Tutorial accompanying the study "Computer-Assisted Language Comparison: State of the Art"

Wu, Mei-Shin; Schweikhard, Nathanael E.; Bodt, Timothaeus A.; Hill, Nathan W.; List, Johann-Mattis

This tutorial supplements the study "Computer-Assisted Language Comparison: State of the Art". In this tutorial, we explain in detail, how our workflow can be tested and applied.

The workflow consists of several Python libraries that interact, one producing the data that can be used by the other. Since the data is available in different stages, each stages allows us to intervene by correcting errors manually that were made by the automated approach.

For users who are interested in testing our workflow on their local machine or further applying it in their own research, some basic knowledge of the Python programming language and the commandline will be required. All the software offered here is available in form of free software. For more information on LingPy, the main programming library used here, we recommend users to check the tutorial accompanying the study "Sequence comparison in computational historical linguistics (https://academic.oup.com/jole/article/3/2/130/5050100)" by List et al. (2018)[1].

1. Code Ocean Capsule

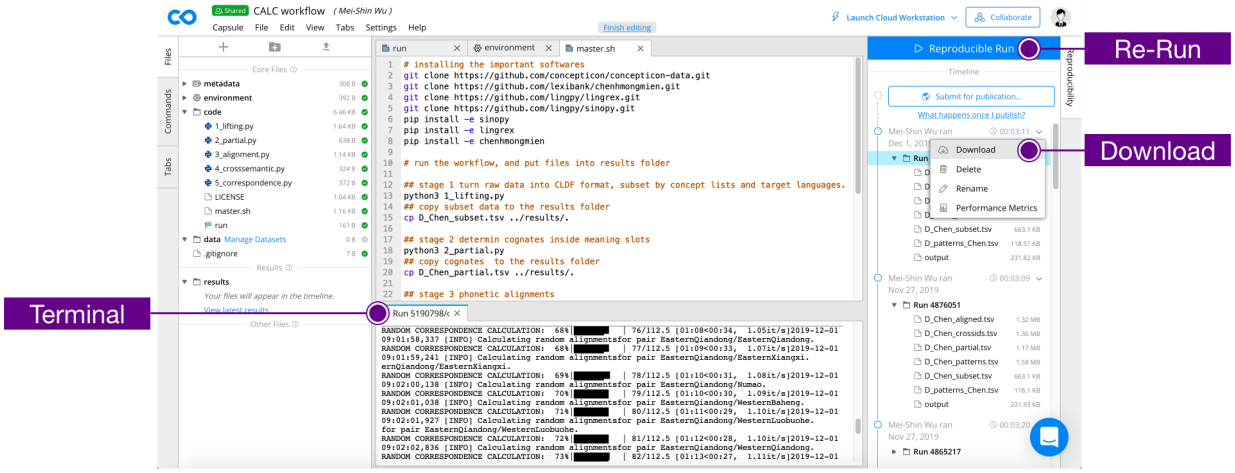
In order to facilitate it for users to quickly test our workflows without installing the software, we have set up a Code Ocean Capsule which users can use to run the code remotely. Code Ocean is an open access platform which enables researchers to reproduce their or others' experiments. For a detailed introduction to the Code Ocean platform, please refer to the website (https://codeocean.com/). To see how our experiments can be run from within the Code Ocean Capsule, follow the following steps:

a) Navigate to the capsule: https://codeocean.com/capsule/8178287/tree/v2

b) Press the "Re-Run" button to reproduce the results.

c) View the progression in the "Terminal" panel.

d) Download all results and unzip the .zip file for further inspection on EDICTOR.



The following files can be found in the downloaded file:

File	Stage	Section
D_Chen_subset.tsv	From raw data to tokenized data	3.1
D_Chen_partial.tsv	From Tokenized Data to Cognate Sets	3.2

File	Stage	Section
D_Chen_aligned.tsv	From Cognate Sets to Alignments	3.3
D_Chen_crossids.tsv	From Alignments to Cross-Semantic Cognates	3.4
D_Chen_patterns.tsv	From Cross-Semantic Cognates to Sound Correspondence	3.5
D_Chen_distance.dst	Validation	4.2, 4.3
D_Chen_tree.tre	Validation	4.2, 4.3

2. Installation Instructions

We assume that users who are interested in running the workflow on their local machine are familiar with the essentials of command-line operations and system administration on either Unix-like systems (such as Linux and macOS) or Windows systems. Also, users should have Python (https://www.python.org/, Version 3.5 or higher) installed, including the package manager `pip`. Additionally, the version control system `git` will be required (https://git-scm.com/). We strongly encourage users to run this code in *virtual environment*. A virtual environment is a practical solution for creating independent configurations for testing and experimenting, with no interference on the system-wide installation and without requiring complex virtualization or containerization solutions. The Python Packaging User Guide gives clear instructions on setting up a virtual enviroment on Windows, Linux and macOS.

We start by installing the dependencies from the commandline. In order to do so, we first download the code that we will use with help of `git`.

```
$ git clone https://github.com/lingpy/workflow-paper.git
$ cd workflow-paper
```

Now that we have done this, we can install all the packages we will need with help of `pip`.

```
$ pip install -r requirements.txt
```

Now that this has been done, we need to configure the access to reference catalogs, such as Concepticon and CLTS in order to make sure that they can be accessed readily by the code. This can be done with help of the `catconfig` argument submitted with the `cldfbench` package which organizes the linguistic datasets.

```
$ cldfbench catfonfig
```

You will be prompted to ask if you want to clone actual versions of Concepticon, Glottolog, and CLTS, and the easiest way to deal with this is to agree and type "y" in all cases.

3. Getting Started

There are two basic ways in which you can run our workflow.

1. You can run it by downloading a set of Python scripts and running them directly on your computer.
2. You can use the `cldfbench` package to run the commands via the commandline, without downloading the data directly.

The advantage of solution 2 is that you do not have to download extra data, since we have integrated the code directly in the `lexibank` version of the dataset of Hmong Mien languages by Chén (2012)[2]. Once this dataset has been installed (and this is the first package we have installed in the previous section as part of all dependencies needed), you can type commands on your commandline, and the code will be carried out. The disadvantage is that the code example itself is not that easy to process for people less experienced with Python. For this reason, we will only note the commands in each of the steps we discuss in the following, and not explain them in more detail.

3.1 From Raw Data to Tokenized Data

The first script essentially loads the data from the repository and creates a wordlist that contains a subselection of all the data that was used. Some aspects of the more difficult "lifting" of data have already been done and distributed along with the original datapackage (https://github.com/lexibank/chenhmongmien), which specifically also contains the orthography profile in the file `etc/orthography.tsv` and can be automatically applied with help of the `cldfbench` package.

```
$ cldfbench lexibank.makecldf chenhmongmien
```

But since the data is available in form of a `cldf` package with the original orthography already tokenized to the formats we need, you can also skip this step and convert the data to the wordlist format required by the `lingpy` package.

```
$ python 1_select.py
```

If you want to test the version from the CLDF-repository directly with `cldfbench`, you can type:

```
$ cldfbench chenhmongmien.wf_select
```

This will select a part of the languages and a part of the concepts, as indicated in the main study and write them to a file `D_Chen_subsets.tsv`. Additionally, you will see some statistics on the terminal, specifically a table indicating the coverage for each language. If you want to select all languages, and not just a subset, type:

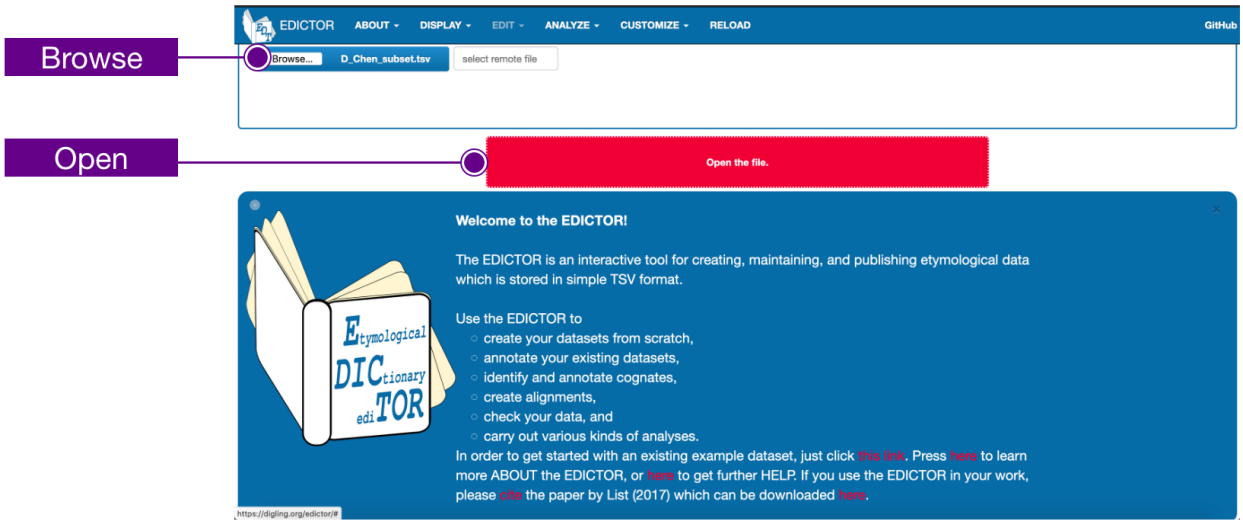
```
$ python 1_select.py all
```

The output `A_Chen_subset.tsv` is generated due to the argument `all` is used. Once the argument `all` is used in the first stage, it has to be added to the rest of stages to ensure that the workflows process the correct files.

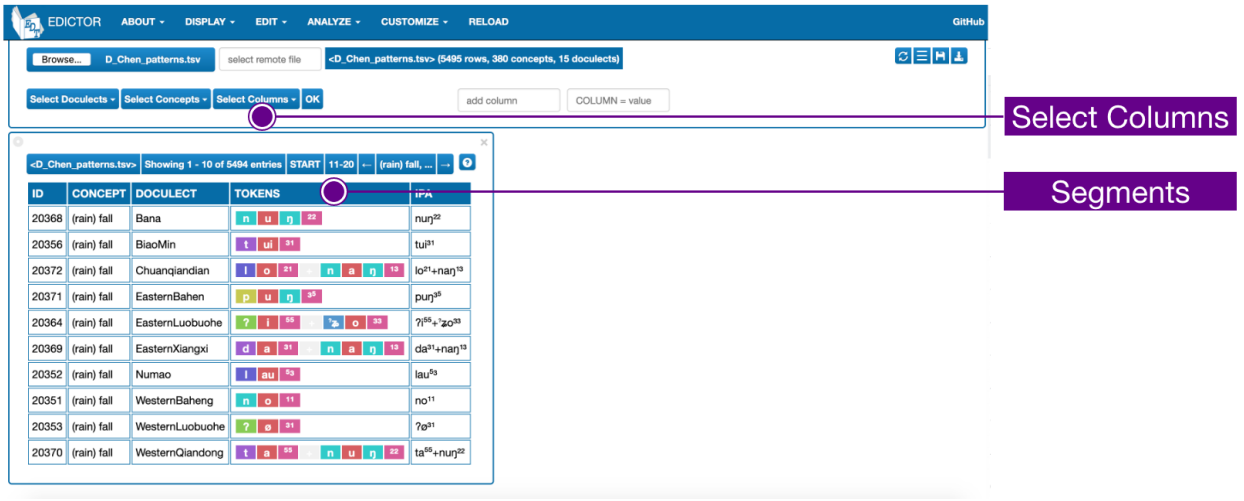
Doculect	Words	Coverage
Bana	502	1.00
BiaoMin	488	0.97
CentralGuizhouChuanqiandian	454	0.90
Chuanqiandian	501	1.00
EasternBahen	492	0.98
EasternLuobuohe	499	0.99
EasternQiandong	442	0.88
EasternXiangxi	492	0.98
Numao	490	0.98
WesternBaheng	500	1.00
WesternLuobuohe	488	0.97
WesternQiandong	494	0.98
WesternXiangxi	502	1.00
Younuo	500	1.00
ZaoMin	455	0.91

Already now you can inspect the data with help of the EDICTOR tool. In order to do so, open the tool's website at https://digling.org/edictor/ and wait until the page is loaded (note that we recommend to browse EDICTOR in Firefox, but GoogleChrome should also not cause further problems).

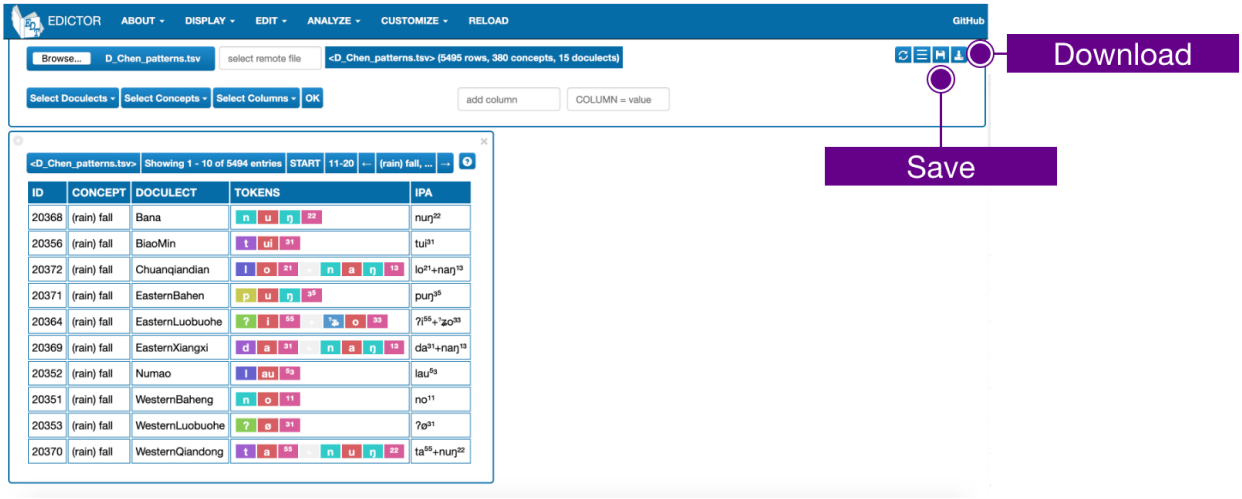
The data is in the file `D_Chen_subset.tsv`, in order to load it to the tool, press the **Browse** button and select the file. Once this has been done, press the **Open the file** button to examine the data, as illustrated in the following figure.



The segmented strings are displayed in the TOKENS column. Press **Select Columns** to inspect the raw forms and other aspects of the data, as shown in the following figure.



