Algorithms for the Manipulation and Transformation of Text Variant Graphs

Tara L. Andrews

Institut für Geschichte, Universität Wien, Austria

Abstract
While text variant graphs are increasingly frequently used for the visualization of a text transmitted in multiple versions, the graph is also a very appropriate model for the querying and transformation of such a text in the course of producing a critical edition. This article describes the algorithms used in the StemmaREST repository for variant text traditions.

Keywords
Variant graph, Digital critical edition, Textual scholarship

1. Introduction

The *variant graph* has, in various forms, seen a steady uptake in use within the textual criticism community since its initial proposal by Schmidt and Colomb.[17] It presents a compact yet expressive way to take in the variation across divergent copies of the same text, where (in most implementations since the first) the readings, or text tokens, form the nodes of the graph and the witness pathways are indicated along the edges. Such a data structure can, moreover, easily be expressed in a TEI-conformant form when divergent witness pathways are encoded using the double endpoint attachment method of critical apparatus expression.[18]

While variant graphs has often been used simply as a means of representation and visualization, there has been some interest in the last ten years in using the graph as a site of user interaction with the text, both within the Stemmaweb project [6] and also as a feature of the TRAViz library [14]. These interactions can include the manual correction of a machine-produced collation, annotation of the text within that collation, or even the establishment of a canonical text and recording of emendations that an editor would make.

When variant graphs are used to interact with and even edit a text tradition, however, it is all too easy for logical mistakes to be introduced into the data structure. The problem of software support to ensure the correctness and consistency of an encoded critical text has been discussed by Burghardt in the TEI context [8]; a similar problem can easily arise in the context of the variant graph. The most damaging sort of mistake is that which alters the ‘ground truth’: that is, it causes the construction of versions of individual witness texts from the collation data

CHR 2023: Computational Humanities Research Conference, December 6 – 8, 2023, Paris, France
tara.andrews@univie.ac.at (T. L. Andrews)
https://ifg.univie.ac.at/en/about-us/staff/academic-staff/tara-l-andrews/ (T. L. Andrews)
0000-0001-6930-3470 (T. L. Andrews)

that are not the same as the versions that went into its making. We might refer to this as a *corrupted collation.*

The purpose of this paper is to give a formal definition of one variant graph model, describe its logical and mathematical properties, and provide some well-defined algorithms for how this graph may be interacted with and modified, in order to provide the functionality that scholarly editors would expect to have while avoiding the production of corrupted collations. The model described in this paper is implemented in the StemmaREST repository,[1] in use via the Stemmaweb tools for a variety of critical edition projects since 2021; many of the concepts and algorithms discussed have their roots in the Perl library `Text::Tradition`,[2] which was the predecessor to StemmaREST.

2. The StemmaREST graph model

Most models for variant graphs are broadly similar. After their initial description by Schmidt and Colomb, the most significant change in their conception and use came around 2010 with the adoption of the variant graph in the CollateX tool: unlike in the prior model, where all information was carried along the edges of the graph, the CollateX graph moved the reading text to the vertices and left only the witness labels on the edges.[10, 13] This change has been retained by all later versions of the variant graph of which the author is aware. Both the CollateX tool and the TRAViz tool also incorporate a visualization of text transposition; this is done respectively by drawing a line between the transposed readings (CollateX) or highlighting them simultaneously (TRAViz). The variant graph visualization in the Stemmaweb tool suite [5, 1] generalizes the idea of relations between readings, of which transposition is but one example. The model used today in Stemmaweb, now powered by StemmaREST, also includes data structures for the representation of witness information and text stemma hypotheses.

Let us first briefly outline the StemmaREST data model.[3] A `Tradition` may possess two or more `Witnesses`, which record information about different manuscript versions of a given text. The Tradition also possesses distinct `Sections`; these sections have a canonical order and, taken together, comprise the text. Each `Section` contains a single text collation composed of `Readings`, the `sequences` in which they occur, and information on how they are `related` to each other. A *reading* is a piece of text that is found in one or more of the *Witness*es. It is frequently a single word, but can be a sequence of words or even a fragment of a word, depending on the needs of the editor. A `Reading` object, then, is a transcription of its text as well as the metadata that is needed to situate it (e.g. a normalized or canonical form, the language it is in, whether it is the form that will be adopted in the edited text, and so on.) The `Reading` objects occur in some `sequence` in each witness, and may be `related` to each other insofar as they are variants of each other.

