UNIVERSITY OF LORRAINE

PROJECT REPORT

Variant Analysis of the Ben Sira Manuscripts

Author:
Prunelle Daudré–Treuil,
Mayank Mishra,
Shehenaz Hossain

Supervisor:
J.S. Ray
S. Robert

NLP-M1
IDMC

January 9, 2023

Contents

UNIVERSITY OF LORRAINE

Abstract

IDMC

Masters in Natural Language Processing

Variant Analysis of the Ben Sira Manuscripts

by Prunelle Daudré–Treuil, Mayank Mishra,
Shehenaz Hossain

The current project is based on the variant analysis of the book of Ben-Sira, a jewish wisdom book written in ancient hebrew which has been copied and transformed over the years. This book is included in the bible in some christian denominations. Since it was first compiled the book has been rewritten multiple times, often encompassing in it textual variations or printing errors. The goal of the project is to analyze these variations in certain manuscripts found at Qumrân, Massada and the Cairo Genizah and perform statistical analysis of the data thus obtained. The collection of data from these variants would utilize the software library *Collatex* in python programming. After a successful analysis of the data obtained from the variants, one could understand from a theological point of view the writing patterns of scribes in a particular period of history.

Chapter 1

Introduction

The Bible, the Torah, the Koran and other related religious books are some of the most studied texts to have ever existed. All the translations and transcriptions are heavily debated, and every denomination has its interpretation of what should or should not be in its sacred texts. A good example of that is the Book of Ben Sira or Sirach. This Jewish book of wisdom is part of the deuterocanonical books, a collection of books considered part of the Old Testament in the Catholic Church, the Eastern Orthodox Church, the Oriental Orthodox Churches, and the Assyrian Church of the East canons and regarded as apocryphal by the Protestant denominations.

In this report, we present an annotated bibliography and a plan of action for a variant analysis of different Ben Sira manuscripts written in ancient Hebrew. These different versions have been retrieved recently and show some transcription errors between them, that need to be analysed.

1.1 Biblical texts and the book of Ben Sira

A biblical text is a handwritten excerpt from the Bible. This text may contain small portions from the Bible (like in a scroll) or may include the entirety of the book (as in the case of multilingual books or polyglots). Whatever the form of a biblical text, these serve as important objects of study to understand how these differ from their original text and aid in reconstructing them through textual criticism.

The book of Ben Sira is one such text composed around 180 B.C.E. in Hebrew language by a sage known by the name of Ben Sira. The book of Ben Sira itself has been copied by various scribes over time, leading to multiple versions of the book. These manuscripts have rich textual variations among them and are critical for textual analysis of the original book. In the current project, we focus on this aspect and try to come up with a statistical analysis of these variants using computer-aided collation techniques.

1.2 Classification of textual variations

When studying textual data copied by scribes, one needs to take into consideration the variations induced by such copying. These variations can be genuine typing/writing errors or may be intentionally added during the process of textual transmission. Understanding what kind of variations the text encompasses, can lead to an enhanced textual reconstruction of the original text. In the present section, we discuss some of such variations which are abundant in biblical texts.

1.2.1 Unintentional Variations

Minuses/Removals

These variations are created when a part of the original text is omitted while copying. These omissions may be further categorized on the basis of what feature of the text induced such an omission.

Pluses/Additions

As the name suggests these variations arise when some extra portion is added to the copied text. These may also be further classified by the feature of text which induced such variation and, might be a ditto or doublet.

1.2.2 Intentional Variations

Apart from additions and removals, the copied text might have been entirely changed to something new intentionally. This type of variation in the text is common and numerous examples may be found in the literature. Many of these variations can also be orthographic wherein the same word is written in two different ways but has the same meaning. For example, the word "colour" in the American style and the word "color" in the British style has the same meaning.

A detailed classification of variations possible during textual transmission can be found in Tov, 2012

1.3 Variant analysis and collation

1.3.1 What is variant analysis?

"Variant analysis" can be defined as a process of identifying and analyzing variations or differences in textual data. This can be useful for a variety of purposes, such as identifying errors or inconsistencies in a text, comparing different versions of a text to see how it has changed over time or analyzing how a certain language has evolved.

1.3.2 What is Collation?

"Collation" is a technique that can be used to assist with variant analysis. It involves organizing and comparing different versions of a text in a systematic way, in order to identify and analyze the variations between them. This can be done manually, by physically comparing different copies of a text side by side, or it can be done using software tools that automatically compare and analyze the texts. Using software for collation is sometimes referred to as computer-aided collation and is a prevalent method in today's technological era.

Collation is a very useful technique for carrying out variant analysis. Some specific ways that collation can be used in a variant analysis include:

- Identifying variations in spelling, punctuation, and other aspects of language usage.
- Comparing different versions of a text to see how it has evolved over time.

- Analyzing the relationship between different versions of a text, such as by identifying which version is the original and which are copies or translations.
- Detecting errors or inconsistencies in a text, such as by comparing it to a known correct version.

