Sequence Comparison in Computational Historical Linguistics Phonetic Alignments and Cognate Detection with LingPy 2.6

Johann-Mattis List, Mary Walworth, Simon J Greenhill, Tiago Tresoldi, Robert Forkel

Sequence Comparison in Computational Historical Linguistics

Phonetic Alignments and Cognate Detection with LingPy 2.6

Johann-Mattis List[1]*, Mary Walworth[1], Simon J. Greenhill[12], Tiago Tresoldi[1], Robert Forkel[1]

[1]Department of Linguistic and Cultural Evolution (MPI-SHH, Jena)

[2]ARC Centre of Excellence for the Dynamics of Language (Australian National University, Canberra)

*Corresponding author: mattis.list@shh.mpg.de

Manuscript accepted for publication in Journal of Language Evolution, Mai 2018

With increasing amounts of digitally available data from all over the world, manual annotation of cognates in multi-lingual word lists becomes more and more time-consuming in historical linguistics. Using available software packages to pre-process the data prior to manual analysis can drastically speed up the process of cognate detection. Furthermore, it allows us to get a quick overview on data which has not yet been intensively studied by experts. LingPy is a Python library which provides a large arsenal of routines for sequence comparison in historical linguistics. With LingPy, linguists can not only automatically search for cognates in lexical data, they can also align the automatically identified words, and output them in various forms, which aim at facilitating manual inspection. In this tutorial, we will briefly introduce the basic concepts behind the algorithms employed by LingPy and then illustrate in concrete workflows how automatic sequence comparison can be applied to multi-lingual word lists. The goal is to provide the readers with all information they need to (a) carry out cognate detection and alignment analyses in LingPy, (b) select the appropriate algorithms for the appropriate task, (c) evaluate how well automatic cognate detection algorithms perform compared to experts, and (d) export their data into various formats useful for additional analyses or data sharing. While basic knowledge of the Python language is useful for all analyses, our tutorial is structured in such a way that scholars with basic knowledge of computing can follow through all steps as well.

1 Introduction

Sequence comparison is one of the key tasks in historical linguistics. By comparing words or morphemes across languages, linguists can identify which words have sprung from a common source in genetically related languages, or which words have been borrowed from one language

to another. By comparing words within a language, linguists can identify grammatical and lexical morphemes, cluster words into families, and shed light on the internal history of languages. So far the majority of this work has been carried out manually. Linguists sift through dictionaries and fieldwork notes, trying to identify those words which reflect a shared history across languages. All etymological dictionaries available today have been based on manual word comparison and their results fill thousands of pages. Even the largest databases which offer cognate judgments, such as the *Austronesian Basic Vocabulary Database* (ABVD, Greenhill et al. 2008) or the *Indo-European Lexical Cognacy Database* (Dunn, 2012) are based on manual assessments of cognacy.

With increasing amounts of digitally available data it becomes harder for linguists to keep up. For example, the *Sino-Tibetan Etymological and Thesaurus database* (Matisoff, 2015), contains more than 500 000 words, but only a small amount of words have been compared etymologically (see Hill and List 2017, 64f). We need to take advantage of increasing amounts of data, refining work on well-established languages, and fostering work on the world's understudied languages. To do this, however, we will have to rethink the way we compare languages.

Historical linguists are skeptical about automating the methods for cognate identification (see (Holman et al. 2011 and commentaries, as well as List et al. 2017b). First, the *accuracy* of automated methods is often low, failing to reproduce the analyses of linguistic experts. Especially the use of the *edit distance* (Levenshtein, 1965) has been criticized for being linguistically too naïve, conflating sound correspondences and lexical replacement, to be useful for subgrouping or cognate detection (Campbell, 2011; Greenhill, 2011). Second, it is hard to *verify* many algorithms as they are seen as black-boxes which hide the crucial decisions leading to cognate judgments and subgroupings, making it difficult for scholars to determine whether similarities are due to inheritance or contact (Jäger, 2015; List et al., 2017b). The non-transparency of automatic methods is highly problematic for computational historical linguistics: if we do not know what evidence decisions are based on, we cannot criticize and improve them.

However, methods for automatic sequence comparison in historical linguistics have dramatically improved during the last two decades. Starting with the pioneering work on pairwise and multiple phonetic alignment (Kondrak, 2000; Prokić et al., 2009), new methods for phonetic alignment and automatic cognate detection solve both the problems of verification and accuracy (List et al., 2017b; Jäger et al., 2017). First, these algorithms are based on phonetically-informed metrics on sound similarities. Importantly, any algorithmically identified correspondences are logged and can be inspected by researchers. Second, in a wide-ranging test of these methods, they have been found to be highly accurate and able to correctly identify cognates in almost 90% of the cases (List et al., 2017b).

LingPy (List et al. 2017a) provides these algorithms as part of a stable open source software package that works on all major platforms. Given the complexity of the problems involving sequence comparison in historical linguistics, computers will not be able to replace human judgments any time soon, but with the recent advancements, the methods are definitely good enough to provide substantial help for classical historical linguists to *pre-analyze* the data to be later corrected by experts, or to check the consistency of human cognate judgments. Over the long run, computational methods can also contribute to the bigger questions of language evolution, be it indirectly, by increasing the amount of digitally available high-quality annotated data, or directly, by providing scholars access to data too large to be processed by humans alone.

In the following, we will give a concise overview on how automatic sequence comparison can be carried out. After discussing general aspects of sequence comparison (Sec. 2), we will introduce basic ideas on the data needed (Sec. 3). We will then turn to the core tasks of automatic sequence comparison, namely automatic phonetic alignment (Sec. 4) and automatic cognate detection (Sec. 5). We conclude by showing how automatic approaches for cognate detection can be evaluated (Sec. 6), and how results can be exported to various formats (Sec. 7).

This paper is supplemented by a detailed interactive tutorial in form of an IPython Notebook (Pérez and Granger, 2007) which illustrates how all methods discussed here can be practically applied. Having installed the necessary software (Tutorial: 1), readers can follow the tutorial step by step and investigate how the algorithms work in practice. Our data is based on a small sample of Polynesian languages taken from the ABVD, which we substantially revised, both with respect to the phonetic transcriptions and the expert cognate judgments. All data needed to replicate the analyses discussed here are supplemented. We give more information in the interactive tutorial (Tutorial: 2.1).

2 Basic Aspects of Sequence Comparison

The words and morphemes which constitute a language are best modeled as *sequences of sounds*. Sequences have information content not only from their elements (*segments*, whether these are phonemes, graphemes, or morphemes) but also via the *order* of the elements, a consistent comparison of sequences should account for both order and content. *Alignments* are a very general way to model differences between sequences. The major idea is to arrange two or more sequences in a matrix in such a way that similar or identical segments which occur in similar positions are placed in the same column of the matrix. If segments are missing in one sequence where no counterpart for a segment can be found, this is represented by a *gap character*, usually the *dash*-symbol (List, 2014b).

Figure 1: Early alignment example for translational equivalents of “nail" in aboriginal languages of California (based on Dixon and Kroeber 1919), contrasted with a “modern" representation using the EDICTOR tool (List 2017).

Sequence alignments are crucial in biology, where they are used to compare protein and DNA sequences (Durbin et al., 2002). In historical linguistics, however, they are usually only *implicitly* employed, and initial attempts to arrange cognate words in a matrix go back to the early

20th century, as one can see from an early example based on Dixon and Kroeber (1919, 61) given in Figure 1. The authors themselves describe this way of representing sequence similarities as a 'columnar form' with the goal to 'bring out parallelisms that otherwise might fail to impress without detailed analysis and discussion' (Dixon and Kroeber, 1919, 55). The figure further shows how the data would look if they were rendered in contemporary alignment editors for historical linguistics (List, 2017). Dixon and Kroeber's wording nicely expresses one of the major advantages of alignments: the transparency of homology assessments. Scholars often list long lists of cognate sets in the literature, claiming that all words are somehow related to each other, but if they do not list the alignments, it is often impossible, even for experts in the same language family, to understand *where exactly* the authors think that certain segments are similar.