It is the structure and properties of the variant graph — that is, the subgraph whose vertices are `Reading` nodes and whose edges are `sequence` edges, and how this interacts with reading relations — that interests us here. In all versions of the variant graph to date, it is modelled as

[1]https://github.com/DHUniWien/tradition_repo
[2]https://github.com/tla/stemmatology
[3]An OWL-based version of this model was also recently presented at DH 2023; see [2].

a connected, rooted, directed acyclic graph (CRDAG). That is:

- *Connected* — the graph contains no disjoint subgroups.
- *Rooted* — there is exactly one vertex with an in-degree of zero; this is the origin, or root.
- *Directed* — all edges have an explicit direction. Paths through the graph must respect this direction.
- *Acyclic* — there is no path in the graph in which a node is visited twice.

In the StemmaREST variant graph, we refer to the root as the *start reading*; this is a special case of `Reading`. In addition to the CRDAG constraints, we add a constraint, which is that the variant graph has a single terminal vertex for all possible paths. This is another special case of `Reading` known as the *end reading*.

When we model variant text into a graph, we must also deal with the issue of witness layers. It quite frequently occurs in textual scholarship that a single witness can carry multiple versions, or layers, of the text in places; this is most easily seen when the scribe has made a correction to his or her own text. Since these corrections can often have some significance when it comes to understanding the transmission history of the text, we must account for the fact that a single witness might give rise to multiple paths through the graph. In order to conceptualize this, we posit that every witness has a *base layer* which is a single complete version of the text; the witness also has zero or more additional layers that represent alternative versions of specific subsets of that text. Formally, we may state the following:

1. Let C be a variant graph consisting of `Reading` nodes as vertices and `sequence` edges connecting them. Let S and E, respectively, denote the initial vertex and terminal vertex (start and end reading) of all paths in C. As described above, C is a connected, rooted, directed acyclic graph.
2. Let W be a manuscript witness; let R be a single ordered set of readings $R \in V(C)$ which represents the text carried in the base layer of W. A single non-cyclical path $P_W = S...E$ must exist from the start reading S to the end reading E whose set of inner vertices $V_P = R$.
3. For every additional layer L in W, there exists one or more disjoint ordered sets of readings $R_L \in V(C), R_L \notin R$; moreover, a single non-cyclical path P_L must exist between some two $r \in R$ whose inner set of vertices $V_{P_L} = R_L$.
4. C is the set of all paths P_W and P_L for a given set of witnesses.

This definition has some overlap with the ‘multi-version document’ (MVD) of Schmidt and Colomb, especially the construction of start and end readings and the constraint that a witness text must comprise a single unbroken path from start to end. The MVD model, however, did not specifically describe ways to account for separate layers in individual witnesses. Of note for the StemmaREST model is that witness layers are not necessarily regarded as full and separate versions of the text. Consider, for example, a manuscript that has multiple scribal corrections in the original hand; typically, where the pre-corrected state of the text can be deciphered, it will be represented in a critical apparatus with the notation ‘a.c.’ (*ante correctionem*). It would be possible (and has been done from time to time) to imagine a witness layer that runs from the beginning to the end of the text, taking none of the corrections into account, and refer to this as ‘the a.c. layer’. The epistemological problem here, however, is that there is no evidence that the text ever existed in a fully uncorrected state; it is much more likely that at least some of the corrections were made immediately during the composition or copying process. For this reason, the StemmaREST variant graph model only stores non-base witness layers where they are recorded to exist.

In StemmaREST’s Neo4J implementation, individual witness layer paths are represented using edge properties, where the property key denotes the name of a layer and the property value is an array of sigla whose witness texts follow that edge. The name of the base layer is `witnesses`; typical names for other layers might be ‘a.c.’ or ‘s.l.’ (*supra lineam*). The property key `witnesses` must be present and non-empty for every edge; property keys for other layers may be present and must be non-empty.