Parry and Ulrich, 2019 in their book have successfully presented verse-by-verse the variants of the "Hebrew Isaiah" and also conducted its variant analysis. This book is a fine example of using collation to find why certain variants exist and their characteristics.

Chapter 2

Collation using Collatex

The following chapter contains a short "description of" and a "how-to-use" of our primary computational tool Collatex.

2.1 The Gothenburg Model

The Gothenburg model is a roadmap for carrying out computer-aided collation, where-in multiple steps are involved. The model is flexible concerning the user's choices as each step in individuality may be modified without affecting the overall process. This particular attribute is the primary reason why Collatex is designed according to this model. In the next few subsections, we discuss these steps and how Collatex incorporates them.

2.1.1 Tokenization

In written corpora, a token is an entity into which the text might be split for collation. Words excluding white spaces in a text is a simple example of a token. However, a token can be a phrase, a combination of words, lines, contractions etc. depending on the user's choice. Since an inherent challenge in defining the tokens exists, Collatex provides the user with a tokenizer which can perform tokenization at any given level of granularity. Also, Collatex allows the user to combine the tokenizer with specific tools, for comparing the tokens as per specific requirements. This greatly eases the process of tokenization and hence enhances collation results.

2.1.2 Normalization

A token in a handwritten document is usually not identical to its subsequent occurrences and hence the user might want the computer to not distinguish these tokens while making comparisons. For example, in a text, all letters in the words may be converted to small caps before comparing. Normalization refers to this exact process. Apart from the above example, normalization may be done on various levels, again depending on the project at hand. Collatex offers a standard normalization based on case sensitivity, punctuation removal and orthographic dissimilarity. Apart from these, for more complex normalization paradigms, Collatex offers API-based plug-ins for specific requirements.

2.1.3 Alignment

After the tokens in a text have been extracted and normalized, the next step is to align them properly for analysis. Collatex offers multiple alignment algorithms to be utilized per the user's requirements. It also allows the user to visualize the aligned tokens in formats like alignment tables, variant graphs, etc. The alignment algorithms available to be used in Collatex all belong to the class of progressive multiple alignment algorithms as described in Spencer and Howe, 2004

2.1.4 Analysis

After a successful alignment of tokens in the text has been completed, there is still a need for analysis. This need is fuelled by the fact that the computer might not be able to align the tokens as specifically as the user demands. For example in cases where exact matches between multiple variants of the same text are required, we would require analysis. Also in cases where transpositions (pages 170-172 Roelli, 2020) are frequent, analysis of the text aligned by the computer is necessary. Collatex offers near matches at the alignment step but finding exact matches in textual variants is a computationally expensive problem and needs analysis at the user's end.

2.1.5 Visualization

The final step before conducting a statistical analysis of obtained results would be that of visualization. As mentioned earlier, Collatex allows the user to visualize textual data and assess variation in it via output features like alignment tables in HTML formats or variant graphs in an SVG/GraphML format.

2.2 Hands-on with Collatex

In this section, we provide information on how to use Collatex for performing variant analysis of textual data.

2.2.1 Input in Collatex

Collatex accepts input in two formats, namely the "plain-text" format and the "JSON" (Java Script Object Notation) format.

2.2.2 Output in Collatex

Collatex offers its users the flexibility of viewing the results in multiple output formats. One can use the following options,

1. **Alignment Table:** An alignment table as the name suggests is a table wherein Collatex aligns tokens from two variations of a text (known as witnesses) in a tabular format to aid the process of analysis. This is a simple method for emphasizing and visualizing textual variations. Collatex allows the user to view the alignment table either with or without the "near-match" feature. The "near-match" feature triggers Collatex to align tokens which possess near-string equality to each other (based on the Levenshtein distance) and leave blank spaces in the table if no such match is found. Additionally, as described in Camps, Ing, and Spadini, 2019, Collatex may be modified to have specific alignment requirements satisfied.

2. **Variant Graphs:** Visualizing textual data containing variations and overlapping hierarchies has been a challenging task for quite some time unless the variant graphs were introduced (Schmidt and Colomb, 2009). These novel

data structures allow visualization of complex textual variations in a systematic way, surpassing traditional markup languages. Collatex offers the utilization of these data structures for observing textual variations in data either in a simplistic or a detailed format.

3. **Other formats:** Apart from the above two widely utilized output formats, Collatex also allows its users to visualize their outputs in a markup language like XML or JSON.

