, so the user has no need to re-enter them later.
Versioning Machine

Programming Language: JavaScript, XML, HTML, XSLT (for translation between XML and HTML)

Open Source: Yes

Input: XML, HTML

Output: XML, HTML

Interface: GUI

Collation Algorithm: N/A. Versioning Machine is a visualization tool (as below).

Visualizations: GUI is essentially a “web page” to view, in a side-by-side display, multiple textual versions, as encoded in TEI-P5 parallel segmentation format.

Web Service: Is a self-contained web folder.

Collation Type: N/A

Number of Files: Multiple

Data Manipulation after Processing (ignore white space, etc.): N/A (user does all the manipulation/processing).

Documentation (not detailed or detailed): Documentation is detailed and is representative of a good tutorial for TEI XML markup.

Installation Requirements: web browser. Since Versioning Machine is a viewer for (user-generated or user-edited) XML data, it is probably assumed (but not required) that the user has an XML editor (such as oXygen) available.

Tool History: Conceived and developed since year 2000 by Susan Schreibman, previously at University of Maryland Libraries (2005-2008) and Maryland Institute for Technology in the Humanities (2001-2004). The Versioning Machine was developed through collaboration with various personnel at institutions including the Digital Humanities Observatory (DHO), the Maryland Institute for Technology in the Humanities (MITH), and the Office of Digital Collections and Research (DCR) at University of Maryland Libraries.

Problems: None.

Recommended: As a visualization tool for TEI-encoded texts.
Technical Experience: Intermediate XML experience is required. No experience is required to run the demo.

Preprocessing Needed: User does all the markup and collation (or uses other utilities to do more automated processing). Program is a viewer for XML files corresponding to TEI-P5 (parallel segmentation).

Features: Compatibility is engineered and tested with many different web browsers.
**GIT**

**Programming Language:** C

**Open Source:** yes. Available from http://git-scm.com

**Input:** plain text

**Output:** plain text

**Interface:** GUI and/or command line. Can use merge/diff tool via GUI for parallel segmentation.

**Collation Algorithm:** associated diff tool (Meld and others).

**Visualizations:** horizontal (parallel segmentation) or vertical (line-by-line).

**Web Service:** The versioning system is offered as a web-based service (the source code provides client/server functionality) but the differencing/comparison is local.

**Collation Type:** horizontal (parallel segmentation).

**Number of Files:** 2-3

**Data Manipulation after Processing (ignore white space, etc.):** N/A.

**Documentation** (not detailed or detailed): Highly detailed documentation is available.

**Installation Requirements:** No special requirements. Exceptionally simple command line installation on Mac (with Homebrew installed): using the following command at the Terminal:

```
brew install git
```

**Tool History:** developed since 2005; conceived by Linus Torvalds (creator of Linux operating system). Please see http://en.wikipedia.org/wiki/Git_(software) for more details, if desired.

**Problems:** Please see section On Use of SCM for DH (below).

**Recommended:** No.

**Technical Experience:** required.

**Preprocessing Needed:** No

**Features:** color-coding of differences.
Screenshot of Ulysses text comparison. Note: collation errors!
SVN (subversion)

Programming Language: C


Input: plain text

Output: plain text

Interface: GUI, Command line

Collation Algorithm: opendiff (and other similar utilities)

Visualizations: Parallel segmentation (two-columns) or traditional “diff” (line-by-line comparison)

Web Service: Implements client and server functionality, to manage versions in a networked setting.

Collation Type: horizontal (parallel segmentation).

Number of Files: 2-3

Data Manipulation after Processing (ignore white space, etc.): can compress white space / ignore case.

Documentation (not detailed or detailed): Highly detailed documentation is available at http://subversion.apache.org/docs/

Installation Requirements: binary packages are available. To compile the package from the source code, there are other requirements (e.g., the gnu ‘gcc’ compiler suite).

Tool History: Subversion has a rich history dating to its creation in 2000 by CollabNet Inc., and is now a flagship product of the Apache Software Foundation with contributions from a large global developer community.

Problems: Please see section On Use of SCM for DH (below).

Recommended: No.

Technical Experience: required.

Preprocessing Needed: No.

Features: collation using associated merge/diff tools.
Stately, plump Buck Mulligan came from the doorhead, bearing a bowl of lather on which a razor and a mirror lay crossed, a yellow leather apron, unbuttoned, which contained gently barked his on the mild morning air. He held the bowl steady and intoned:

- Introduce author here.

Halted, he peered down the dark winding stairs and called out ceremonially:

Come up, Jemmy. Come up, you fearful Jesus.

- Introduce author here.