In this model, the traversal is a simple matter of following the correctly labeled sequence paths from the start node to the end node. We do this by providing a custom `Evaluator` algorithm to a Neo4J traversal description, as given in Appendix A.1. The validity of a given variant graph may be checked at any point by ensuring that every defined layer of every witness, substituted by the witness’s base layer where the given layer is not present at all, produces a single unbroken sequence of readings from start to end.

3. Relations between variant readings

One of the advantages of a graph model for text collation, as described in 2012, is the ability to indicate relations between variant readings as a separate category of edge in the graph — that is, to classify variants. This is a feature that is usually indispensable to scholarly editors, who must work through the variants and make determinations about how an edited version of the text should be constituted, or understand how the different versions of the text came to be. Variant classification is an essential first step, for example, for the construction of a *stemma codicum* in most accepted methods.[7, 20]

3.1. The `related` edge

The StemmaREST model allows for the following properties of a `related` edge between two readings:

- `type` — indicates the classification being made (see below).
- `scope` — indicates whether the relation applies only to this variant location, or to all variant locations for this text where the related readings occur. Possible values are `local` and `document`.
- `is_significant` — indicates whether, according to the judgment of the editor, the variation is stemmatically significant. The three values ‘yes’, ‘maybe’, and ‘no’ are accepted.
- `alters_meaning` — boolean to indicate whether the variation alters the semantic interpretation of the text.

- `non_independent` — boolean to indicates whether, according to the judgment of the editor, the variation was unlikely to occur in two unrelated manuscripts. This corresponds to the concept of a *Bindefehler*.
- `a_derivable_from_b`, `b_derivable_from_a` — booleans to indicate whether, in the judgment of the editor, one of the readings would have been correctable to the other by a typical scribe. This corresponds to the concept of a *Trennfehler*.

3.2. The relation type model

Of particular interest here is the typology of relations. The set of relation types that apply to any given text tradition is defined by the user; each relation type is given a name and description and is defined according to a set of properties. With the properties given, it is possible to create a limited hierarchy of relation types.

- `is_colocation` — a boolean to indicate whether this relation is defined within a single variant location. A transposition, for example, is not a colocation. One of the features of the StemmaREST model is to ensure that, if a colocation relation is set between two readings, those readings remain topologically at the same location in the text.
- `is_transitive` — a boolean to indicate whether this relation type is *transitive*.
- `is_generalizable` — a boolean to indicate whether this relation type may be applied at document scope.
- `bindlevel` — an integer to indicate how closely related readings of this type are. Lower values indicate closer relations. This value is used to deduce *implication* for transitive relations.

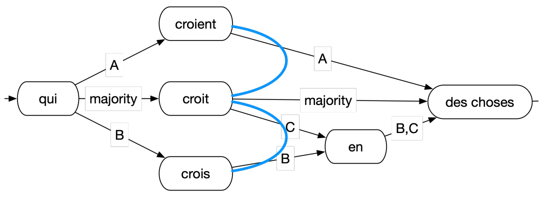
3.3. Transitivity and implication

The logic of transitivity and implication in the graph is crucial to the robust representation of text variation. To illustrate this, let us define two relation types `spelling` and `grammatical`; as their names imply, readings that are linked through relations of these types are, respectively, spelling variations and grammatical variations of each other. Both these relation types will have true values for all three booleans (the relation implies colocation of the respective readings; the relations are logically transitive; the relations may be applied to identical colocated pairs of readings throughout the text). Since spelling variants are much more often treated as 'the same reading' than grammatical variants are, we set a lower `bindlevel` value (e.g. 1) for the `spelling` relation type than the value we would set (e.g. 2) for the `grammatical` relation type.

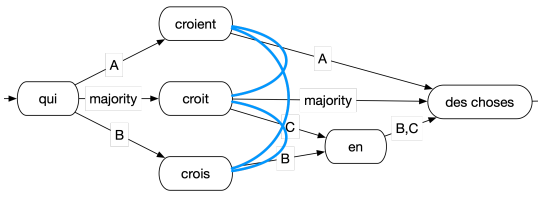
An example of transitivity can be seen in Figure 1. Here, the editor has set relations of type `grammatical` between the two reading pairs ('croit', 'croient') and ('croit', 'crois'); since all three readings are conjugations of the verbe 'croire', they all have the same relationship to each other as variants and the third relation has been inferred by the model.