In order to save data to your computer, after you have manually edited them, you need to ”download” them. This may be a bit surprising, since effectively, you do not download the data, but since the EDICTOR is working on a browser, it does not have any access to the data on your computer, and ”download” is the only way to communicate with your machine. Thus, in order to save your data and load it to your machine, you first have to press the **save** icon at the top-right corner in order to to store the edited data in the web browser. When now pressing the **download** icon at the top-right, your browser will either directly download the data and store them in your download folder, or it will ask you to specify a specific file destination.



Be careful when editing data in the EDICTOR without saving and downloading them. If you close your browser, all the edits you made will be lost, so you should regularly save and download your data when working with the EDICTOR. As a shortcut, you can also type CONTROL+S to save and CONTROL+E to "export" the data (i.e., to download them).

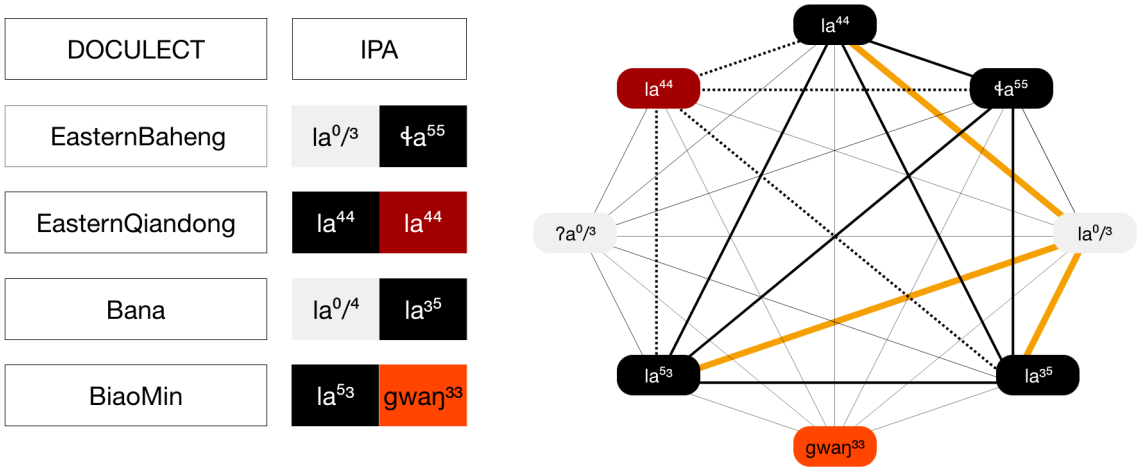
3.2 From Tokenized Data to Cognate Sets

Partial cognate detection is an important task, specifically when working with Southeast Asian language data. The algorithm we use for this taks was first proposed in the study "Using Sequence Similarity Networks to Identify Partial Cognates in Multilingual Wordlists" by List et al. (2016)[3], where the algorithm is described in due detail.

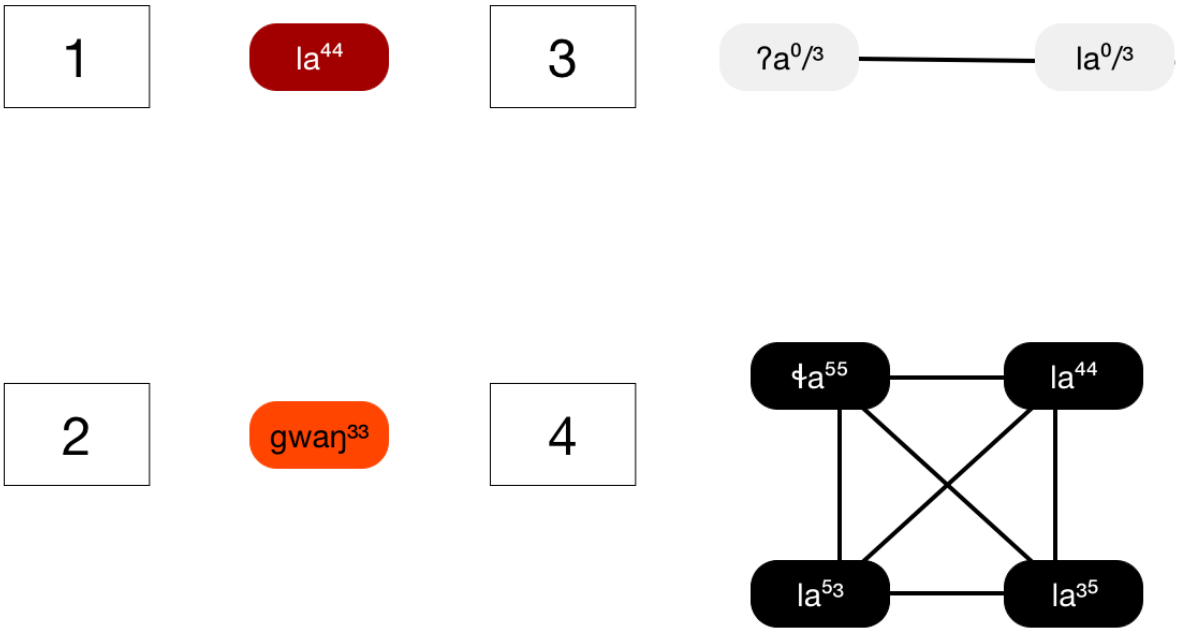
To illustrate how the algorithm works, we provide an example with four words for 'moon' in the Eastern Baheng, Eastern Qiandong, Bana and Biao Min language varieties.

The major steps of the algorithm are the following:

1. Calculate the distances of all morpheme pairs.
2. Create a fully connected network from the distance scores.
3. Filter the network by deleting edges in the following fashion: A. Two morphemes in the same word should not be linked (see the dashed lines in the following figure). B. A morpheme in a word should not be linked to two morphemes in another word (see the yellow edges in the figure).
4. Remove the edges with similarity scores below a given threshold.



Once this has been done, an algorithm for Community Detection in networks[4] is used to partition the network into ”communities”, with each community representing one partial cognate set.



In order to calculate partial cognates, we use the algorithm as provided by the `lingpy` software package and apply it to our subselection of languages.

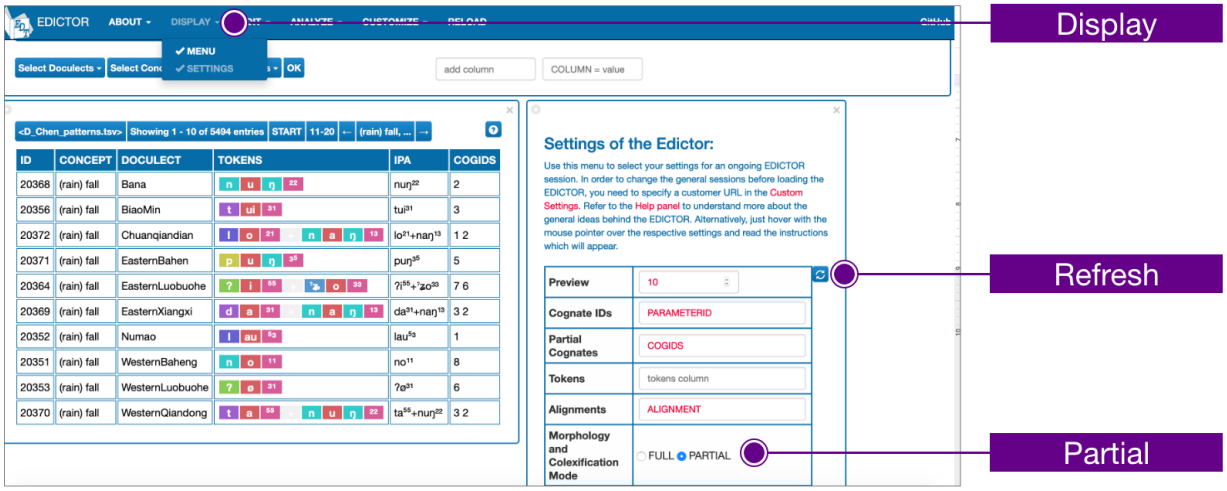
```
$ python 2_partial.py
```

If you want to test the version from the CLDF-repository directly with `cldfbench`, you can type:

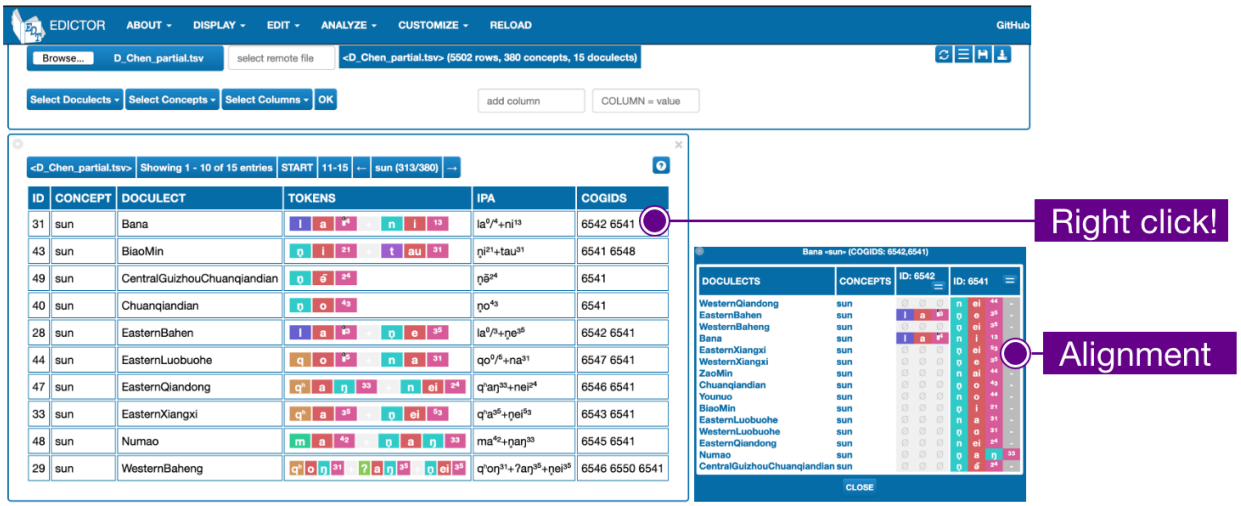
```
$ cldfbench chenhmongmien.wf_partial
```

This will take some time when you run it the first time. The data can be found in the file `D_Chen_partial.tsv`.

To inspect the data with EDICTOR, load `D_Chen_partial.tsv` as shown before. Then press **DISPLAY** to select **SETTINGS** in the drop-down menu. Select **PARTIAL** in the **Morphology and Colexification Mode** entry. Press the **Refresh** button.



In order to investigate the partial cognates, you need to select the column which stores the identifiers. To do so, press **Select Columns** and select **COGIDS** in the drop-down menu. If you right-click on any number in the "COGIDS" column, a pop-up window will open and show all the cognate sets for a given word form in form of an alignment. Since we have not yet aligned the data, the alignment will be wrong at this point.



3.3 From Cognate Sets to Alignments

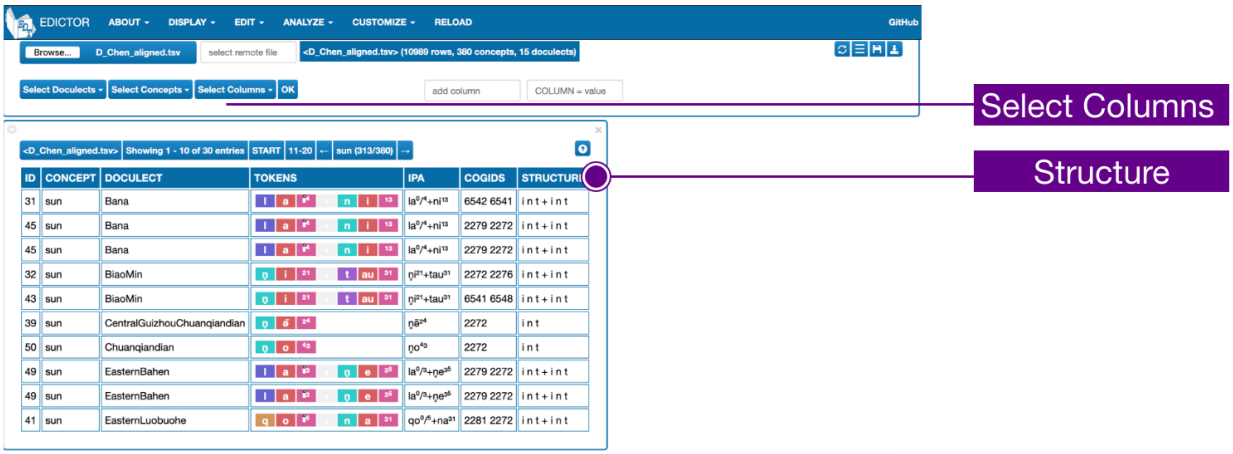
To align the data, we use the new procedure for template-based alignment, which is available from the `lingrex` package which we have installed as one of the requirements of our workflow, and the `sinopy` package, which helps us to compute syllable templates from all morphemes in the data. Running the code is again straightforward.

```
$ python 3_alignment.py
```

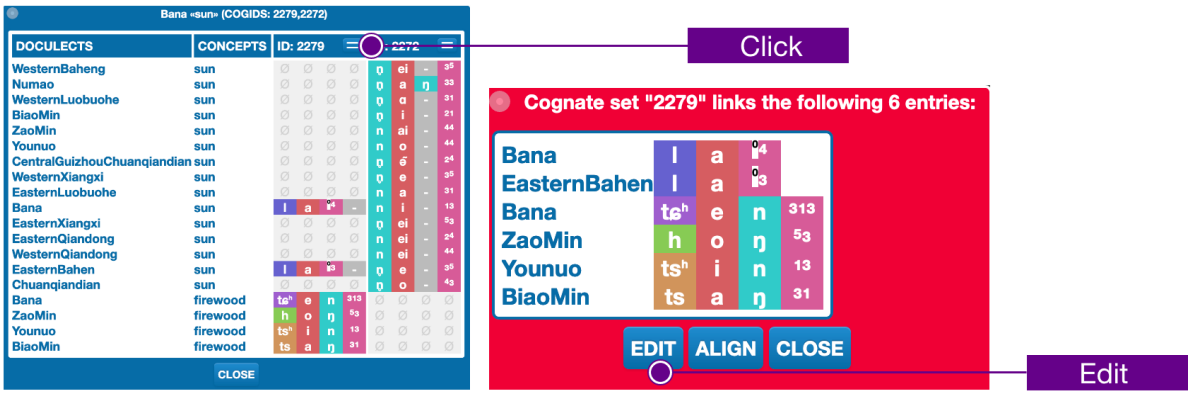
If you want to test the version from the CLDF-repository directly with `cldfbench`, you can type:

```
$ cldfbench chenhmongmien.wf_alignment
```

The aligned data will be stored in the file `D_Chen_aligned.tsv`. To inspect the alignments in EDICTOR, load this file and follow the previous steps we mentioned in Section 3.2. In addition to selecting the **COGIDS** column now, we also select the **STRUCTURE** column, since this column provides the templates for each morpheme, which we have automatically added to the data with help of `sinopy`.