Given that the inference of historically related words is *not* based on superficial word similarities but on recurrent systematic similarities, known as *regular sound correspondences* (Lass, 1997, 130), all judgments regarding the relatedness of words across languages directly rely on previously established sequence alignments (Fox, 1995, 67f). Alignment analyses not only increase the transparency of cognate judgments, they play a crucial role in substantiating these judgments in a first place. As can be seen from Table 1, similarities in cognate words in Sikaiana and Tahitian (data taken from Greenhill et al. 2008) are *not* based on the identity of sounds, but rather in the regularity of occurrence: whenever Sikaiana has a [k] and a [l], Tahitian has a [ʔ] and a [r], respectively. Without alignments, we could not identify this similarity. Alignments are also at the core of all automatic sequence comparison approaches in historical linguistics, as we will see throughout this tutorial.

Cognate List		Alignment				Correspondences		
Sikaiana	*louse*	k	u	t	u	Sik.	Tah.	Freq.
Tahitian	*louse*	ʔ	u	t	u	k	ʔ	3 x
Sikaiana	*dog*	k	u	l	iː	u	u	3 x
Tahitian	*dog*	ʔ	u	r	iː	t	t	1 x
Sikaiana	*skin*	k	i	l	i	r	l	2 x
Tahitian	*skin*	ʔ	i	r	i	i(ː)	i(ː)	3 x

Table 1: Recurring similarities in Sikaiana and Tahitian

3 Data Preparation

When searching for cognates across languages, we usually assume that our data are given in some kind of *wordlist*, a list in which a number of concepts is translated into various languages. How many concepts we select depends on the research question, and various concept lists and questionnaires, ranging from 40 (Brown et al., 2008) up to more than 1000 concepts (Haspelmath and Tadmor, 2009) have been proposed so far (see the overview in List et al. 2016a). Our data example for this tutorial is based on the questionnaire of the ABVD project (Greenhill et al., 2008), consisting of 210 concepts, which were translated into 31 different Polynesian languages. For closely related languages, such as those in the Polynesian family, this gives us enough information to infer regular correspondences automatically, although it is clear that for

analyses of more distant language relationship the number of words per language may not be enough.

numeric identifier	language name	concept label	value in source	phonetic transcription	space-segmented transc,	variants of word, if given	cognate identifier
ID	**DOCULECT**	**CONCEPT**	**VALUE**	**IPA**	**TOKENS**	**VARIANTS**	**COGID**
188	Emae	Eight	βaru	βaru	β a r u		750
447	Rennell_Bellona	Eight	baŋgu	baŋgu	b a ⁿg u		750
703	Tuvalu	Eight	valu	valu	v a l u		750
927	Sikaiana	Eight	valu	valu	v a l u		750
1135	Penrhyn	Eight	varu	varu	v a r u		750
6114	Kapingamarangi	Eight	waru,walu	waru	w a r u	walu	750
6115	Kapingamarangi	Eight	waru,walu	walu	w a l u	waru	750

Figure 2: Input format required by the LingPy package. The last two entries show how synonyms can be handled by placing different variants of one concept in one language variety into different rows with a separate ID each.

The basic format used by LingPy is a tab-separated input file in which the first row serves as a header and defines the content of the rest of the rows. The very first column is reserved for numerical identifiers (which all need to be unique), while the order of the other columns is arbitrary, with specific columns being required, and others being optional. Essential columns which always must be provided are the *language name* (`DOCULECT`), the *comparison concept* (`CONCEPT`), the *original transcription* (`IPA`, `FORM`, or `VALUE`), and a *space-segmented form* of the transcription (`TOKENS`). Multiple synonyms for the same comparison concept in the same language should be written in separate rows and given a separate `ID` each. The data in the `TOKENS`-column should supply the transcriptions in *space-segmented form*, that is, instead of transcribing the Fila word for "all" as [eutʃi], the software expects [e u tʃ i], which is internally interpreted as a sequence of five segments, namely [e], [u], [tʃ] and [i], with [tʃ] representing a voiceless post-alveolar affricate. If the `TOKENS` are not supplied to the algorithm, it will try to segment the data automatically, provided it can find the column `IPA`, which is otherwise not necessarily required to appear in the data. This however, may lead to various problems and unexpected behavior. We therefore urge all users of LingPy to make sure that they supply segmented data to the algorithm, making furthermore sure that they adhere to the general standards of transcription as they are represented in the International Phonetic Alphabet (IPA, IPA 1999).[1] The format can be created manually by using either a text-editor, or a spreadsheet program that allows to export to tab-separated format. To a large degree, this input format is compatible with the one advocated by the Cross-Linguistic Data Formats (CLDF) initiative (Forkel et al., 2017), the main difference being that LingPy requires a flat single file with tabstop as separators, while

[1] Linguists are often skeptical when they hear that LingPy requires explicit phonetic transcriptions, and often, they are even reluctant to interpret their data along the lines of the IPA. But in order to give the algorithms a fair chance to interpret the data in the same way in which they would be interpreted by linguists, a general practice for phonetic transcriptions is indispensable, and the IPA is the most widely employed transcription system.

CLDF supports multiple files. CLDF furthermore encourages the use of reference catalogs, such as Glottolog (Hammarström et al., 2017) or Concepticon (List et al., 2018), in order to increase the comparability of linguistic data across datasets, while LingPy is indifferent regarding the overall comparability as long as the data is internally consistent. As of version 2.6, LingPy offers routines to convert to and from CLDF (see Tutorial: 6.3). Figure 2 provides a basic summary on LingPy's input formats. More information on the format, and how it can be loaded into LingPy can be found in the supplemented interactive tutorial (Tutorial: 2.2-3).

Data quality and consistency plays a crucial role in the outcome of an automatic sequence comparison. As a general rule of thumb, we recommend all linguists who apply LingPy or other software to carry out automatic sequence comparison, to pay careful attention to what we call the SANE rules for data sanity: users should pay close attention to providing a sensible *segmentation* of their data, they should *aim* for high coverage, there should be *no* mixing of data from different sources (as this usually leads to inconsistent transcriptions and may also increase the number of synonyms), and synonyms should be *evaded*.[2] These rules are summarized in Table 2. If the original data does not provide reliable phonetic transcriptions, as it was the case with the Polynesian data we use in this tutorial, *orthography profiles* (Moran and Cysouw, 2017) provide an easy way to refine transcriptions while at the same time segmenting the data, and the EDICTOR tool (List, 2017) offers convenient ways to check phonological inventories of all varieties (Tutorial: 2.4). Various coverage statistics can be computed in LingPy (see Tutorial: 2.5). Synonym statistics can also be easily computed (see Tutorial: 2.6). Users should always keep in mind that the quality of automatic sequence comparison crucially depends on the quality of the data submitted to the algorithms.

4 Automatic Phonetic Alignment

Alignments are crucial for historical language comparison to search for *regular sound correspondence* patterns, *layers of borrowed words*, or even use them as the starting point for *linguistic reconstruction* (Fox, 1995). A further important advantage is that they can be easily quantified, as we will see in Sec. 5. Since phonetic alignment is heavily influenced by bioinformatics, linguists using phonetic alignments should have some basic understanding of original algorithms and terminology. In this context, it is not necessarily important to understand *how* the algorithms work in detail. Instead, we think it is more important to learn (also by testing the algorithms with different data and parameters) how the different options from which users can choose influence the results. In the following, we will quickly introduce basic algorithms and concepts involving alignments in historical linguistics, and how they relate to alignments in bioinformatics. We will follow the traditional division into *pairwise* and *multiple alignments* (which result from the differences in complexity of the algorithms), and introduce the most important concepts and parameters that users should know when applying the methods.

[2] We know well how difficult it is to conform to the latter point. What is clear is that tossing coins to select one out of many synonyms, as originally suggested by Gudschinsky (1956), will have a deleterious impact on any analysis (List, 2018). In order to avoid synonyms in qualitative work, we recommend to thoroughly review the guidelines in Kassian et al. (2010).