Transitivity alone, however, is not sufficient for correct automatic deduction of relations between readings. Consider the situation in Figure 2; if the editor were to set a `grammatical` relation between 'suggestiones' and 'suggestionem', and between 'suggestiones' and 'sug-

Figure 1: Relation transitivity: conjugations of 'croire'

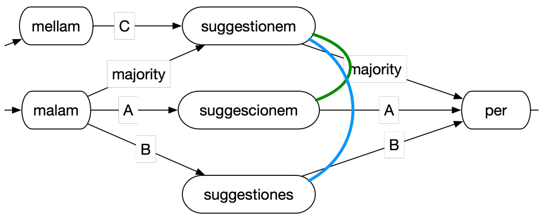


(a) A set of grammatical variants. Two relations are set by the editor...

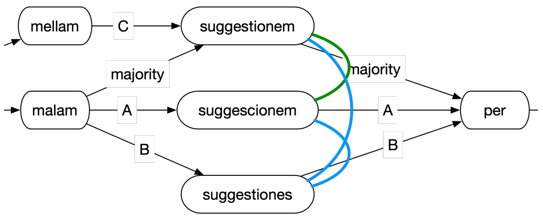


(b) ...the third is inferred based on transitivity rules.

Figure 2: Relation implication: variants of 'suggestio'



(a) A mixed pair of variant relations. Two relations are set by the editor...



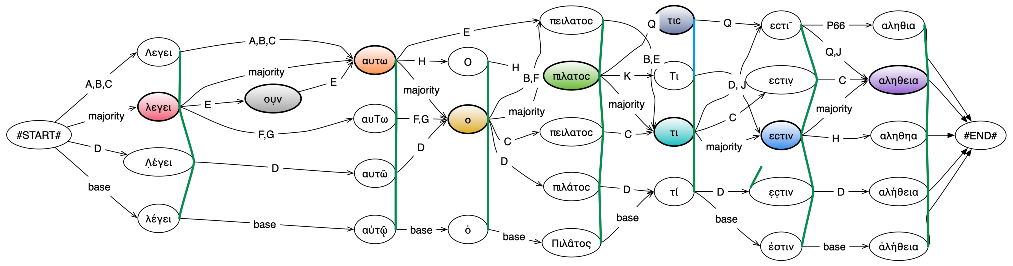
(b) ...the third is inferred based on transitivity and implication rules.

gescionem', the logic of transitivity would give us a `grammatical` relation between 'suggestionem' and 'suggescionem', when an editor would rather see a `spelling` relation. Here the concept of *implication*, represented by our `bindlevel` setting, comes into play: wherever a transitive relation (e.g. `grammatical`) is set between two readings, and one of these readings has a more closely-bound (e.g. `spelling`) relation to a third reading, a more loosely-bound relation between the as-yet unconnected readings will be inferred. This holds true no matter the order in which the differently-typed relations are set; our experience, however, is that for variant locations with very many variants, the best results are achieved by 'working outwards' from the most closely bound readings to the least.

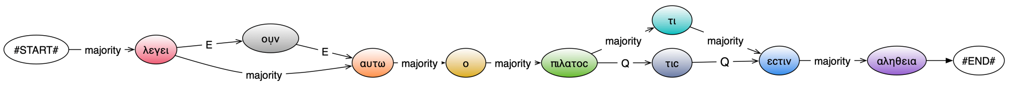
The system of transitivity and implication, simplistic though it is, handles the logical consequences of different levels of normalization and serves thereby as both a practical help and a sanity check for users of the StemmaREST model. A more comprehensive typology of variant readings would be a useful future direction; one possibility would be to construct a graph hierarchy of possible relation types, and use the hierarchy to determine implication rather than the *bindlevel* value.[4]

[4]At least one proposal for a different typology, in the form of an ontology, has been made [11]; implementation of this particular typology in StemmaREST would additionally require the direction of relation edges to be taken into account, as they currently are not.