2.2.3 Examples

Using a plain-text input

- Without near-match feature

Input:

```
from collatex import*
collation = Collation()
collation.add_plain_witness("A", "An␣apple␣a␣day␣keeps␣the␣doctor␣away.")
collation.add_plain_witness("B", "A␣peach␣a␣day␣keeps␣the␣doctor␣away.")
alignment_table = collate(collation)
print(alignment_table)
```

Output:

```
+---+----------+------------------------------+
| A | An apple | a day keeps the doctor away. |
| B | A peach  | a day keeps the doctor away. |
+---+----------+------------------------------+
```

- With near-match feature

Input:

```
from collatex import *
collation = Collation()
collation.add_plain_witness("A", "The␣happy␣boy␣wears␣a␣grey␣coat")
collation.add_plain_witness("B", "The␣boy␣wears␣a␣gray␣coat␣")
alignment_table = collate(collation, near_match = True, segmentation=False)
print(alignment_table)
```

Output:

```
+---+-----+-------+-----+-------+---+------+------+
| A | The | happy | boy | wears | a | grey | coat |
| B | The | -     | boy | wears | a | gray | coat |
+---+-----+-------+-----+-------+---+------+------+
```

- Output in HTML format

Input:

```
from collatex import*
collation = Collation()
collation.add_plain_witness("A", "The␣quick␣brown␣fox␣jumps␣over␣the␣dog.")
collation.add_plain_witness("B", "The␣brown␣fox␣jumps␣over␣the␣lazy␣dog.")
collate(collation, output = "html")
```

Output:

```
A       The     quick    brown fox jumps over the        -        dog.
B       The     -        brown fox jumps over the        lazy     dog.
```

- Output in HTML2 format

  Input:

```
from collatex import*
collation = Collation()
collation.add_plain_witness("A", "The␣quick␣brown␣fox␣jumps␣over␣the␣dog.")
collation.add_plain_witness("B", "The␣brown␣fox␣jumps␣over␣the␣lazy␣dog.")
collate(collation, output = "html2")
```

  Output:

```
                          A                                  B
                        The                                The
                      quick                                  –
brown fox jumps over the          brown fox jumps over the
                          –                               lazy
                       dog.                               dog.
```

Using a JSON input

Input:

```
import json
from collatex import *
collation = Collation()
json_input = """{
    "witnesses": [
        {
            "id": "A",
            "tokens": [
                {
                    "t": "The ",
                    "n": "The"
                },
                {
                    "t": "quick ",
                    "n": "quick"
                },
                {
                    "t": "brown ",
                    "n": "brown"
                },
                {
                    "t": "fox ",
                    "n": "fox"
                },
                {
                    "t": "jumps ",
                    "n": "jumps"
                },
                {
                    "t": "over ",
                    "n": "over"
                },
                {
                    "t": "the ",
                    "n": "the"
                },
                {
                    "t": "dog",
                    "n": "dog"
                },
                {
                    "t": ".",
                    "n": "."
                }
            ]
        },
        {
            "id": "B",
            "tokens": [
                {
                    "t": "The ",
                    "n": "The"
                },
                {
                    "t": "brown ",
                    "n": "brown"
                },
                {
                    "t": "fox ",
                    "n": "fox"
                },
                {
                    "t": "jumps ",
                    "n": "jumps"
                },
                {
                    "t": "over ",
                    "n": "over"
                },
                {
                    "t": "the ",
                    "n": "the"
                },
                {
                    "t": "lazy ",
                    "n": "lazy"
                },
                {
                    "t": "dog",
                    "n": "dog"
                },
                {
                    "t": ".",
                    "n": "."
                }
            ]
        }
    ]
}"""
print(collate(json.loads(json_input)))
```

Output:

```
+---+-----+-------+--------------------------+------+------+
| A | The | quick | brown fox jumps over the | -    | dog. |
| B | The | -     | brown fox jumps over the | lazy | dog. |
+---+-----+-------+--------------------------+------+------+
```

Using Variant graph output

For the input given below we can obtain different outputs depending on whether we used "svg_simple" or "svg" in the output field of the collate function.

Input:

```
from collatex import *
import json
collation = Collation()
json_input = """{
    "witnesses": [
        {
            "id": "A",
            "tokens": [
                {
                    "t": "The ",
                    "n": "The"
                },
                {
                    "t": "beautiful",
                    "n": "beautiful"
                },
                {
                    "t": "color",
                    "n": "color"
                }
            ]
        },
        {
            "id": "B",
            "tokens": [
                {
                    "t": "The ",
                    "n": "The"
                },
                {
                    "t": "beautiful",
                    "n": "beautiful"
                },
                {
                    "t": "colour",
                    "n": "colour"
                }
            ]
        },
        {
            "id": "C",
            "tokens": [
                {
                    "t": "The ",
                    "n": "The"
                },
                {
                    "t": "beautiful",
                    "n": "beautiful"
                },
                {
                    "t": "flower",
                    "n": "flower"
                }
            ]
        }
    ]
}"""
collate(json.loads(json_input), output="svg_simple/svg")
```

Output:

- Simple variant graph

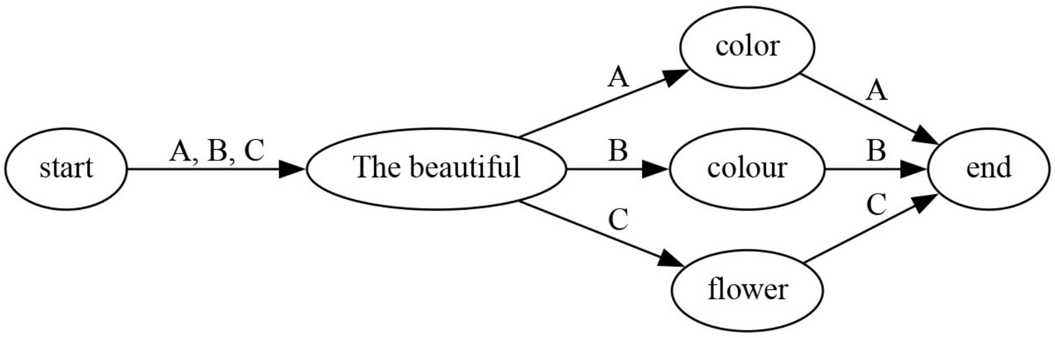


FIGURE 2.1: Simple variant graph

- Complete variant graph

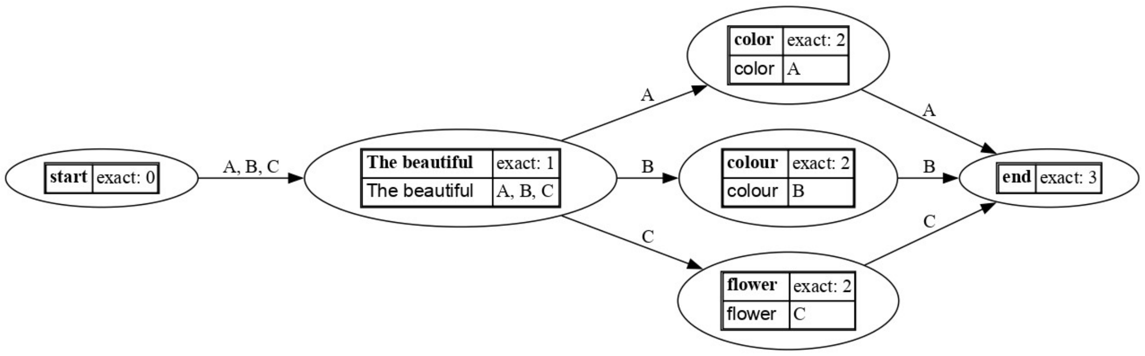


FIGURE 2.2: Complete variant graph

Chapter 3

Plan of action and the stepping stones

3.1 Plan of Action

To compare the variations in the texts of Ben Sira, we intend to use the Collatex collation tool. However, before we can directly input the different variations of the verses, text pre-processing is essential to consolidate which verses are going to be compared. Briefly, the steps planned to accomplish the task can be listed as:

- Pre-process the XML files
- Parse the XML file to load the data
- Gathering the data for the collation
- Collation and Analysis

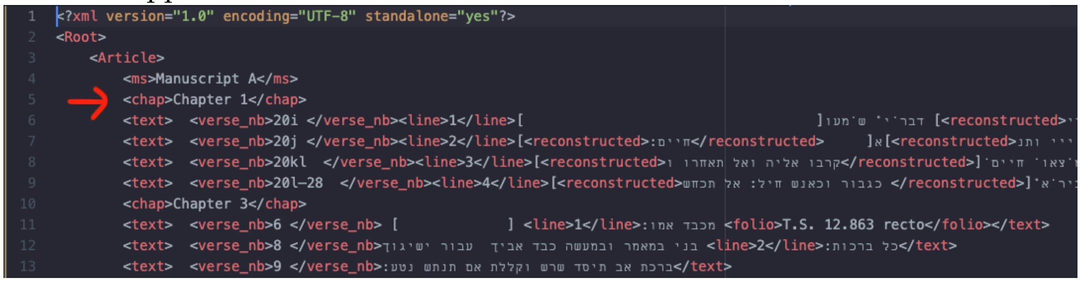
In the upcoming sections, we describe our plan of action in depth and thereafter we will have a quick overview of the current progress achieved so far.

3.1.1 Pre-processing

For the project, we have been provided with the data in the form of XML files along with their Document Type Definition (DTD) files. The Ben Sira texts have been provided as an XML file with 9 manuscripts.

As the first step, we start by looking at the document type definition. For start, the manuscripts are enclosed by `<ms>` tags and each chapter is demarcated by the tag `<chap>`. Inside each chapter, the verses are enclosed in `<text>` tag with additional information such as verse number being enclosed using `<verse_nb>` tag.

As of now, we are doing two main processing steps mainly to help the XML parser associate the verses with the chapters properly. To illustrate the problem we put a small snippet of the XML below.