Segmentation matters	*Example*	
Consistent phonetic transcription and segmentation are of crucial importance for automatic sequence comparison. Computers cannot guess whether multiple graphemes represent separate or single sound segments.	**NOT:** **BUT:**	Fila [eutʃi] “all" Fila [e u tʃ i] “all"
Aim for high coverage	*Example*	
Each language should have about the same number of words recorded across the wordlist. A high mutual coverage is important to allow algorithms to find enough information to determine the major signal.	**NOT:** **BUT:**	L_1 150, L_2 50 L_1 200, L_2 200
No mixing of data from different sources	*Example*	
Mixing data for the same language from various sources can lead to inconsistencies in the phonetic representation of words, even if they are all given in plain phonetic transcriptions. This will weaken the evidence for regular sound correspondences.	**NOT:** **BUT:**	L_1=Source$_1$+Source$_2$ L_1=Source$_1$, L_2=Source$_2$
Evade synonyms	*Example*	
Languages often have multiple words for a given meaning. However, these can cause problems for sequence comparison and further downstream analyses like phylogenetic reconstruction. Having abundant synonyms in the data (e.g., 40 words for *snow*) will necessarily blur this signal.	**NOT:** **BUT:**	Tahitian [tai] ‘sea’, [moana] ‘ocean’ Tahitian ‘sea’

Table 2: SANE Rules for Data Sanity

4.1 Pairwise Alignment Analyses

Pairwise alignment analyses in biology and computer science date back to the 1970s when scholars like Needleman and Wunsch (1970), and Wagner and Fischer (1974) proposed algorithms based on the *dynamic programming paradigm* (Eddy, 2004b) which drastically reduced the computation time for the task of aligning two sequences with each other. The basic idea of the algorithms by Needleman & Wunsch and Wager & Fischer was to split the problem of finding one optimal alignment between two sequences into subparts and building the general solution from optimal alignments of smaller subsequences (Durbin et al., 2002, 19).[3]

The major parameters of pairwise alignment algorithms are the *scoring function*, the *gap function* and the *alignment mode*. The scoring function (Figure 3A, Tutorial: 3.1.1) determines how the matching of segments is penalized (or favored). In biology, it is well-known that amino-acid mutations follow certain transition preferences. The scoring function defines transition probabilities for each segment pair, and biologists make use of a large number of empirically derived scoring functions (Eddy, 2004a). In linguistics, on the other hand, we know well that certain sounds are more likely to occur in correspondence relations with each other (Dolgopolsky, 1964; Brown et al., 2013), and this knowledge can be used as a proxy when designing a scoring function in linguistics. While biology deals with small alphabets, in linguistics, the numbers of possible sounds in the languages of the world amounts to the thousands (Moran et al., 2014). It is not practical to design a matrix containing and confronting all sounds with each other, and most algorithms reduce the size of the alphabet by lumping similar sounds into a set of pre-defined sound classes (Figure 3B, Tutorial: 3.1.2), for which transition probabilities can be efficiently

[3] It would go beyond the scope of this tutorial to explain these famous algorithms in all detail. Instead we refer the readers to Kondrak (2002, 20-65) as well as an interactive demo of the Wagner-Fischer algorithm in List (2016).

A Scoring Function
Determines how segments are compared with each other. Most generally represented as a symmetric matrix of transition probability scores. Scores between segments are usually given in logarithmic scale, with unexpected matches being smaller 0 and expected ones being greater 0. On the right, a short scoring matrix is shown, in which matches between vowels (a) and consonants (p, f, h) are not allowed and therefore assigned high negative values.

	p	**f**	**h**	**a**
p	10	6	2	-10
f	6	10	6	-10
h	2	6	10	-10
a	-10	-10	-10	10

B Sound Classes
To reduce the huge number of sounds in phonetic alignment analyses, sounds are clustered into classes, assuming that correspondences inside a class are more likely than outside a class. Scoring functions are defined for sound classes, and sound sequences are converted to sound-class sequences before aligning them. On the right, it is shown, how Austral "you" can be rendered in different sound-class alphabets.

IPA	ʔ	oː	ɢ	u	a
Dolgo	H	V	K	V	V
SCA	H	U	K	Y	A
ASJP	7	o	q	u	a

C Gap Function
Gaps are introduced in alignments if a given element cannot be matched with any other element. Gap penalties can be defined globally, by assigning the same penalty to any segment of a given sequence, or individually, based on the context in which the segment occurs. In phonetic alignment analyses, individual gap penalties can be derived from *prosodic profiles* which reflect the relative sonority of each segment in a sequence. On the right, it is shown, how individual gap penalties are derived for the Austral "you".

IPA	ʔ	oː	ɢ	u	a
Sonority	1	7	1	7	7
Prostring	#	V	C	V	>
Weights	2.0	1.5	1.75	1.3	0.8
Gap-Scores	-4	-3	-3.5	-2.6	-1.6

D Alignment Mode
Alignment modes determine how sequences are compared. Global alignment compares the sequences entirely, assuming that they are indeed comparable in all their parts. Local alignment seeks to find the most similar subsequence of an alignment and deliberately strips off prefixes or suffixes. Semi-global alignment can only ignore prefixes or suffixes in one of the sequences. If the other sequence also has a suffix, it needs to strip it off. On the right, it is shown, how the different alignment modes account for the phonetic alignment of "you (dual)" in Ra'ivavae and Mangareva. While the local and the semi-global alignment fail to identify the regular sound correspondence of glottal stop and [k], the global alignment correctly matches words. This does not mean, however, that global alignment always yields the best solutions. Unaligned parts are shaded gray.

Global Alignment							
Ra'ivavae	ʔ	oː	ɢ	u	a		
Mangareva	k	oː	r	u	a		
Local Alignment							
Ra'ivavae	(ʔoː)	ɢ	-	-	u	a	
Mangareva	-	k	oː	r	u	a	
Semi-Global Alignment							
Ra'ivavae	ʔ	oː	ɢ	-	-	u	a
Mangareva	-	-	k	oː	r	u	a

Figure 3: Basic parameters and concepts in pairwise alignment analyses

defined, and which are then given as input for the alignment algorithm (List, 2012a; Holman et al., 2008).

The introduction of gaps in an alignment (Figure 3C, Tutorial: 3.1.3) can be seen as a special case of a scoring function. Instead of comparing two segments, the algorithm checks whether the introduction of a gap might be preferable. While gaps were originally given the same penalty, independent of the element with which they were compared, later studies showed that they could even be individually adjusted for each position in a sequence (Thompson et al., 1994). In linguistics, we know that sounds in certain positions (like initial consonants) are less likely to be lost and that new sounds tend to appear in specific contexts as well. In LingPy, position-specific gap penalties are derived from the *prosodic profiles* of sequences (List, 2012a). Prosodic profiles essentially reflect for each segment of a word whether it occurs in weak or strong prosodic positions, and the user-defined gap penalty is modified accordingly.

The alignment mode (Figure 3D, Tutorial: 3.1.4) basically determines which parts of individual sequences are compared. It is often impossible to compare two words entirely. Instead, we compare only certain parts of which we know that they are cognate, ignoring parts of which we know they are not. Since the same problem occurs when comparing the genes of diverse species in bioinformatics, biologists have long since been working on solutions, reflected in local alignment analyses (Smith and Waterman, 1981) in which only the most similar parts of sequences are compared (see Figure 3), while the rest is ignored, or *semi-global alignments* (Durbin et al., 2002, 26f).

What should users keep in mind when carrying out pairwise alignment analyses? As a rule of thumb, we recommend caution with local alignment analyses, since these can show unexpected behaviour. We also recommend care with custom-changes applied to the scoring or the gap function. Users often naively think that just “telling" the computers which sound changes they should assume would automatically lead to excellent alignments and at times complain that LingPy’s standard algorithms fail to “detect certain obvious changes". However, alignments are no way to determine sound changes, they are at best a first step for linguistic reconstruction, and none of the algorithms which have been proposed so far models any kind of change. What is modeled instead are *correspondences* of sounds. It is difficult, if not impossible, to design an algorithm that aligns sequences of all kinds of diversity without proposing certain analyses which look awkward to a trained linguist. But remember: automatic sequence comparison is not there to replace the experts, but to help them.