Figure 3: Normalization of a graph via projection



(a) The original graph; spelling relations are marked in green



(b) The projected graph of spelling normalization; cluster representatives are majority readings

4. Relation-based normalization of a variant graph

Text normalization is a frequent desideratum of scholarly editors, who are usually faced with the conflicting desires to represent the text versions as faithfully as possible on the one hand, and to avoid burdening their readers on the other hand with variation that would be considered trivial.[12, 16, 3] Normalization is also a core issue in stemmatics; the widely-adopted best practices for construction of a stemma depend on distinguishing ‘significant error’ from ‘insignificant’ changes that could be easily reversed by a later copyist.[15, 19, 9]

A StemmaREST variant graph can be normalized by means of its reading relations. A common task, for example, is to omit variations in spelling and punctuation. In order to produce a collation normalized in this way, we first identify the clusters of multiple readings that should be treated as a single reading — that is, the readings that are related to each other with either the ‘spelling’ type or the ‘punctuation’ type. Readings that are part of no such relation then become one-member clusters.

The next step is to create a *graph projection* in which each cluster is represented by one of its members. The choice of representative may be made in any number of ways, e.g. a reading selected at random, the reading present in a majority or plurality of witnesses, or the reading that has been selected as a lemma (i.e. canonical) by the editor.

Each edge in the projection corresponds to an edge in the original graph, replacing the original source and target vertices with their respective cluster representatives. Its properties are calculated as a union of the properties of all edges in the original graph that it represents. An example of normalization is given in Figure 3.

5. Self-consistency of the variant graph: reading rank and reading location

It is not difficult to find examples in printed critical editions of apparatus entries that are mistaken or confusing. Even when a tool such as Classical Text Editor is used, the intention of the editor does not always translate correctly to the underlying data model, as is frequently discovered when the editor attempts to export a TEI XML-encoded version of their edition [4]. The logical notion of variant location is not only key to the clear and error-free construction of an apparatus, but is also a necessary precondition for any sort of computer-assisted stemmatic analysis of the manuscript tradition.

5.1. Calculation of reading rank

It is clear that, if a normalized projection is made of a variant graph, that projection is subject to the same rules provided in section 2: it must itself be a CRDAG whose base witness paths run unbroken from the start reading to the end reading, and the sequence of readings produced for any given witness path must correspond correctly to the editor's view of how the text of the respective manuscript reads. This restriction in turn informs what relations may be created within the graph: a reading relation may only be set if one of the following two conditions holds:

1. The relation type is defined as a colocation, and a graph projection using this relation together with all other colocated relations defined in the graph would result in a valid graph according to our definition.
2. The relation type is NOT defined as a colocation, and a graph projection using this relation together with all other colocated relations defined in the graph would NOT result in a valid graph according to our definition.

We improve the efficiency of our consistency check by defining colocations in terms of *reading rank*. One can speak of ranking the vertices of a DAG according to their sequence in the paths defined for that graph. A typical algorithm for this is

$$K_n = 1 + \max\left(K_p\right),$$

where K_n is the rank of vertex n and K_p is the rank of the source vertex of an incoming edge to n.

In our variant graph, we understand reading rank to be synonymous with variant location. That is, all readings with the same rank can be understood to be at the same location in the text, and thus variants of each other. Colocation in the sense of reading relations, then, is strictly equivalent to identical node rank of the related readings. For this we slightly modify our formula for calculating rank:

$$K_n = \max\left(\max\left(K_c\right), 1 + \max\left(K_p\right)\right)$$

where K_c is the rank of a vertex connected by a colocation relation to n. An implementation of our ranking algorithm is given in Appendix A.2.

5.2. Use of reading rank as a constraint

Once the reading rank is calculated for all readings in a variant graph, it may be used to set constraints on other graph operations. Perhaps the most crucial of these, from the point of view of avoiding a corrupted collation, is as a constraint for setting relations between readings. For any proposed reading relation between readings a and b, we first check whether the relation is a colocation and $K_a = K_b$; if so, the relation may be allowed without further calculation. If this condition does not hold, then a full projection (that is, a projection that takes into account all colocated relations) must be made for all readings where $K_a \leq K_r \leq K_b$ and the graph must be searched for the existence of a path between the representative vertices of a and b.