As we already mentioned, if you right-click on any number in the "COGIDS" column, a pop-up window will show the alignment. Click on the = sign to modify the alignment. The modification itself is very straightforward: just click on a sound segment to move it to the right, and click on a gap segment to delete this segment.



3.4 From Alignments to Cross-Semantic Cognates

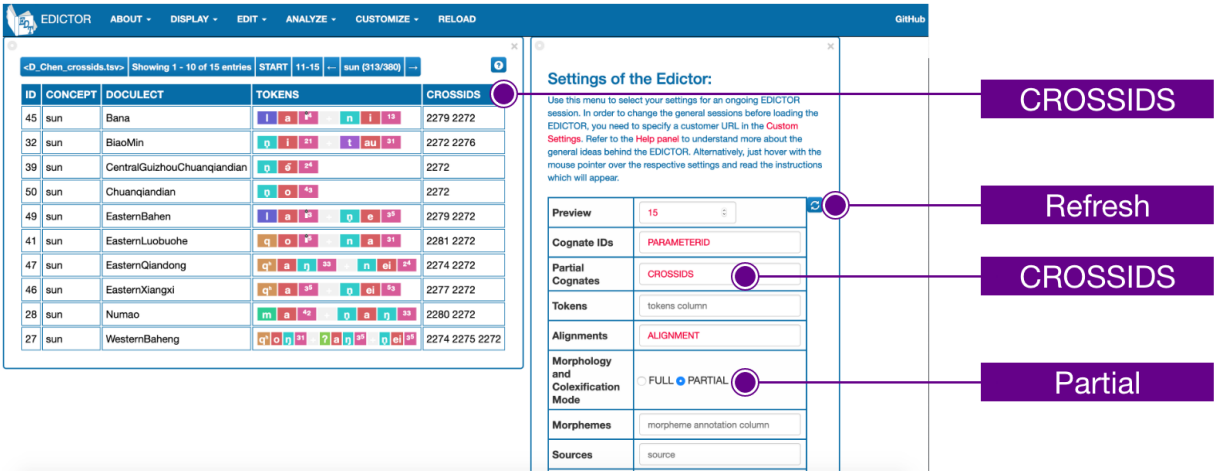
The algorithm for cross-semantic cognate detection as we propose it here is illustrated in more detail in the main study. It is implemented as part of the `lingrex` package. Again, it is straightforward to run the code.

```
$ python 4_crosssemantic.py
```

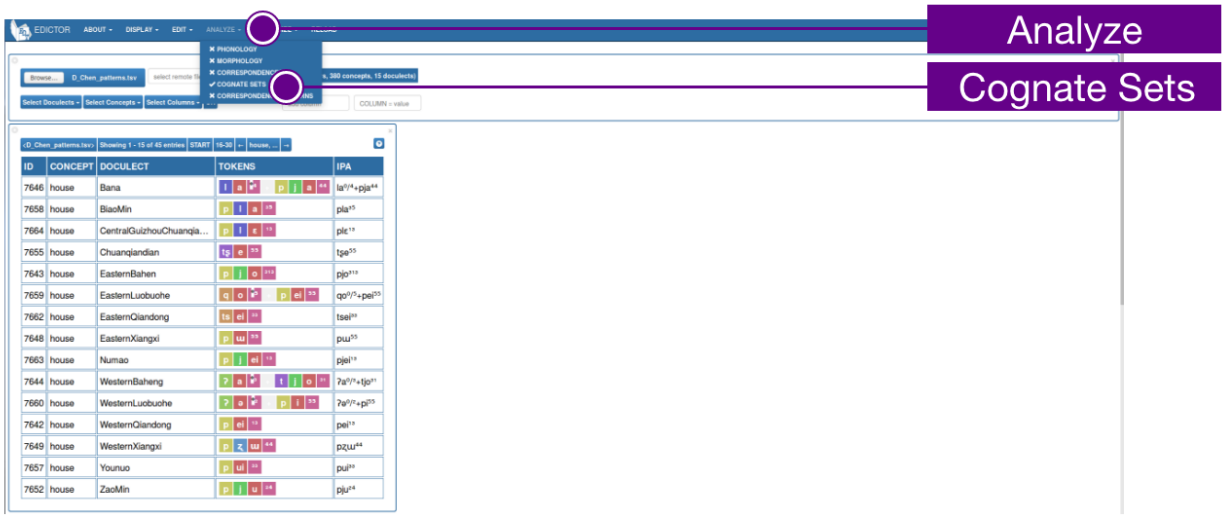
If you want to test the version from the CLDF-repository directly with `cldfbench`, you can type:

```
$ cldfbench chenhmongmien.wf_crosssemantic
```

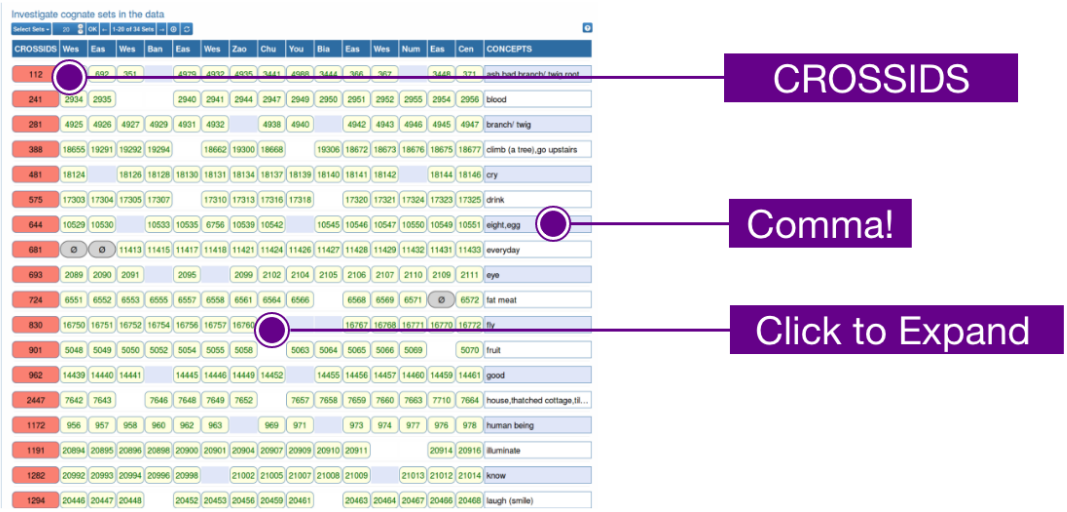
The output file is `D_Chen_crossids.tsv`, and we load it into the EDICTOR tool, just as we did before, but when checking the **SETTINGS** in the menu this time, we need to specify that the column "CROSSIDS" holds the partial cognates. To do so, just type in **CROSSIDS** in the text field **Partial Cognates** in the settings menu and then press the **refresh** button.