4.2 Multiple Alignment Analyses in Linguistics

Pairwise alignments are crucial for most automatic cognate detection methods (List, 2014b; Jäger et al., 2017). In order to visualize cognate judgments, or to reconstruct proto-forms, however, pairwise alignments are not of great help, as most linguistic research applies to at least three if not more language varieties. It may sound counter-intuitive for readers not familiar with the major workflows for automatic cognate detection that pairwise alignments are mainly used to detect cognates across multiple languages, while multiple alignments are only later computed from existing cognate sets. Why not compute multiple alignments right from the beginning, as for example, proposed by Wheeler and Whiteley (2015)? The reason for this workflow is that alignments only make sense when representing cognate words – aligning unrelated words just leads to chance similarities.

For reasons of algorithmic complexity, pairwise alignment algorithms cannot simply be rewritten to account for an arbitrary number of sequences. In order to address this problem, early approaches used heuristics that approximate optimal multiple alignments (Feng and Doolittle, 1987; Thompson et al., 1994). Most of these algorithms compute pairwise alignments in a first step and then combine the data in a pairwise fashion until all alignments are merged into one multiple alignment. The easiest way to do so is with help of a *guide tree*, a clustering of all sequences, which determines in which order sequences are merged with each other. This procedure is illustrated in Figure 4 for the alignment of four words for ‘dog’ in four Polynesian languages (Tutorial: 3.2).

Many extensions of the classical guide-tree heuristics have been proposed in the biological literature (Notredame et al., 2000; Morgenstern et al., 1998) and also adapted in linguistic appli-

cations (List, 2012a; Jäger and List, 2015; Hruschka et al., 2015). While the fine-tuning of the algorithms may have a solid impact on multiple alignment analyses involving large sets of language varieties, as we often encounter in dialectology (compare the results of Prokić et al. 2009 compared to List 2012a), the problem of erroneous alignments is much less pronounced when using smaller datasets and working in workflows which start from cognate detection and compute multiple alignments in a later stage. For these reasons, we refrain from giving more detailed descriptions of multiple sequence alignment here, but instead refer the readers to the literature that we quoted in this section and the examples in the interactive tutorial (Tutorial: 3.2).

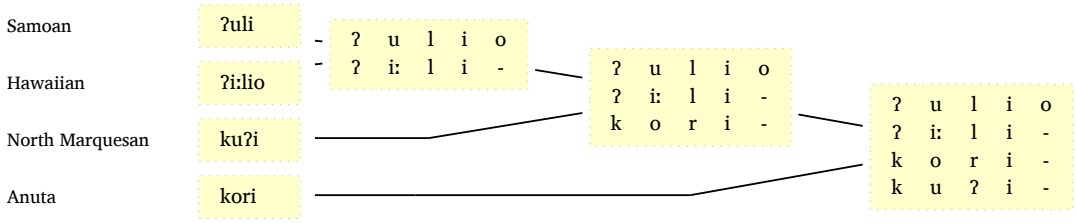

Figure 4: Combining words for ‘dog’ in Samoan, Hawaiian, North Marquesan, and Anuta into a multiple alignment with help of a guide tree

5 Automatic Cognate Detection

As mentioned in the previous section, we can only meaningfully align words if we know they are historically related. In order to identify which words *are* related, however, we still need to compare them, and most automatic approaches, including the core methods available in LingPy, make use of pairwise sequence comparison techniques in order to find historically related words in linguistic datasets.

The basic workflow of most automatic cognate detection methods can be divided into two major steps. In the first step, pairwise alignment is used to align all words to retrieve distance scores for each pair of words in the data which occur in the same concept slot. If normalized, distance scores typically rank between 0 and 1, with 0 indicating the identity of the objects under comparison, and 1 indicating the maximal difference that can be encountered for the objects. In a second step, these distances are used to *partition* the words into presumable cognate sets using tree- or network-based partitioning algorithms. If we take five words for “neck" from our Polynesian data, Ra‘ivavae [ʔagapoʔa], Hawaiian [ʔaːʔiː], Mangareva [kaki], Maori [ua], and Rapanui [ŋao], for example, we can use the *normalized edit distance* to compare all four words with each other and write the results into a matrix, as shown in Table 5A.[4]

In Table 5B, we have carried out the same pairwise comparison, but this time with a different sequence comparison measure, following the sound-class based alignment method (SCA, List 2012a), in which the idea of sound classes is combined with sequence alignment methods. Table 5C shows the results retrieved from the LexStat method (List 2012b) which derives distances

[4] In the *normalized edit distance* (NED), the *edit distance* between two strings is further normalized by dividing it by the length of the longer string. In this way, we can control for the length of the compared sequences.

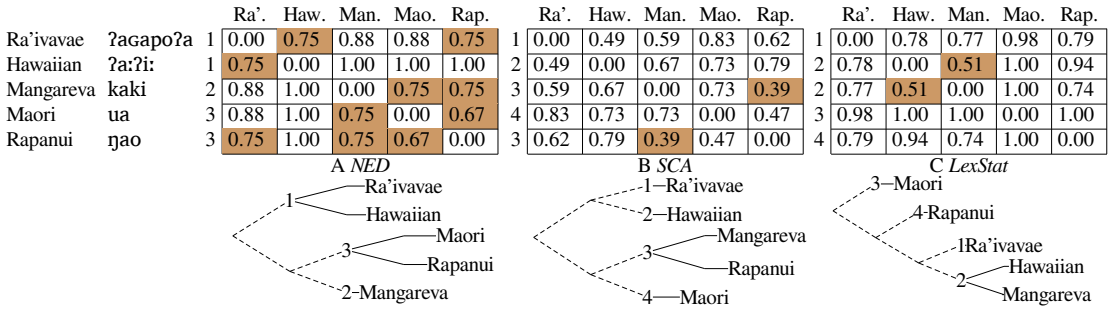

Figure 5: Contrasting distances retrieved from three different alignment approaches for Polynesian words for "neck". Cells shaded in brown indicate that distances are smaller than the default threshold for the algorithms. The first column of each table indicates the cognate decisions resulting from the matrix and the threshold. How these cognate decisions are determined is further illustrated in the trees below each matrix. They show how a flat cluster algorithm which stops once a certain threshold is reached can be used to partition the words into cognate sets.

from a previous search for regular sound correspondences. As can be seen, when comparing only the matrices, the methods generally differ in the way they handle sequence similarities. While NED has rather high scores which do not vary much from each other, SCA has consistently smaller scores with more variation, and LexStat has higher scores but more variation than NED.

In the second step, the matrix of word-pair distances is used to partition the words into cognate sets. For this, partitioning algorithms are used which split the words into cognate sets by trying to account as closely as possible for the pairwise distances of all words in a given meaning slot. Early approaches were based on a flat version of the well-known UPGMA algorithm (Sokal and Michener, 1958), which is an agglomerative cluster algorithm that returns the data in form of a tree. The flat variant of UPGMA stops merging words into bigger subgroups once a user-defined threshold of average pairwise distances among the words in each cluster has been reached (List, 2012b). In order to show how algorithms arrive from pairwise distance scores in a matrix at cognate set partitions, we provide a concrete example in Figure 5. First, we have marked all cells in which the distance is smaller than the recommended threshold for each method (following List et al. 2017b).[5] Second, we added guide trees (reflecting the clustering proposed when applying the UPGMA algorithm without stopping it earlier) below each matrix, which show how the flat clustering algorithm proceeds. If the algorithm stops grouping words into a given cluster, because the average threshold has been reached, this is indicated by a dashed line, which indicates how the clustering would have proceeded if the algorithm had not stopped. Given that we know that of these five words in the figure, only Hawaiian [ʔaːʔiː] and Mangareva [kaki] *are* cognate, we can immediately see that the LexStat algorithm is proposing the correct cognates in this example.