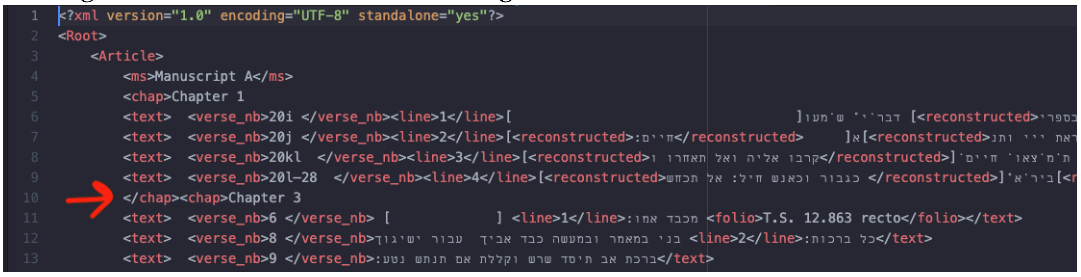


```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Root>
    <Article>
        <ms>Manuscript A</ms>
        <chap>Chapter 1</chap>
        <text>  <verse_nb>20i </verse_nb><line>1</line>[                    ]ש׳מעו דב׳רי׳ [<reconstructed>
        <text>  <verse_nb>20j </verse_nb><line>2</line>[<reconstructed>:חיים</reconstructed>     ]א[<reconstructed>ותנ ׳׳׳
        <text>  <verse_nb>20kl  </verse_nb><line>3</line>[<reconstructed>ו תאחרו ואל קרבו אליה</reconstructed>]צאו׳ חיים׳
        <text>  <verse_nb>20l-28  </verse_nb><line>4</line>[<reconstructed>אל תכחש :כגבור וכאנש חיל </reconstructed>]ירא׳
        <chap>Chapter 3</chap>
        <text>  <verse_nb>6 </verse_nb> [              ] <line>1</line>מכבד אמו: <folio>T.S. 12.863 recto</folio></text>
        <text>  <verse_nb>8 </verse_nb>בני במאמר ובמעשה כבד אביך  עבור ישיגוך <line>2</line>כל ברכות:</text>
        <text>  <verse_nb>9 </verse_nb>ברכת אב תיסד שרש וקללת אם תנתש נטע:</text>
```

Here the `<chap>` tag does not enclose the verses that are included in the chapter as the `</chap>` tag closes immediately and the XML parser is not able to correctly relate the verses to the chapter. To fix this we have re-processed the XML files containing the `<chap>` tags to make it easier for the parser.

Processing for manuscripts with `<chap>` tag

To fix the issues of associating the verses correctly with their chapters we are pre-processing the XML files to look something like below

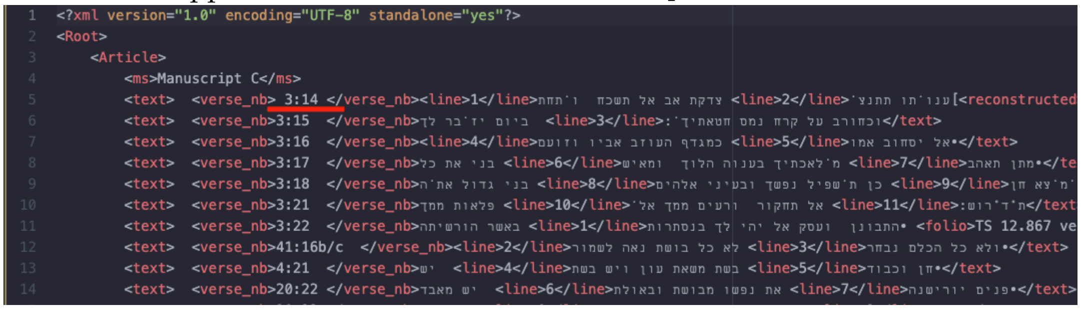


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Root>
    <Article>
        <ms>Manuscript A</ms>
        <chap>Chapter 1
        <text>  <verse_nb>20i </verse_nb><line>1</line>[                              ]ש'מעון דבר'י'י [<reconstructed>בספרי
        <text>  <verse_nb>20j </verse_nb><line>2</line>[<reconstructed>חיים:</reconstructed>     ]א[<reconstructed>ראת ייי ונת
        <text>  <verse_nb>20kl  </verse_nb><line>3</line>[<reconstructed>קרבו אליה ואל תאחרו ו</reconstructed>]ת'מ'צאו' חיים'
        <text>  <verse_nb>20l–28  </verse_nb><line>4</line>[<reconstructed>כגבור וכאנש חיל: אל תכחש</reconstructed>]ב'י'ר'א'[<r
        </chap><chap>Chapter 3
        <text>  <verse_nb>6 </verse_nb> [            ] <line>1</line>מכבד אמו: <folio>T.S. 12.863 recto</folio></text>
        <text>  <verse_nb>8 </verse_nb>בני במאמר ובמעשה כבד אביך  עבור ישיגוך <line>2</line>כל ברכות:</text>
        <text>  <verse_nb>9 </verse_nb>ברכת אב תיסד שרש וקללת אם תנתש נטע:</text>
```

We mainly relocate the closing chapter tags i.e. `</chap>` before the beginning of the next chapter. By doing so, the XML parser can easily associate the verses with each chapter correctly.

Processing for manuscripts without `<chap>` tag

Out of the 9 manuscripts given, one of the manuscripts i.e `Manuscript C` does not demarcate the chapter beginning and end well. We believe in this manuscript the chapter numbers are encoded with the verse numbers. To illustrate this we take a look at a small snippet of XML code from the `Manuscript C` below.