To inspect the distribution of partial cognates, press **ANALYZE** in the top-level menu and select **Cognate sets** in the drop-down menu.



As a result, a new panel will open and show the distribution of all cognate sets across the different language varieties. Pressing the red button with the cognate set identifier on the left will open the alignment. Pressing the yellow buttons with the word identifiers will show you the original morpheme. On the right, in the column **CONCEPTS**, you will find those cognate sets which are attested for more than one concept as separated by a comma. Clicking on this field will modify the main wordlist panel in such a way that only the selected concepts will appear.



3.5 From Cross-Semantic Cognates to Sound Correspondence Patterns

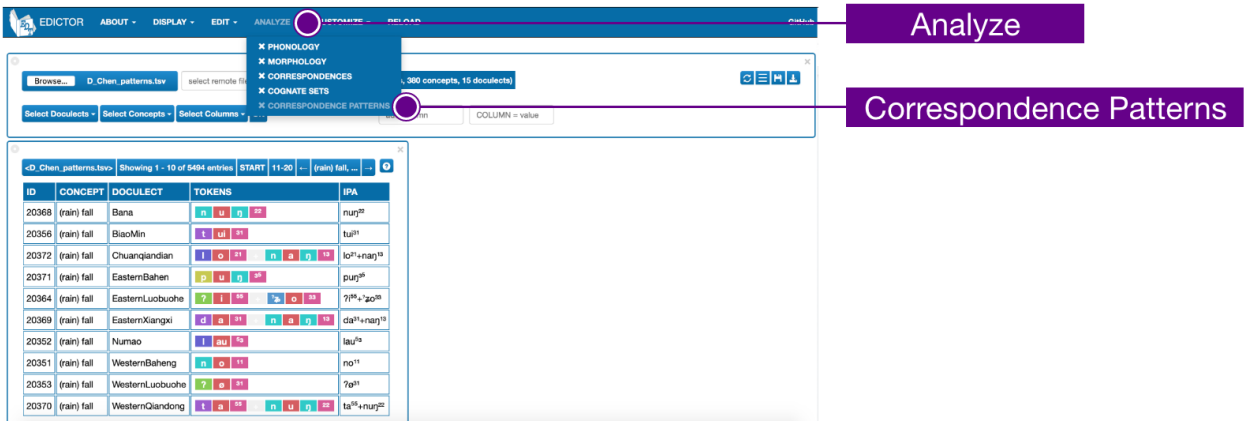
As a final step, we will try to infer the major correspondence patters in the data, using the algorithm by List (2019)[5] which is available from the `lingrex` package. Running the code is straightforward, as before.

```
$ python 5_correspondence.py
```

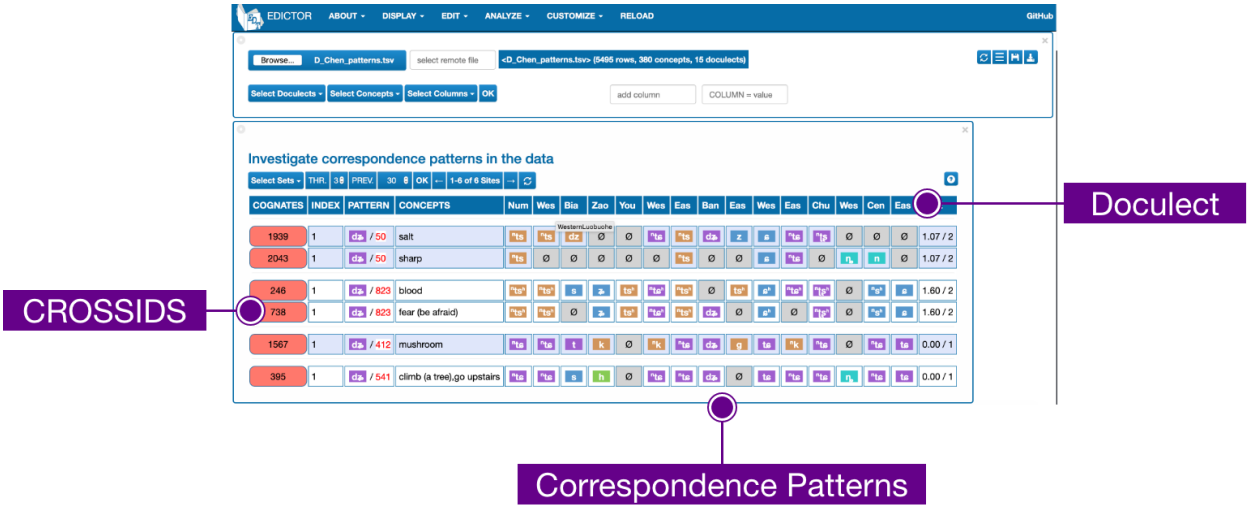
If you want to test the version from the CLDF-repository directly with `cldfbench`, you can type:

```
$ cldfbench chenhmongmien.wf_correspondence
```

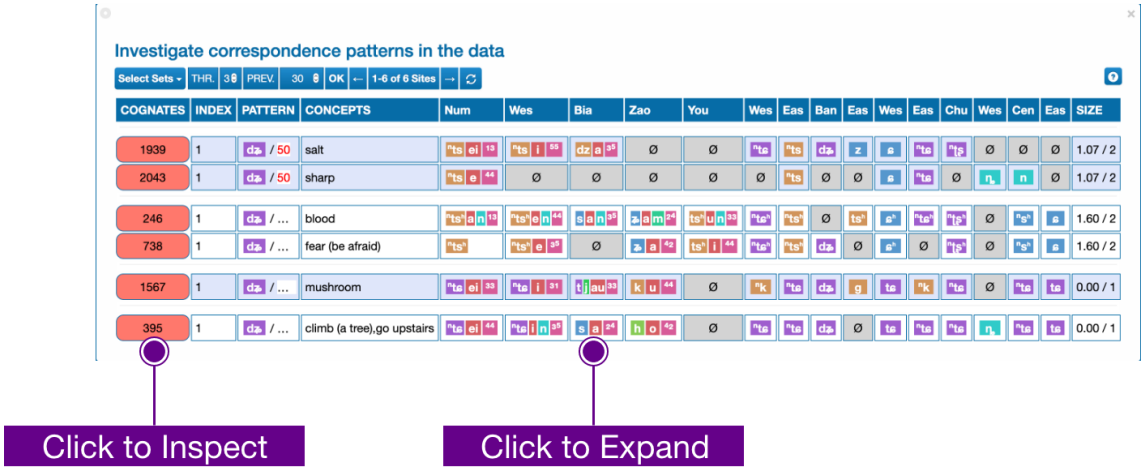
This creates two output files. One, called `D_Chen_patterns.tsv` is the file without wordlist that can be loaded by EDICTOR and inspected, and one file contains the patterns that have been inferred alone, called `D_patterns_Chen.tsv`. In order to inspect the patterns, we recommend to use the EDICTOR tool, which requires the same steps that we already applied when loading our cross-semantic cognates. Once this has been done, press the **ANALYZE** button in the top menu and select **CORRESPONDENCE PATTERNS** in the drop-down menu.



In order to allow for a good display, the doculect names are all abbreviated. Hovering the mouse cursor on an abbreviation will show you the full name.



Clicking on a cell in the correspondence pattern panel will allow you to see not only the sound in question, but the full morpheme in which this sound occurs.