The performance of LexStat is not surprising, if we take its more sophisticated working procedure into account. LexStat uses global and local pairwise alignments to preanalyze the data,

[5]The threshold for the algorithms are: NED: 0.75, SCA: 0.45, LexStat: 0.6.

A Pairwise Distance Calculation

In order to calculate distances between words, different methods can be used. A simple way is to count, how often two two aligned strings show different characters (*edit distance*). By dividing this number by the length of the alignment, we can normalize this distance so that it ranges between 0 and 1. More sophisticated ways consist in computing the alignment scores (which are rendered as similarities) and converting these to distance scores using a formula for normalization (Downey et al. 2008).

Hawaiian	ʔ	a:	ʔ	i:	
Mangareva	k	a	k	i	
	1 +	1 +	1 +	1	= 4 / 4 = 1.0
Mangareva	k	a	k	i	
Rapanui	ŋ	a	-	o	
	1 +	0 +	1 +	1	= 3 / 4 = 0.75

B Flat Clustering (Partitioning)

Algorithms for flat clustering (also called partitioning algorithms) take a matrix of pairwise distances between objects as input and return a partition of the objects in which each object is assigned to exactly one group. As a common notation, we can assign each object a numeric ID. In this way we can easily compare to which degree different partition proposals for the same data differ.

Objects	**Part. 1**	**Part. 2**	**Part. 3**
A	1	1	1
B	1	2	1
C	2	2	1

C Permutation Test

It is not possible to use simple combinatorics to predict how many sound correspondences we would expect if all languages in our data were unrelated, since we do not know the underlying phonotactics of our languages. But we can compute the sound correspondences by shuffling our data multiple times, comparing words expressing different concepts, and counting how many correspondences we find for presumably unrelated word pairs. The table on the right shows correspondence distributions and the resulting LexStat score for some sounds in Sikaiana and Tuamotuan.

Sikaiana		**Tuamotuan**		**Distributions**		
IPA	**Class**	**IPA**	**Class**	**Att.**	**Exp.**	**Score**
f, v	B	f, v	B	10.0	1.1	7.32
f, v	B	k	K	0	1.8	-3.33
h	H	h, ʔ	H	28.0	8.4	5.69
h	H	k	K	1.0	4.2	-3.33

Figure 6: Some basic concepts important for automatic cognate detection

computing *language-specific* scoring functions (List, 2012b), in which the similarity of the segments in a given language pair depends on the overall number of matches that could be found in the preprocessing stage.[6] In these scoring functions, sound segments for all languages in the data are represented as sound-class strings in a certain prosodic environment. This representation is useful to handle sound correspondences in different contexts (word-initial, word-final, etc.). For each language pair in the data, *LexStat* creates an *attested* and an *expected* distribution of sound correspondences. The attested distribution is computed for words with the same meaning and whose SCA-score is beyond a user-defined threshold. The expected distribution is computed by shuffling the word lists in such a way that words with different meanings are aligned and compared, with the users defining, how often word lists should be shuffled. This *permutation test* following suggestions by Kessler (2001) makes sure that the sound correspondences identified are unlikely to have arisen by chance. The distributions resulting from this permutation test are then combined in *log-odds* scores (see Figure 3 above) which can then in turn be used to re-align all words and determine their *LexStat*-distance.[7] These scores are then again used to create a matrix of pairwise distances as shown in Figure 5. Our interactive tutorial shows how input data can be quickly checked before carrying out the (at times time-consuming) computa-

[6] For an example, consider the matches between Sikaiana and Tahitian shown in Table 1. Although Sikaiana [k] is different from [ʔ], they are similar from a language-specific perspective, since they recur across many aligned cognate sets between both languages. When comparing [k] in English with [ʔ] in German, however, they are not similar, as we won't find a cognate set in which those two sounds correspond.

[7] As alignment algorithms yield similarity scores as a default, the similarity scores are converted to distance scores with help of the formula proposed by Downey et al. (2008).

tion (Tutorial: 4.1) and provides additional information regarding the differences between the cognate detection methods available in LingPy (Tutorial: 4.2) and illustrates in detail how each of them can be applied (Tutorial: 4.3).

More recent approaches for cognate set partitioning use Infomap (Rosvall and Bergstrom, 2008), a *community detection* algorithm which uses random walks in a graph representation of the data to identify those clusters in which significantly more edges can be found inside a group than outside (Newman, 2006). In order to model the data as a graph, words are represented as nodes and distances between words are represented as edges which are drawn between all nodes whose pairwise distance is beyond a user-defined threshold (List et al., 2017b). Recent studies have shown that the graph-based partitioning approaches slightly outperform the flat agglomerative clustering procedures (List et al., 2017b, 2016b; Jäger et al., 2017).

The advantage of *LexStat* and similar algorithms is that the algorithm infers a lot of information from the data itself. Instead of assuming language-independent distance scores which would be the same for all languages in the world, it essentially infers potential sound correspondences for each language pair in separation and uses this information to determine language-specific distance scores. The disadvantages of *LexStat* are the computation time and the dependency of data with high mutual coverage. It was designed in such a way that it refuses to cluster words into cognate sets if sufficient information is lacking. As a rule of thumb, derived from earlier studies (List, 2014a), we recommend applying *LexStat* only if the basic concept lists of a given dataset consists of at least 200 words, and if the mutual coverage of the data exceeds 150 word pairs. If the data is too sparse, such as, for example, in the ASJP database (Wichmann et al., 2016) which gives maximally 40 concepts per language, we recommend to use either the SCA approach, or to turn to more sophisticated machine learning approaches (Jäger et al., 2017), which have been designed and trained in such a way that they yield their best scores on smaller datasets. In all cases, users should be aware that the algorithms may fail to detect certain cognates. The reasons range from rare sound correspondences which can trigger problematic alignments, via sparseness of data (especially when dealing with divergent languages), up to problems of morphological change which may easily confuse the algorithms as they may yield partial cognates and produce words that cannot be fully aligned anymore (List et al., 2017b). In Table 3, we summarize some basic differences between the four methods mentioned so far.

Method	**Scoring Function**	**Sound Classes**	**Gap Function**	**Alignment Mode**	**Partitioning**
NED	identity	-	-	global	flat UPGMA
SCA	language-independent	SCA-model	prosodic profiles	global	flat UPGMA
LexStat	language-specific	SCA-model	prosodic profiles	semi-global	flat UPGMA
LexStat-Infomap	language-specific	SCA-model	prosodic profiles	semi-global	Infomap

Table 3: Comparing different algorithms for cognate detection implemented in LingPy with respect to some fundamental parameters of sequence comparison.

Once the words have been clustered into cognate sets, it is advisable to align all cognate words with each other, using a multiple alignment algorithm (Tutorial: 4.4). Alignments are useful in multiple ways. First, users can easily inspect them with web-based tools (Tutorial: 4.5). Second, they can be used to statistically investigate the identified sound correspondence patterns in the data (see Tutorial: 4.6). Both the manual and the automatic check of the results provided by

automatic cognate detection methods are essential for a successful application of the methods. Only in this way users can either convince themselves that the results come close to their expectations or that something weird is going on. In the latter situation, we recommend that users thoroughly check to which degree they have conformed to our SANE rules for dataset sanity outlined above in Sec. 3. We also recommend that users do not change the different parameters too much, especially when applying LingPy the first time. Instead of trying to fix minor errors (such as obvious cognates missed or lookalikes marked as cognates) by changing parameters, it is often more efficient to correct errors manually. Although Rama et al. (2018) report promising results on fully automated workflows, we do not recommend relying entirely on automatic cognate detection when it comes to phylogenetic reconstruction, since the algorithms tend to be too conservative, often missing valid cognates (List et al., 2017b), but we are confident enough to recommend it for initial data exploration, and for the preparsing of data in order to increase the efficiency of cognate annotation.