The other primary use of reading rank as a constraint arises when the editor is ready to establish the text. One of the primary purposes of almost any critical edition of a text is to present a single version that the editor considers the best representative of the text. This version is often known as the *lemma text*. In terms of the variant graph, then, the lemma text is modelled as a single path which obeys the same rules as a witness base path: that is, it runs without interruption through some set of readings from start to end. An additional constraint for the lemma text path is that its creation may not alter the rank of any of its readings. When the editor comes to set a lemma text through StemmaREST, this constraint will be enforced to ensure that there is never more than one lemma reading at any given rank in the variant graph, and that the lemma text respects the order of the existing ranks. In this way the consistency of the eventual edition and its critical apparatus is ensured.

6. Conclusion

The idea of a variant graph is now around fifteen years old. Its properties and its use has been described more or less formally by those who have adopted it, but the implications have often been left to the reader's intuition. Over the course of developing the StemmaREST repository, we have encountered many more consequences, pitfalls, implications, and inadvertent production of corrupted collations than we have yet seen described in any of the existing literature; all of these situations have led to a much more thorough and explicit definition of what a variant graph is. We present that definition here with the hope that it will be of service to digital textual scholarship.

Acknowledgments

The StemmaREST code was begun in the spring semester of 2015 by a team of students in the class 'Software Engineering' under the supervision of Prof. Thomas Studer of the University of Bern; its further development was made possible in the framework of the SNSF-funded project 'The *Chronicle* of Matthew of Edessa Online' (#159433) and through support from the University of Vienna. Further contributors to the source code have included Julia Damerow, Sascha Kaufmann, and Tamim Ziai.

References

[1] T. L. Andrews. “Analysis of variation significance in artificial traditions using Stemmaweb”. In: *Digital Scholarship in the Humanities* 31.3 (2016), pp. 523–539. DOI: 10.1093/llc/fqu072.

[2] T. L. Andrews. “Graph schema validation at last? Revisiting the Stemmarest data model with Neo4J and SHACL”. In: *ADHO Digital Humanities Conference 2023 (DH2023)*. Graz, 2023, pp. 318–319. DOI: 10.5281/zenodo.8107471.

[3] T. L. Andrews. “Transcription and Collation”. In: *Stemmatology in the Digital Age: An Introduction*. Ed. by P. Roelli and M. Buzzoni. Berlin: De Gruyter, 2020, pp. 160–175. DOI: 10.1515/9783110684384-004.

[4] T. L. Andrews. “Where are the tools? The Landscape of Semi-Automated Text Edition”. In: *Digitale Edition in Österreich*. Ed. by R. Bleier and H. W. Klug. Schriften des Instituts für Dokumentologie und Editorik 16. Norderstedt: Books on Demand, 2023, pp. 3–17. URL: https://kups.ub.uni-koeln.de/70445/.

[5] T. L. Andrews and C. Macé. “Beyond the tree of texts: Building an empirical model of scribal variation through graph analysis of texts and stemmata”. In: *Literary and Linguistic Computing* 28.4 (2013), pp. 504–521. DOI: 10.1093/llc/fqt032.

[6] T. L. Andrews and J. J. Van Zundert. “An Interactive Interface for Text Variant Graph Models”. In: *Digital Humanities 2013: Conference Abstracts*. Lincoln, NE, 2013, pp. 89–91. URL: http://dh2013.unl.edu/abstracts/ab-379.html.

[7] P. Baret, C. Macé, and P. Robinson. “Testing Methods on an Artificially Created Textual Tradition”. In: *The Evolution of Texts: Confronting Stemmatological and Genetical Methods*. Ed. by C. Macé, P. Baret, A. Bozzi, and L. Cignoni. Linguistica computazionale Xxiv–xxv. Pisa; Rome: Istituti Editoriali e Poligrafici Internazionali, 2006, pp. 255–283.

[8] M. Burghart. “The TEI Critical Apparatus Toolbox: Empowering Textual Scholars through Display, Control, and Comparison Features”. In: *Journal of the Text Encoding Initiative* Issue 10 (2016). DOI: 10.4000/jtei.1520.

[9] P. Chiesa. “Principles and practice”. In: *Handbook of Stemmatology: History, Methodology, Digital Approaches*. Ed. by P. Roelli and O. E. Haugen. De Gruyter, 2020, pp. 74–87. DOI: 10.1515/9783110684384-003.