4 Validation

We calculate the shared cognates between language pairs and output the scores in the form of a pairwise distance matrix. The script `6_phylogeny.py` gives two documents, a distance matrix (`A_Chen_distance.dst` or `D_Chen_distance.dst`) and a tree file, based on a Neighbor-Joining analysis (`A_Chen_tree.tre` or `D_Chen_tree.tre`).

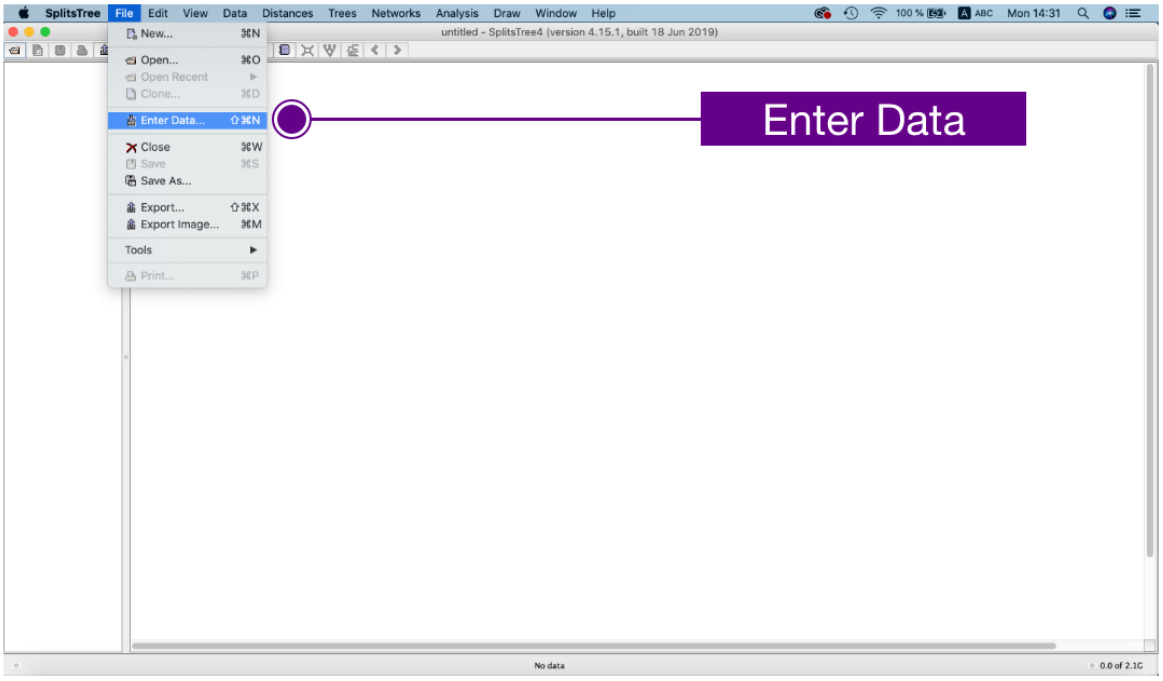
There are many ways to work with the distance matrix , here, we gives one of the approach to visualize the matrix as a neighbor-net network with the help of SplitsTree.

To get started, first make sure to install SplitsTree [6] from https://software-ab.informatik.uni-tuebingen.de/download/splitstree4/welcome.html and follow the installation instructions. In order to compute the distance matrix with our code, use the command line (here we compute it for the entire dataset, so we run it with the keyword `all`)

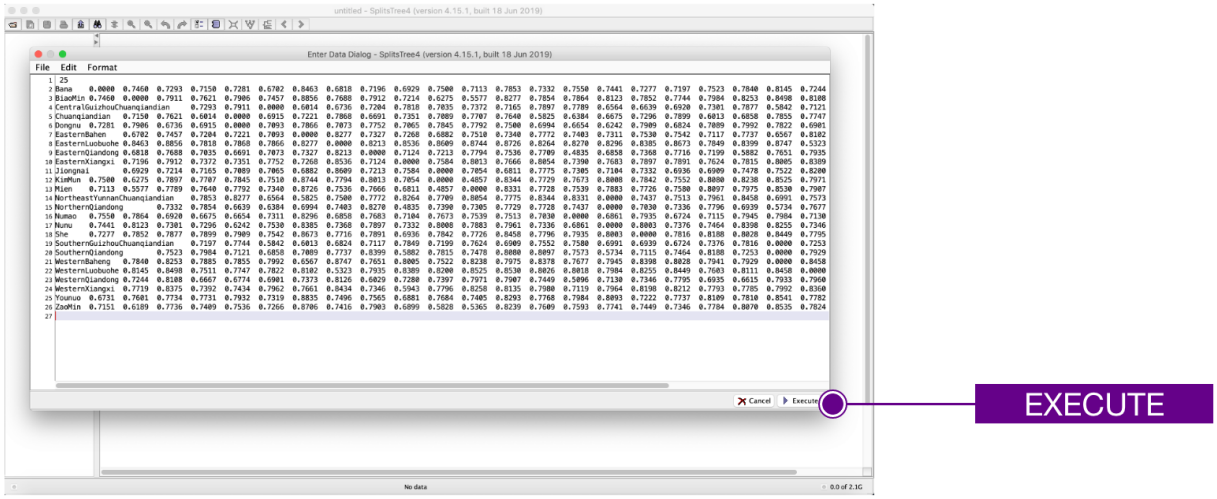
```
python 6_phylogeny.py all
```

To generate a Neighbor-Net from the distance matrix, open the file **A__Chen__distance.dst** or **D__Chen__distance.dst** with any plain text editor and start the SplitsTree software. Then click on **File**

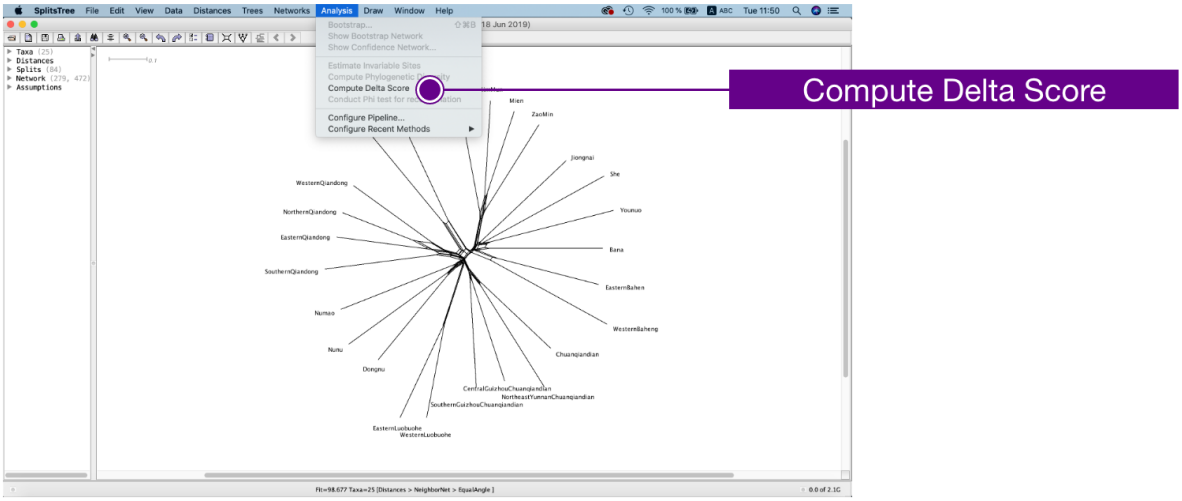
and **Enter Data**, as shown in the image below.