6 Evaluation

We have claimed above that automatic cognate detection had made great progress of late. We make this claim based on tests in which the performance of automatic cognate detection algorithms was compared with expert cognate judgments (List et al., 2017b). There are different ways to compare expert cognate judgments with algorithmic ones. A very simple but nevertheless important first one is to compare different cognate judgments manually, by eyeballing the data. Even if one lacks expert cognate judgments for a given dataset, this may be useful, as it helps to get a quick impression on potential weaknesses of the algorithm used for a given analysis. Comparing cognate judgments in concrete, however, can be quite tedious, especially if the data are not presented in any ordered fashion. For this reason, LingPy offers a specific format that helps to compare different cognate judgments in a rather convenient way. How this comparison can be carried out is illustrated in Table 4, where we use the numeric annotation for cognate clusters as described in Figure 6 to compare expert cognate judgments for 'to turn' in eight East Polynesian language with those produced by edit distance, the SCA, and the LexStat method, respectively. As can be seen from the table, NED lumps all words into one cluster, obviously being confused by the similarity of the vowels across all words. SCA comes close to the expert annotation, but wrongly separates Hawaiian [wili] from the first cluster, obviously being confused by the dissimilarity of the sound classes. LexStat correctly identifies all cognates, obviously thanks to its initial search for language-specific similarities between sound classes. In the interactive tutorial, we show how users can compute similar overviews on differences in cognate detection analyses and conveniently compare them (Tutorial: 5.1).

While manual inspection is important it is also crucial to have an independent and objective score that tells us how well algorithms perform on a given dataset. Knowing the approximate performance may for example be useful when working with large datasets which would take too long to be analysed manually. If we annotate part of the data and see that the automatic methods perform well enough, we could then use the automatic approaches to carry out our analyses and report the expected accuracy in the study. Our recommended evaluation measure are B-Cubed scores (Bagga and Baldwin, 1998; Amigó et al., 2009) which Hauer and Kondrak (2011) first

Doculect	Form	Expert	NED	SCA	LexStat
Ra‘ivavae	taːviɢi [4580]	1	1	1	1
Hawaiian	wili [5835]	1	1	4	1
North-Marquesan	kaviʔi [3575]	1	1	1	1
Rapanui	taviri [1838]	1	1	1	1
Hawaiian	huli [5834]	2	1	2	2
Maori	huri [936]	2	1	2	2
Sikaiana	tahuli [3283]	2	1	2	2
Mangareva	tiːrori [2101]	3	1	3	3

Table 4: Comparing automatic cognate detection methods with expert cognate judgments for words for ‘to turn’ in East Polynesian languages.

introduced as a measure to assess the quality of cognate detection algorithms compared to expert judgments.

The details of how B-Cubed scores are computed are explained elsewhere in detail (List et al., 2017b), and it would go beyond the scope of this tutorial to introduce them here again. For users interested in automatic cognate detection, but reluctant to learning in depth about evaluation measures in computational linguistics, it is sufficient to know how the B-Cubed scores should be interpreted. Usually the scores are given in three forms, which all rank between 0 and 1: *precision*, *recall*, and *F-Score*. Precision comes closest to the notion of *true positives* in historical linguistics. Recall is close to the notion of *true negatives*, accordingly, and the F-Score, the harmonic mean of precision and recall, can be seen as a general summary of the two, derived by the formula $2\frac{P \times R}{P+R}$, where P is the precision and R is the recall. If the scores are high, this means the algorithms come close to the judgment of the experts, a score of 1.0 in precision and recall (and therefore also the F-Score) means that the results are 100% identical.

	RANDOM	NED	SCA	LexStat	LS-Infomap
Precision	0.47	0.81	0.88	0.95	0.94
Recall	0.73	0.96	0.84	0.92	0.93
F-Score	0.57	0.88	0.86	0.93	0.94

Table 5: B-Cubed scores for different cognate detection algorithms compared against a test set of East Polynesian languages

In Table 5 we report the results achieved by four automatic cognate detection methods on a small subset of ten East Polynesian languages which we retrieved from our Polynesian dataset for illustrative purposes.[8] In addition to the three methods reported already in Table 4, we added a random cognate detector which was sampled from 100 trials, and the Infomap version of the

[8] We have not fully explored the practical limitations in terms of number of languages or number of concepts when comparing languages with LingPy. Jäger et al. (2017) and Rama et al. (2017) report successful applications of LingPy’s cognate detection algorithms for as many as 100 languages. Although we think that the number might in fact be even higher, based on tests we carried out ourselves on 150 and more languages, we recommend to be

LexStat algorithm (`LS-Infomap`), in which the cognate set partitioning is carried out with the Infomap algorithm instead of the flat version of UPGMA (see Sec. 5 above).[9] NED shows a rather low precision compared to the other non-random approaches, indicating that it proposes many false positives (as we could see above in Table 4). On the other hand, its recall is very high, indicating that it does not miss many cognate sets. SCA obviously has a lot of problems with the data, performing worse than NED in general, with a rather low precision and recall. Both LexStat approaches largely outperform the other approaches in general, and especially the very high precision is very comforting, since it indicates that the algorithms do not propose too many false positives. That the Infomap version of LexStat performs better than LexStat with UPGMA is also shown in this comparison, although the differences are much lower than reported in List et al. (2017b). It would be very interesting to compare the scores we achieved with general scores of levels of agreement among human experts. Unfortunately, no systematic study has been carried out so far.[10] The interactive tutorial gives a detailed introduction into the computation of B-Cubed scores with LingPy (Tutorial: 5.2).

Given the differences in the results regarding precision, recall, and generalized F-scores, it is obvious that the choice of the algorithm to use depends on the task at hand. If users plan to invest much time into manual data correction, having an algorithm with high recall that identifies most of the cognates in the data while proposing a couple of erroneous ones is probably the best choice. Users can achieve this by choosing a high threshold or an algorithm such as NED, which yields a rather high recall in form of the B-Cubed scores, at least for the Polynesian data in our sample. In other cases, however, when user-correction is not feasible because of the size of the dataset, it is useful to choose low thresholds or generally conservative algorithms with high B-Cubed precision in order to minimize the amount of false positives.

7 Data Export

LingPy provides direct export of the cognate judgments to the Nexus format (Maddison et al., 1997), allowing users to analyse automated cognate judgments with popular packages for phylogenetic reconstruction, such as SplitsTree (Huson, 1998), MrBayes (Ronquist et al., 2009), or BEAST 2 (Bouckaert et al. 2014, see Tutorial: 6.1). If phylogenetic trees are computed from distance matrices, both matrices and trees can be written to file and further imported in software packages for tree manipulation and visualization (Tutorial: 6.2). In addition, data can be exported (and also be imported) to the wordlist format proposed by the CLDF initiative (Forkel et al., 2017), which is intended to serve as a generic format for data sharing in cross-linguistic studies (Tutorial: 6.3).

careful when analysing too many languages, as algorithmic performance may drastically drop when investigation samples that are too large.

[9] The threshold for LexStat-Infomap was set to 0.55, following List et al. (2017b). The random cognate annotation algorithm was designed in such a way that it has the tendency to lump cognates to larger clusters.

[10] The only study known to us addressing these problems is Geisler and List (2010), but it has, unfortunately, not been sufficiently quantified.

8 Concluding Remarks

In this tutorial we have tried to show how automatic sequence comparison in LingPy can be carried out. Given the scope of this paper, it is clear that we could not cover all aspects of alignments and cognate detection in all due detail. We hope, however, that we could help readers understand what they should keep in mind if they want to carry out sequence comparison analyses on their own. Additional questions will be answered in an interactive tutorial supplemented with this paper, and for deeper questions going beyond the pure application of sequence comparison algorithms – such as additional analyses (e.g., the *minimal lateral network* method for borrowing detection, List et al. 2014, or an algorithm for the detection of partial cognates, List et al. 2016b), routines for plotting and data visualization, or customization routines for user-defined sound-class models. – we recommend the readers to turn to the extensive online documentation of the LingPy package (`http://lingpy.org`). We have emphasized multiple times throughout this paper that the algorithms cannot and should not be used to replace trained linguists. Instead, they should be seen as a useful complement to the large arsenal of methods for historical language comparison which can help experts to derive initial hypotheses on cognacy, speed up tedious the annotation of cognate sets, and increase their efficiency and consistency.