[10] R. H. Dekker and G. Middell. “Computer-supported collation with CollateX: Managing Textual Variance in an Environment with Varying Requirements”. In: *Supporting Digital Humanities*. Copenhagen, 2011, pp. 1–7. URL: https://pure.knaw.nl/ws/portalfiles/portal/799786159/Computer%5C%5Fsupported%5C%5Fcollation%5C%5Fwith%5C%5FCollateX%5C%5Fhaentjens%5C%5Fdekker%5C%5Fmiddell.pdf.

[11] F. Giovannetti. “The Critical Apparatus Ontology (CAO): Modelling the TEI Critical Apparatus as a Knowledge Graph”. In: *Graph Data Models and Semantic Web Technologies in Scholarly Digital Editing*. Ed. by E. Spadini, F. Tomasi, and G. Vogeler. Schriften des Instituts für Dokumentologie und Editorik 15. Norderstedt: BoD, 2021, pp. 125–139. URL: https://kups.ub.uni-koeln.de/55230/.

[12] D. C. Greetham. *Textual Scholarship: an Introduction*. New York: Garland Publishing, 1992.

[13] R. Haentjens Dekker, D. Van Hulle, G. Middell, V. Neyt, and J. van Zundert. "Computer-supported collation of modern manuscripts: CollateX and the Beckett Digital Manuscript Project". In: *Literary and Linguistic Computing* 30.3 (2015), pp. 452–470. DOI: 10.1093/llc/fqu007.

[14] S. Jänicke, A. Geßner, G. Franzini, M. Terras, S. Mahony, and G. Scheuermann. "TRAViz: A Visualization for Variant Graphs". In: *Digital Scholarship in the Humanities* 30.suppl_1 (2015), pp. i83–i99. DOI: 10.1093/llc/fqv049.

[15] P. Maas. *Textkritik*. 2nd edition. Leipzig: Teubner, 1950.

[16] L. D. Reynolds and N. G. Wilson. *Scribes and Scholars: A Guide to the Transmission of Greek and Latin Literature*. Oxford: Clarendon Press, 1991.

[17] D. Schmidt and R. Colomb. "A data structure for representing multi-version texts online". In: *International Journal of Human-Computer Studies* 67 (2009), pp. 497–514.

[18] TEI Consortium. "12.2.2. The Double End-Point Attachment Method". In: *Guidelines for Electronic Text Encoding and Interchange. Version 4.6.0.* 2023, pp. 453–455. URL: http://www.tei-c.org/p5/.

[19] S. Timpanaro. *The genesis of Lachmann's method*. Trans. by G. W. Most. Chicago and London: University of Chicago Press, 2005.

[20] P. Trovato. *Everything You Always Wanted to Know about Lachmann's Method*. Trans. by F. Poole. Padova: libreriauniversitaria.it Edizioni, 2014.

A. Java implementations for selected algorithms

A.1. Traversal for a selected witness path

Let `sigil` be a string whose value is the sigil for a `Witness` node; let `layer` be a string whose value is a named witness layer; let `db` be the `GraphDatabaseService` object for querying the database.

```java
public class WitnessPath {
/* ... */
    public Evaluator getEvalForWitness () {
        return path -> {

            if (path.length() == 0) {
                return Evaluation.EXCLUDE_AND_CONTINUE;
            }
            // Find all relevant alternative paths out from last node;
            // there should be zero or one.
            Relationship correct = null;
            for (String layer : alternative) {
                Node priorNode = path.lastRelationship().getStartNode();
                for (Relationship r :
↪   priorNode.getRelationships(Direction.OUTGOING, seqType))
                    if (r.hasProperty(layer) &&
↪   witnessIn(r.getProperty(layer)))
                        if (correct != null)
                            // There is more than one relevant path; cut the
↪   tree off.
                            return Evaluation.EXCLUDE_AND_PRUNE;
                        else
                            correct = r;
            }
            // There is one relevant path; return depending on whether
            // that path is us.
            if (correct != null)
                return correct.equals(path.lastRelationship())
                        ? Evaluation.INCLUDE_AND_CONTINUE :
↪   Evaluation.EXCLUDE_AND_PRUNE;

            // Follow the main path in the absence of an alternative
            if (path.lastRelationship().hasProperty("witnesses")
                    &&
↪   witnessIn(path.lastRelationship().getProperty("witnesses")))
                return Evaluation.INCLUDE_AND_CONTINUE;

            return Evaluation.EXCLUDE_AND_PRUNE;
        };
    }
    private Boolean witnessIn (Object property) {
        String[] arr = (String []) property;
        for (String str : arr) {
            if (str.equals(sigil)) {
                return true;
            }
        }
        return false;
    }
}

/* ... */
Evaluator e = new WitnessPath(sigil, layer).getEvalForWitness();
db.traversalDescription().depthFirst()
        .relationships(ERelations.SEQUENCE, Direction.OUTGOING)
        .evaluator(e)
        .uniqueness(Uniqueness.RELATIONSHIP_PATH)
        .traverse(startNode)
        .nodes();
```