Stately he came forward and mounted the round granite. He faced towards the sun and began gravely to turn the stone, the surrounding land and the distant mountains. Then, catching sight of Stephen Dedalus, he bent forward and made rapid steps in the air, peeping in his throat as he shook his head. Stephen Dedalus, disinterestedly, leaned his eye on the top of the stonecutter and looked curiously at the shaking gargery that bled his eyes, square in the length, and at the light uncoloured had grown and lived like pale oil.

Buck Mulligan propped an instant under the mirror and then covered the bowl gently.

- Introduce author here.

He added in a preacher's tone:

For this, O dearly beloved, is the genuine Christian's body and soul all at once.

He nodded sidewise up and gave a long slow whistle of call then paused amid his own thoughts.

- Introduce author here.

The window of it he said gently. Your dearabone man, an ancient Greek.

He pointed his finger to friendly jest and went over to the parapet, laughing to himself. Stephen Dedalus stepped up, followed his gaze halfway and put down on the edge of the gargery, watching his stroll as he grasped his zipper on the parapet, slipped the bond in the bowl and fattened cheeks and neck.

He shouted above the gargery:

- Introduce author here.

Buck Mulligan's one voice went on.

- Introduce author here.

He smiled, half in earnest, and his eyes were fixed on the parapet, laughing to himself. Stephen Dedalus smiled, nodded his head gravely and his eye on the top of the stonecutter and looked curiously at the shaking gargery that bled his eyes, square in the length, and at the light uncoloured had grown and lived like pale oil.

Buck Mulligan propped an instant under the mirror and then covered the bowl gently.

- Introduce author here.

He added in a preacher's tone:

For this, O dearly beloved, is the genuine Christian's body and soul.

He nodded sidewise up and gave a long slow whistle of call then paused amid his own thoughts.

- Introduce author here.

The window of it he said gently. Your dearabone man, an ancient Greek.

He pointed his finger to friendly jest and went over to the parapet, laughing to himself. Stephen Dedalus stepped up, followed his gaze halfway and put down on the edge of the gargery, watching his stroll as he grasped his zipper on the parapet, slipped the bond in the bowl and fattened cheeks and neck.

He shouted above the gargery:

- Introduce author here.

Buck Mulligan's one voice went on.

- Introduce author here.

He smiled, half in earnest, and his eyes were fixed on the parapet, laughing to himself. Stephen Dedalus smiled, nodded his head gravely and his eye on the top of the stonecutter and looked curiously at the shaking gargery that bled his eyes, square in the length, and at the light uncoloured had grown and lived like pale oil.

Buck Mulligan propped an instant under the mirror and then covered the bowl gently.

- Introduce author here.

He added in a preacher's tone:

For this, O dearly beloved, is the genuine Christian's body and soul.
Appendix II

On SCM (Source Code Management) for use in DH
Ashlin Richardson

Unfortunately SCM systems do not provide a solution for the analysis of textual variation, for several reasons (an excerpt of [19] is reproduced here, as follows):

“SCM at its lowest level is based on calculating and recording the differences or ‘deltas’ between only two versions at a time, not the differences between all versions globally, as would be required for literary or linguistics texts. This applies even to a system like SCCS, which uses deltas interleaved within a single file (Rochkind, 1975).

SCM systems are based on the needs of programmers, i.e. on ‘configurations’ of software, as well as revision information (Gulla et al., 1991). In SCM granularity is also usually the line or module, whereas in literary and linguistics texts a much finer resolution of a word or even a character is required.

Versions in SCM are generally immutable (Munch, 1993), whereas for literary and linguistics texts versions are required to be always mutable. A programmer selects an old version of a program, edits it and commits the changed version back as a new text. The original version is necessarily preserved so it can be retrieved later. By contrast, an editor of a humanities text needs to select a particular version, work on it and save it back in its edited form, without changing the overall number of versions. Hence, although some techniques, such as algorithms for calculating differences, can be useful in formulating a solution to the current problem, actual SCM systems cannot be used to record textual variation.”

Even though SCM systems are designed to manage works (like software projects) that are actively being written (rather than study existing works) SCM’s deserve a great deal of attention, on the basis that they may offer insight and inspiration towards the modeling of the temporal evolution of word structures as a proxy for the evolution of thoughts, which is increasingly relevant as more and more of our activities are subject to increasingly detailed and continuous recording.

“I may have oversystematized Ulysses” ~ James Joyce

What if Joyce used SCM? Without doubt, the resulting wealth of breadcrumbs to follow would provide substantial further elaboration on the significance of the more mythological content, shedding great illumination on the underlying motivations and higher meanings of his kaleidoscopic works.