Supplementary Material

The supplementary material contains the code, the data, and the extended interactive tutorial in form of an Ipython (Jupyter) notebook and as HTML document and can be accessed via GitHub (`https://github.com/lingpy/lingpy-tutorial`), Zenodo (`https://zenodo.org/record/1252231`), and the Open Science Framework (`https://osf.io/xc7tj/`). Updates to this tutorial will appear on GitHub and released to Zenodo. The tutorial can also be accessed online (`http://htmlpreview.github.io/?https://github.com/lingpy/lingpy-tutorial/blob/master/notebook.html`).

Acknowledgments

This research was supported by the ERC Starting Grant "Computer-Assisted Language Comparison" (Grant CALC 715618, JML, TT) and the Australian Research Council's Centre of Excellence for the Dynamics of Language (Australian National University, Grant CE140100041, SJG). As part of the GlottoBank project (`http://glottobank.org`), this work was further supported by the Department of Linguistic and Cultural Evolution of the Max Planck Institute for the Science of Human History (Jena) and the Royal Society of New Zealand (Marsden Fund, Grant 13-UOA-121). We are grateful to three anonymous reviewers for challenging and constructive advice. We thank Russell Gray for encouraging and helpful advice during the preparation of this project, and to Christoph Rzymski for helpful comments on an earlier version of this draft.

References

Enrique Amigó, Julio Gonzalo, Javier Artiles, and Felisa Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. *Information Retrieval*, 12(4):461–486, 2009. ISSN 1386-4564.

Amit Bagga and Breck Baldwin. Entity-based cross-document coreferencing using the vector space model. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*, pages 79–85. Association of Computational Linguistics, 1998.

Remco Bouckaert, Joseph Heled, Denise Kühnert, Tim Vaughan, Chieh-Hsi Wu, Dong Xie, Marc A. Suchard, Andrew Rambaut, and Alexei J. Drummond. BEAST 2: A Software Platform for Bayesian Evolutionary Analysis. *PLoS Computational Biology*, 10(4):e1003537, 2014. doi: 10.1371/journal.pcbi.1003537.

Cecil H. Brown, Eric W. Holman, Søren Wichmann, Viveka Velupillai, and Michael Cysouw. Automated classification of the world's languages. *Sprachtypologie und Universalienforschung*, 61(4):285–308, 2008.

Cecil H. Brown, Eric W. Holman, and Søren Wichmann. Sound correspondences in the world's languages. *Language*, 89(1):4–29, 2013.

Lyle Campbell. Comment on: Automated dating of the world's language families based on lexical similarity. *Current Anthropology*, 52:866–867, 2011.

R. B. Dixon and A. L. Kroeber. *Linguistic families of California*. University of California Press, Berkeley, 1919.

Aron B. Dolgopolsky. Gipoteza drevnejšego rodstva jazykovych semej Severnoj Evrazii s verojatnostej točky zrenija. *Voprosy Jazykoznanija*, 2:53–63, 1964.

Sean S. Downey, Brian Hallmark, Murray P. Cox, Peter Norquest, and Stephen Lansing. Computational feature-sensitive reconstruction of language relationships: developing the ALINE distance for comparative historical linguistic reconstruction. *Journal of Quantitative Linguistics*, 15(4):340–369, 2008.

Michael Dunn. *Indo-European lexical cognacy database (IELex)*. Max Planck Institute for Psycholinguistics, Nijmegen, 2012. URL `http://ielex.mpi.nl`.

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchinson. *Biological sequence analysis. Probabilistic models of proteins and nucleic acids*. Cambridge University Press, Cambridge, 7 edition, 2002.

Sean R. Eddy. Where did the BLOSUM62 alignment score matrix come from? *Nature Biotechnology*, 22(8):1035–1036, 2004a.

Sean R. Eddy. What is dynamic programming? *Nature Biotechnology*, 22(7):909–910, 2004b.

D. F. Feng and R. F. Doolittle. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *Journal of Molecular Evolution*, 25(4):351–360, 1987.

Robert Forkel, Johann-Mattis List, Michael Cysouw, and Simon J. Greenhill. *CLDF. Cross-Linguistic Data Formats. Version 1.0*. Max Planck Institute for the Science of Human History, Jena, 2017. doi: 10.5281/zenodo.1117644. URL `https://doi.org/10.5281/zenodo.1117644`.

Anthony Fox. *Linguistic reconstruction*. Oxford University Press, Oxford, 1995. ISBN 0-19-870000-8.

Hans Geisler and Johann-Mattis List. Beautiful trees on unstable ground. Notes on the data problem in lexicostatistics. In Heinrich Hettrich, editor, *Die Ausbreitung des Indogermanischen. Thesen aus Sprachwissenschaft, Archäologie und Genetik*. Reichert, Wiesbaden, 2010. Document has been submitted in 2010 and is still waiting for publication.

Simon Greenhill. Levenshtein distances fail to identify language relationships accurately. *Computational Linguistics*, 37(4):689–698, 2011.

Simon J. Greenhill, Robert Blust, and Russell D. Gray. The Austronesian Basic Vocabulary Database: From bioinformatics to lexomics. *Evolutionary Bioinformatics*, 4:271–283, 2008.

S. C. Gudschinsky. The ABC's of lexicostatistics (glottochronology). *Word*, 12(2):175–210, 1956.

Harald Hammarström, Robert Forkel, and Martin Haspelmath. *Glottolog*. Max Planck Institute for Evolutionary Anthropology, Leipzig, 2017. URL `http://glottolog.org`.

M. Haspelmath and U. Tadmor. The Loanword Typology project and the World Loanword Database. In Martin Haspelmath and Uri Tadmor, editors, *Loanwords in the world's languages*, pages 1–34. de Gruyter, Berlin and New York, 2009.

Bradley Hauer and Grzegorz Kondrak. Clustering semantically equivalent words into cognate sets in multilingual lists. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*, pages 865–873. AFNLP, 2011.

Nathan W. Hill and Johann-Mattis List. Challenges of annotation and analysis in computer-assisted language comparison: A case study on Burmish languages. *Yearbook of the Poznań Linguistic Meeting*, 3(1):47–76, 2017.

Eric W. Holman, Søren Wichmann, Cecil H. Brown, Viveka Velupillai, André Müller, and Dik Bakker. Explorations in automated lexicostatistics. *Folia Linguistica*, 20(3):116–121, 2008.

Eric. W. Holman, Cecil H. Brown, Søren Wichmann, André Müller, Viveka Velupillai, Harald Hammarström, Sebastian Sauppe, Hagen Jung, Dik Bakker, Pamela Brown, Oleg Belyaev, Matthias Urban, Robert Mailhammer, Johann-Mattis List, and Dimitry Egorov. Automated dating of the world's language families based on lexical similarity. *Current Anthropology*, 52 (6):841–875, 2011.

D. J. Hruschka, S. Branford, E. D. Smith, J. Wilkins, A. Meade, M. Pagel, and T. Bhattacharya. Detecting regular sound changes in linguistics as events of concerted evolution. *Curr. Biol.*, 25(1):1–9, 2015.

Daniel H. Huson. SplitsTree: analyzing and visualizing evolutionary data. *Bioinformatics*, 14 (1):68–73, 1998.

International Phonetic Association IPA. *Handbook of the International Phonetic Association. A guide to the use of the international phonetic alphabet*. Cambridge University Press, Cambridge, 1999.

Gerhard Jäger. Support for linguistic macrofamilies from weighted alignment. *Proceedings of the National Academy of Sciences of the United States of America*, 112(41):12752–12757, 2015.

Gerhard Jäger and Johann-Mattis List. Factoring lexical and phonetic phylogenetic characters from word lists. In H. Baayen, G. Jäger, M. Köllner, J. Wahle, and A. Baayen-Oudshoorn, editors, *Proceedings of the 6th Conference on Quantitative Investigations in Theoretical Linguistics*, Tübingen, 2015. Eberhard-Karls University.