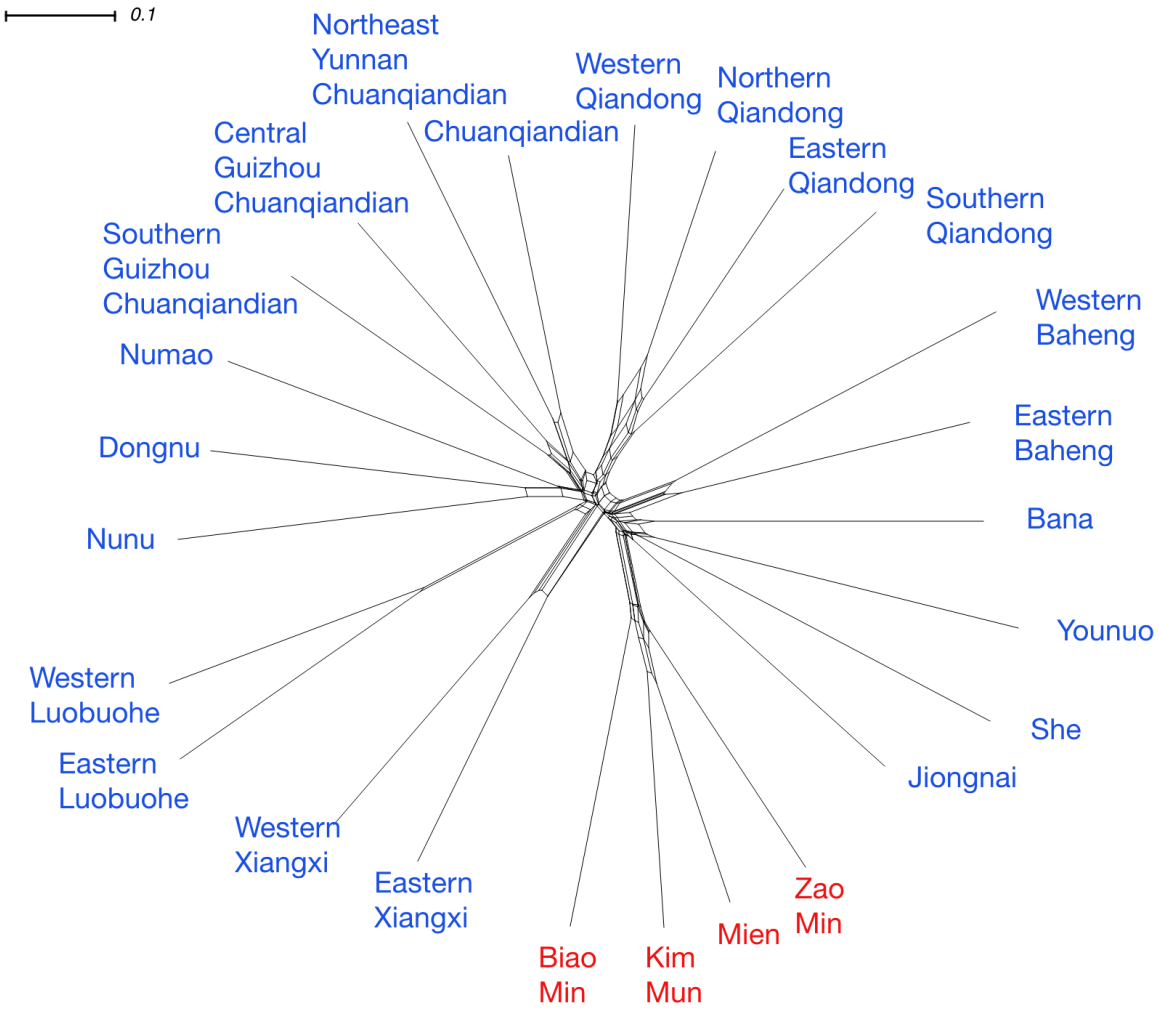
Then copy the distance matrix in the paste it into the **Enter Data Dialog**, and press **Execute**



You can now inspect the network. To analyze the data further, youc an compute the delta scores, showing the degree of reticulation in the data, by pressing **Analysis** and then **Compute Delta Score**, as shown below.



The resulting Neighbor-Net is shown in the following figure. For the purpose of illustration, the Mienic language varieties are colored in red, the Hmongic group is highlighted in blue.



The following table shows the delta scores we computed from the data.

Taxon	Delta score
Bana	0.34706
Biao Min	0.27289
Central Guizhou Chuanqiandian	0.29924
Chuanqiandian	0.29172
Dongnu	0.32416
Eastern Baheng	0.32056
Eastern Luobuohe	0.33529
Eastern Qiangong	0.32083
Eastern Xiangxi	0.33736
Jiongnai	0.32644
Kim Mun	0.26992
Mien	0.25672
Northeast Yunnan Chanqiandian	0.29748
Northern Qiandong	0.28447

Taxon	Delta score
Numao	0.34185
Nunu	0.32375
She	0.31671
Southern Guizhou Chuanqiandian	0.34376
Southern Qiandong	0.30988
Western Baheng	0.35259
Western Luobuohe	0.3211
Western Qiandong	0.31137
Western Xiangxi	0.35174
Younuo	0.2996
Zao Min	0.26797

The average delta score is 0.313. As mentioned before, the distances between taxa are calculated via shared cognates. The shorter the distances between two taxa, the higher the similarities between them. If the taxa share cognates not only within their group but also outside their groups, the network finds it challenging to determine the best cluster for them. The larger the reticulate structure, or the less tree-like the data is, the higher is the delta score. For one particular language variety's delta score this means that this specific language contributes to a certain amount of conflict in the data.

5. Conclusion

In this tutorial, we provided details of how to execute our workflow for Computer-Assisted Language comparison, using the scripts we wrote, while at the same time illustrating how the results can be manually inspected and modified. We have not discussed the details of the code we wrote, but wer recommend users proficient in Python to have a look.

6. References

1. List J-M, Walworth M, Greenhill SJ, Tresoldi T, Forkel R. Sequence comparison in computational historical linguistics. Journal of Language Evolution [Internet]. 2018;3(2):130–44. Available from: https://academic.oup.com/jole/article/3/2/130/5050100?guestAccessKey=cf8fe64e-3996-4cb1-ba2c-317a7cd81bf4
2. 陳其光 CQ. Miàoyáo yǔwén [Internet]. Běijīng: Zhōngyāng Mínzú Dàxué 中央民族大学 [Central Institute of Minorities]; 2012. Available from: https://en.wiktionary.org/wiki/Appendix:Hmong-Mien_comparative_vocabulary_list
3. List J-M, Lopez P, Bapteste E. Using sequence similarity networks to identify partial cognates in multilingual wordlists. In: Proceedings of the Association of Computational Linguistics 2016 (Volume 2: Short Papers) [Internet]. Berlin: Association of Computational Linguistics; 2016. pp. 599–605. Available from: http://anthology.aclweb.org/P16-2097
4. Rosvall M, Bergstrom CT. Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci USA. 2008;105(4):1118–23.
5. List J-M. Automatic inference of sound correspondence patterns across multiple languages. Computational Linguistics [Internet]. 2019;1(45):137–61. Available from: https://www.mitpressjournals.org/doi/full/10.1162/coli_a_00344
6. Huson DH. SplitsTree: Analyzing and visualizing evolutionary data. Bioinformatics. 1998;14(1):68–73.