```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Root>
    <Article>
        <ms>Manuscript C</ms>
        <text>  <verse_nb> 3:14 </verse_nb><line>1</line>צדקת אב אל תשכח  ו'תחת <line>2</line>עונ'ו'ת ת'נצ'[<reconstructed
        <text>  <verse_nb>3:15  </verse_nb>ביום יז'כר לך  <line>3</line>וכחורב על קרח נמס חטאתיך:</text>
        <text>  <verse_nb>3:16  </verse_nb><line>4</line>כמגדף העוזב אביו וזועם <line>5</line>לא• יסחוב אמו</text>
        <text>  <verse_nb>3:17  </verse_nb>בני את כל <line>6</line>מ'לאכתיך בענוה הלוך  ומאיש <line>7</line>מתן• תאהב</te
        <text>  <verse_nb>3:18  </verse_nb>בני גדול את ה' <line>8</line>כן ת'שפיל נפשך ובעיני אלהים <line>9</line>ת'מ'צא חן
        <text>  <verse_nb>3:21  </verse_nb>פלאות ממך <line>10</line>אל תחקור  ורעים ממך אל' <line>11</line>ת'ד'רוש:</text
        <text>  <verse_nb>3:22  </verse_nb>באשר הורשית <line>1</line>התבונן•  ועסק אל יהי לך בנסתרות <folio>TS 12.867 ve
        <text>  <verse_nb>41:16b/c  </verse_nb><line>2</line>לא כל בושת נאה לשמור <line>3</line>ולא• כל הכלם נבחר</text>
        <text>  <verse_nb>4:21  </verse_nb>יש  <line>4</line>בשת משאת עון ויש בשת <line>5</line>חן• וכבוד</text>
        <text>  <verse_nb>20:22 </verse_nb>יש מאבד  <line>6</line>את נפשו מבושת ובאולת <line>7</line>פנים• יורישנה</text>
```

We believe that verse number 3:14 denotes verse number 14 of Chapter 3. In order to process this file we intend to pre-process it to make it look similar to the other XML files so that there is uniformity in the data before we move onto the data loading stage.

3.1.2 Parsing the XML data

Before using any of the collation tools for the comparison of the textual variants of Ben Sira, the data from the XML file needs to be processed to extract the verses appropriately. In this section, we discuss the information we extract from the tags to process it at a later stage.

In order to parse the XML data we are currently using a python library named *BeautifulSoup* which is commonly used for parsing HTML and XML tags. In the background, several other python libraries like *ElementTree* and XML to JSON converters like *xmltodict* were also tried. But it was decided to stick with *Beautiful Soup*.

For now, we extract the information from the following tags in XML files:

- Manuscript File Name

- Chapter Number
- Verse Number
- Verse text

We keep the information extracted from the tags in the form of a dictionary in python so that it is easier to locate the verse by its manuscript, chapter and verse number. For the verses which do not have a verse number, we keep them away for now, until and unless we find an effective way to match them with the other verses. Apart from the verses, we also extract the chapter present in each manuscript. This is to assist in matching the chapters across the manuscript.

While we extract information from the aforementioned tags, we ignore the text enclosed from the tags below as we are yet to understand the significance of the tags in terms of text version comparison.

- Folio
- Reconstructed
- Margin
- Line Number
- Superscript
- Supralinear
- Greek

3.1.3 Gathering the data

To prepare the data to perform the actual verse comparison using Collatex, we need to prepare the set of verses or also known as witnesses for the actual comparison. We first compare the different manuscripts to find the overlapping chapters between them. After we have found the common chapters between them, we find the common verses between the two chapters by comparing the verse numbers. This step for now gives us two sets of verses for comparison.

However, we also plan to expand this comparison as there are certain chapters which occur in more than two manuscripts. In these types of cases, there are 3 or more verses for variation comparison.

3.1.4 Collation

After we have successfully built up the witnesses or the texts to be compared, we input them into the Collatex tools for analysis. We use the Collatex python library to load the witnesses and thereafter the Collatex tool outputs the comparison in the form of an alignment table.

During analysis, we mainly compare if the alignment table produced by the analysis tools is able to match up similar words and able to highlight the differences well. Currently, a more rigorous study needs to be done on whether a more objective way can be used to check the appropriateness of the alignment table other than manual subjective assessment.

3.2 The stepping stones

3.2.1 Finding common chapters across manuscripts

In terms of progress, we have been able to segregate the matching chapters across the manuscripts and keep them in a dictionary along with their matching manuscripts. However, our program in its current form is unable to match common chapters across more than 2 manuscripts. This limitation we intend to have it resolved in the future.

The current python output of the dictionary variable containing the matching chapters is attached below:

```
{'ms_a_new.xml': {'ms_b_new.xml': [['Chapter 10', 'Siracide 10'],
                                   ['Chapter 11', 'Siracide 11'],
                                   ['Chapter 15', 'Siracide 15'],
                                   ['Chapter 16', 'Siracide 16']],
                  'ms_d_new.xml': [['Chapter 7', 'Siracide 7'],
                                   ['Chapter 8', 'Siracide 8']],
                  'ms_e_new.xml': [],
                  'ms_f_new.xml': []},
 'ms_b_new.xml': {'ms_d_new.xml': [['Siracide 36', 'Siracide 36'],
                                   ['Siracide 37', 'Siracide 37'],
                                   ['Siracide 38', 'Chapitre 38']],
                  'ms_e_new.xml': [['Siracide 32', 'Chapter 32'],
                                   ['Siracide 33', 'Chapter 33']],
                  'ms_f_new.xml': [['Siracide 31', 'Siracide 31'],
                                   ['Siracide 32', 'chapter 32'],
                                   ['Siracide 33', 'Chapter 33']]},
 'ms_d_new.xml': {'ms_e_new.xml': [],
                 'ms_f_new.xml': []},
 'ms_e_new.xml': {'ms_f_new.xml': [['Chapter 32', 'chapter 32'],
                                   ['Chapter 33', 'Chapter 33']]}}
```

Not all the manuscripts have been processed so far and some of the manuscripts are still left for the future.

For storing the verses we use another python dictionary which is of the structure `verses[manuscript][chapter_number][verse_number]`. The limitation of this way of storing the verses is that the verses that do not have a verse number in the original manuscript are skipped as they cannot be assigned a verse number key. We intend to solve this limitation in the upcoming months by finding a different way of storing the verses effectively.

3.2.2 Preliminary collation results

In this section, we describe some of the collation results that we have obtained so far. We will be showing a few examples of the collation output, however, we do not analyse the collation output in detail at this stage.

Instance 1

From the matching lists, we compare verse number 23 from manuscript A and B from chapter 10. The raw verses are as follows. The collation output is shown in Fig. A.1.

```
Manuscript A:  אין לבזות דׄלׄ מׄשׄכיל     ואין לכבד כׄל איש []כׄם:
Manuscript B:  אין לבזות דל משכיל     ואין לכבד כל איש חמס:
```

Instance 2

In instance 2 we compare the verses from manuscripts A and B again. However this time from chapter 11 verse number 4. The verses are outlined below with the collation output described in Fig. A.2

```
Manuscript A:  בׄעׄטׄה א[]ר אל תהׄתׄל     ואל תקלס במרירי יום:
Manuscript B:  במעוטף בגדים אל תתפאר     ואל תקלס במרירי יום
```

Instance 3

For the third instance, we compare verses between manuscripts A and D from chapter 7 with verse number 31. The output is shown in Fig. A.3 with the verses shown below:

```
Manuscript A:  כׄבד אל והדׄר כהן     ות[]לקם כאשר צׄוותה:
Manuscript B:  :[ ]כׄהׄן     ותן חׄלקׄם כאשר צׄויתה:
```

3.2.3 Current outlook

So far we have been able to organise the different verses from the Ben Sira into an appropriate form to be loaded into the Collatex tool. In its current format, the verses can still be fed into the Collatex tool and the output is generated with some inconsistencies. However, we plan to analyse the output completely in the upcoming months. Further, there are certain limitations in the way we are extracting the verses as there are stray square brackets left in the extracted verses, which might lead to inconsistent verse comparison across the manuscripts. Further, we need to work on generating witnesses where the same verse is mentioned in more than two manuscripts. Finally, work remains to be done in analysing the collation output more effectively, as without the knowledge of the Hebrew language this task becomes difficult.

Chapter 4

Conclusion

4.1 Summary

This report contains a presentation of the different methods and tools needed in our current project. As elaborated in the previous part, the main part of the project is the use of the python library Collatex to handle the variant analysis. This library, using the Gothenburg Model, will be used to tokenize and normalize the text of the manuscripts. Then the collated text will be aligned in order to bring out the different variations. Afterwards, the different variations between the two manuscripts, intentional or not, will be ready for visualization and analysis.

In addition, the library Dicta is planned to be used in order to be able to treat the Hebrew alphabet. If needed, some more common NLP libraries, like Spacy will be used as a backup plan.

4.2 Further work

It is interesting to note that the goal of this project is not just to compare the manuscripts of the Book of Ben Sira but also to develop a standardized library allowing other users in the field of textual reconstruction, to apply variant analysis on other manuscripts. It should answer questions like- what are the types of errors present in the text? If the error is on the scale of some letters or some words. For the substitutions of a letter, which letters were substituted? At which frequency were the letters substituted? Can some general rules be found? For the errors concerning words, is the words swapped with synonyms or with random words? Have some words been added, or on the contrary, removed? Or maybe some words have been moved or transposed? Were the meaning of the sentences lost? etc. If the library thus created helps the users coherently answer the above questions, then the project would be considered a success.

Appendix A

Images for collation results

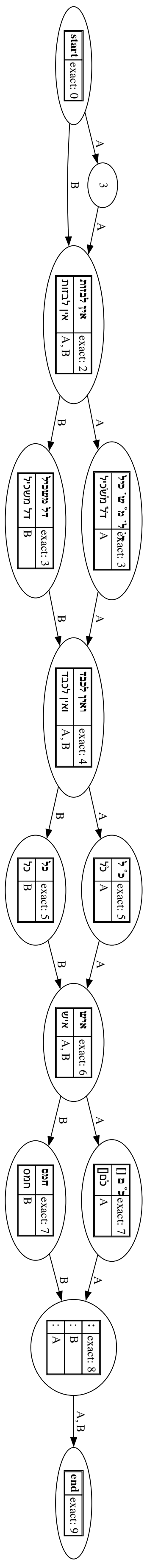


FIGURE A.1: Collation output for Instance 1

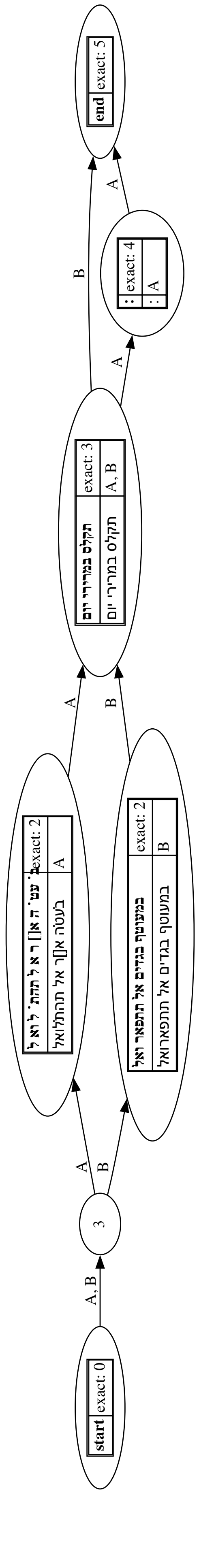


FIGURE A.2: Collation output for Instance 2

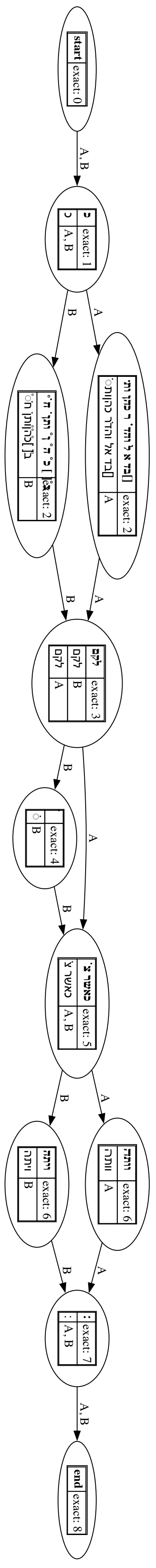


FIGURE A.3: Collation output for Instance 3

Bibliography

Andrews, Tara Lee and Caroline Macé (2013). “Beyond the tree of texts: Building an empirical model of scribal variation through graph analysis of texts and stemmata”. In: *Lit. Linguistic Comput.* 28, pp. 504–521.

Baden, Christian, Neta Kligler-Vilenchik, and Moran Yarchi (2020). “Hybrid Content Analysis: Toward a Strategy for the Theory-driven, Computer-assisted Classification of Large Text Corpora”. In: *Communication Methods and Measures* 14.3, pp. 165–183. DOI: 10.1080/19312458.2020.1803247. eprint: https://doi.org/10.1080/19312458.2020.1803247. URL: https://doi.org/10.1080/19312458.2020.1803247.

Camps, Jean-Baptiste, Lucence Ing, and Elena Spadini (2019). “Collating Medieval Vernacular Texts. Aligning Witnesses, Classifying Variants”. In: *DH2019 Digital Humanities Conference 2019*.

Parry, D.W. and E. Ulrich (2019). *Exploring the Isaiah Scrolls and Their Textual Variants*. Supplements to the Textual His. Brill. ISBN: 9789004410596. URL: https://books.google.fr/books?id=Su0JxgEACAAJ.

Roelli, P. (2020). *Handbook of Stemmatology: History, Methodology, Digital Approaches*. De Gruyter Reference. De Gruyter. ISBN: 9783110684391. URL: https://books.google.fr/books?id=W47_DwAAQBAJ.

Schmidt, Desmond and Robert Colomb (2009). “A data structure for representing multi-version texts online”. In: *International Journal of Human-Computer Studies* 67.6, pp. 497–514. ISSN: 1071-5819. DOI: https://doi.org/10.1016/j.ijhcs.2009.02.001. URL: https://www.sciencedirect.com/science/article/pii/S1071581909000214.

Spencer, Matthew and Christopher J. Howe (2004). “Article: Collating Texts Using Progressive Multiple Alignment”. In: *Computers and the Humanities* 38.3, pp. 253–270. ISSN: 00104817. URL: http://www.jstor.org/stable/30204940 (visited on 12/31/2022).

Tov, E. (2012). *Textual Criticism of the Hebrew Bible*. G - Reference, Information and Interdisciplinary Subjects Series. Fortress. ISBN: 9781451403299. URL: https://books.google.fr/books?id=5Sh7dBDD7ykC.

Wolfart, H Christoph and Francis Pardo (1979). “Computer-aided philology and algorithmic linguistics: a case study”. In: *International Journal of American Linguistics* 45.2, pp. 107–122.