Gerhard Jäger, Johann-Mattis List, and Pavel Sofroniev. Using support vector machines and state-of-the-art algorithms for phonetic alignment to identify cognates in multi-lingual wordlists. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics. Long Papers*, pages 1204–1215, Valencia, 2017. Association for Computational Linguistics.

Alexei Kassian, George S. Starostin, A. Dybo, and Vasiliy Chernov. The Swadesh wordlist. An attempt at semantic specification. *Journal of Language Relationships*, 4:46–89, 2010.

Brett Kessler. *The significance of word lists*. CSLI Publications, Stanford, 2001.

Grzegorz Kondrak. A new algorithm for the alignment of phonetic sequences. In *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*, pages 288–295, 2000.

Grzegorz Kondrak. *Algorithms for language reconstruction*. Dissertation, University of Toronto, Toronto, 2002.

Roger Lass. *Historical linguistics and language change*. Cambridge University Press, Cambridge, 1997.

V. I. Levenshtein. Dvoičnye kody s ispravleniem vypadenij, vstavok i zameščenij simvolov. *Doklady Akademij Nauk SSSR*, 163(4):845–848, 1965.

Johann-Mattis List. SCA. Phonetic alignment based on sound classes. In Marija Slavkovik and Dan Lassiter, editors, *New directions in logic, language, and computation*, pages 32–51. Springer, Berlin and Heidelberg, 2012a.

Johann-Mattis List. LexStat. Automatic detection of cognates in multilingual wordlists. In *Proceedings of the EACL 2012 Joint Workshop of Visualization of Linguistic Patterns and Uncovering Language History from Multilingual Resources*, pages 117–125, Stroudsburg, 2012b.

Johann-Mattis List. Investigating the impact of sample size on cognate detection. *Journal of Language Relationship*, 11:91–101, 2014a.

Johann-Mattis List. *Sequence comparison in historical linguistics*. Düsseldorf University Press, Düsseldorf, 2014b.

Johann-Mattis List. Wagner-Fischer Demo. *figshare*, 2016. doi: http://dx.doi.org/10.6084/m9.figshare.3158836.v1. URL `https://figshare.com/articles/Wagner_Fischer_Demo/3158836`.

Johann-Mattis List. A web-based interactive tool for creating, inspecting, editing, and publishing etymological datasets. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics. System Demonstrations*, pages 9–12, Valencia, 2017. Association for Computational Linguistics.

Johann-Mattis List. Tossing coins: linguistic phylogenies and extensive synonymy. *The Genealogical World of Phylogenetic Networks*, 7(2), 2018. URL `http://phylonetworks.blogspot.de/2018/02/tossing-coins-linguistic-phylogenies.html`.

Johann-Mattis List, Shijulal Nelson-Sathi, Hans Geisler, and William Martin. Networks of lexical borrowing and lateral gene transfer in language and genome evolution. *Bioessays*, 36(2): 141–150, 2014.

Johann-Mattis List, Michael Cysouw, and Robert Forkel. Concepticon. A resource for the linking of concept lists. In Nicoletta Calzolari (Conference Chair), Khalid Choukri, Thierry Declerck, Marko Grobelnik, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis, editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation*, pages 2393–2400. European Language Resources Association (ELRA), 2016a.

Johann-Mattis List, Philippe Lopez, and Eric Bapteste. Using sequence similarity networks to identify partial cognates in multilingual wordlists. In *Proceedings of the Association of Computational Linguistics 2016 (Volume 2: Short Papers)*, pages 599–605, Berlin, 2016b. Association of Computational Linguistics.

Johann-Mattis List, Simon Greenhill, and Robert Forkel. *LingPy. A Python library for historical linguistics*. Max Planck Institute for the Science of Human History, Jena, 2017a. doi: https://zenodo.org/badge/latestdoi/5137/lingpy/lingpy. URL `http://lingpy.org`.

Johann-Mattis List, Simon J. Greenhill, and Russell D. Gray. The potential of automatic word comparison for historical linguistics. *PLOS ONE*, 12(1):1–18, 2017b.

Johann Mattis List, Michael Cysouw, Simon Greenhill, and Robert Forkel, editors. *Concepticon. A Resource for the linking of concept list*. Max Planck Institute for the Science of Human History, Jena, 2018. URL http://concepticon.clld.org/.

D. R. Maddison, D. L. Swofford, and W. P. Maddison. NEXUS: an extensible file format for systematic information. *Syst. Biol.*, 46(4):590–621, 1997.

James A. Matisoff. *The Sino-Tibetan Etymological Dictionary and Thesaurus project*. University of California, Berkeley, 2015.

Steven Moran and Michael Cysouw. *The Unicode Cookbook for Linguists: Managing writing systems using orthography profiles*. Zenodo, Zürich, 2017. doi: 10.5281/zenodo.290662. URL https://doi.org/10.5281/zenodo.290662.

Steven Moran, Daniel McCloy, and Richard Wright, editors. *PHOIBLE Online*. Max Planck Institute for Evolutionary Anthropology, Leipzig, 2014. URL http://phoible.org/.

B. Morgenstern, W. R. Atchley, K. Hahn, and A. Dress. Segment-based scores for pairwise and multiple sequence alignments. In Janice Glasgow, Tim Littlejohn, Francois Major Richard Lathrop, David Sankoff, and Christoph Sensen, editors, *Proceedings of the Sixth International Conference on Intelligent Systems for Molecular Biology*, pages 115 – 121, Menlo Park, 1998. AAAI Press.

Saul B. Needleman and Christan D. Wunsch. A gene method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

M. E. J. Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical Review E*, 74 (3):1–19, 2006. ISSN: 1539-3755, ISBN: 1539-3755.

Cédric Notredame, Desmond G. Higgins, and Jaap Heringa. T-Coffee. *Journal of Molecular Biology*, 302:205–217, 2000.

Fernando Pérez and Brian E. Granger. IPython: a system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29, 2007. ISSN 1521-9615. doi: 10.1109/MCSE.2007.53. URL http://ipython.org.

Jelena Prokić, Martijn Wieling, and John Nerbonne. Multiple sequence alignments in linguistics. In *Proceedings of the EACL 2009 Workshop on Language Technology and Resources for Cultural Heritage, Social Sciences, Humanities, and Education*, pages 18–25, 2009.

Taraka Rama, Johannes Wahle, Pavel Sofroniev, and Gerhard Jäger. Fast and unsupervised methods for multilingual cognate clustering. *CoRR*, abs/1702.04938, 2017. URL http://arxiv.org/abs/1702.04938.

Taraka Rama, Johann-Mattis List, Johannes Wahle, and Gerhard Jäger. Are automatic methods for cognate detection good enough for phylogenetic reconstruction in historical linguistics? In *Proceedings of the North American Chapter of the Association of Computational Linguistics*, 2018.

Frederik Ronquist, Paul van der Mark, and John P. Huelsenbeck. Bayesian phylogenetic analysis using MrBayes. In Philippe Lemey, Marco Salemi, and Anne-Mieke Vandamme, editors, *The phylogenetic handbook. A practical approach to phylogenetic analysis and hypothesis testing*, pages 210–266. Cambridge University Press, Cambridge, Second Edition edition, 2009.

M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proc. Natl. Acad. Sci. U.S.A.*, 105(4):1118–1123, 2008.

T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 1:195–197, 1981.

Robert. R. Sokal and Charles. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28:1409–1438, 1958.

J. D. Thompson, D. G. Higgins, and T. J. Gibson. CLUSTAL W. *Nucleic Acids Research*, 22 (22):4673–4680, 1994.

Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *Journal of the Association for Computing Machinery*, 21(1):168–173, 1974. ISSN 0004-5411.

W. C. Wheeler and Peter M. Whiteley. Historical linguistics as a sequence optimization problem: the evolution and biogeography of Uto-Aztecan languages. *Cladistics*, 31(2):113–125, 2015.

Søren Wichmann, Eric W. Holman, and Cecil H. Brown. *The ASJP database*. Max Planck Institute for the Science of Human History, Jena, 2016. URL `http://asjp.clld.org`.