A.2. Ranking algorithm for readings in a collation

```
private static class RankEvaluate implements Evaluator {

    private final Long rank;

    RankEvaluate(Long stoprank) {
        rank = stoprank;
    }

    @Override
    public Evaluation evaluate(Path path) {
        if (path.endNode().equals(path.startNode()))
            return Evaluation.INCLUDE_AND_CONTINUE;
        Node testNode = path.lastRelationship().getStartNode();
        if (testNode.hasProperty("rank")
                && (Long) testNode.getProperty("rank") >= rank) {
            return Evaluation.INCLUDE_AND_PRUNE;
        } else {
            return Evaluation.INCLUDE_AND_CONTINUE;
        }
    }
}

public static class AlignmentTraverse implements PathExpander {

    private final HashSet<String> includeRelationTypes = new HashSet<>();

    // Walk the graph of sequences and colocated relations
    public AlignmentTraverse(Node referenceNode) throws Exception {
        // Get the colocated types for this node's tradition
        List<RelationTypeModel> rtms =
 ↪  RelationService.ourRelationTypes(referenceNode);
        for (RelationTypeModel rtm : rtms)
            if (rtm.getIs_colocation())
                includeRelationTypes.add(rtm.getName());
    }

    @Override
    public Iterable<Relationship> expand(Path path, BranchState state) {
        return expansion(path, Direction.OUTGOING);
    }

    private Iterable<Relationship> expansion(Path path, Direction dir) {
        ArrayList<Relationship> relevantRelations = new ArrayList<>();
        // Get the sequence relationships
        for (Relationship relationship : path.endNode()
                .getRelationships(dir, ERelations.SEQUENCE,
 ↪  ERelations.LEMMA_TEXT, ERelations.EMENDED))
            relevantRelations.add(relationship);
        // Get the alignment relationships and filter them
        for (Relationship r : path.endNode().getRelationships(Direction.BOTH,
 ↪  ERelations.RELATED)) {
            if
 ↪  (includeRelationTypes.contains(r.getProperty("type").toString()))
                relevantRelations.add(r);
        }
        return relevantRelations;
    }
}


public static Set<Node> recalculateRank (Node startNode, boolean
 ↪  recalculateAll) throws Exception {
    RankCalcEvaluate e = new RankCalcEvaluate(startNode, recalculateAll);
    AlignmentTraverse a = new AlignmentTraverse(startNode);
    GraphDatabaseService db = startNode.getGraphDatabase();

    ResourceIterable<Node> touched = db.traversalDescription().depthFirst()
            .expand(a)
            .evaluator(e)
            .uniqueness(Uniqueness.RELATIONSHIP_GLOBAL)
            .traverse(startNode).nodes();
    // Run the traverser and commit the updated ranks
    Set<Node> changed = new HashSet<>();
    for (Node n : touched.stream().collect(Collectors.toSet())) {
        if (!n.hasProperty("newrank"))
            throw new Exception (String.format("Node %d (%s) traversed but
 ↪  not re-ranked!",
                    n.getId(), n.getProperty("text")));
        Long nr = (Long) n.removeProperty("newrank");
        if (!n.hasProperty("rank") || !n.getProperty("rank").equals(nr)) {
            changed.add(n);
            n.setProperty("rank", nr);
        }
    }
    return changed;
}
```