Master Thesis

Submitted in partial fulfilment of the requirements for the degree of
Master of Arts in General Linguistics

Alignment and word comparison with Pair Hidden Markov Models

Author:
Johannes Wahle

1st Supervisor:
Prof. Dr. Gerhard Jäger
2nd Supervisor:
Dr. Armin Buch

Eberhard Karls Universität Tübingen
Seminar für Sprachwissenschaft
Allgemeine Sprachwissenschaft
August, 2013

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

Ich versichere, dass ich die Arbeit ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Stellen und Personen, welche mich bei der Vorbereitung und Anfertigung der Abhandlung unterstützten, wurden genannt und Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Tübingen, den September 16, 2013

Johannes Wahle

Er dachte eine Zeit nach. Dann sprach er weiter: »Man darf nie an die ganze Straße auf einmal denken, verstehst du? Man muss nur an den den nächsten Schritt denken, an den nächsten Atemzug, an den nächsten Besenstrich. Und immer wieder nur an den nächsten.«

Wieder hielt er inne und überlegte ehe er hinzufügte: »Dann macht es Freude; das ist wichtig, dann macht man seine Sache gut. Und so soll es sein.«

Und abermals nach einer langen Pause fuhr er fort: »Auf einmal merkt man, dass man Schritt für Schritt die ganze Straße gemacht hat. Man hat gar nicht gemerkt wie und man ist nicht außer Puste.«Er nickte vor sich hin und sagte abschließend: »Das ist wichtig.«

Michael Ende - Momo

Acknowledgments

I would like to take this occasion and thank some people who helped me in one or the other way during the whole process of writing this thesis.

I am especially grateful to my first supervisor Prof. Dr. Gerhard Jäger. His ongoing support, guidance and hints during the research and the writing phase helped me to see the whole topic much clearer and understanding the details better.
I want to thank my second supervisor Dr. Armin Buch as well. His hints and remarks on the structure of the thesis helped me a lot in making my thoughts more concrete. This suggestions have made the structure of this thesis much clearer to me.

I also want to thank:

Sarah Schulz for corrections as well as some help and guidance during the course of programming.
Niko Schenk for helpful suggestions on programming and server issues.
Johannes Dellert for helpful discussions about the topic and programming issues.
Martijn Wieling for providing some source code and some hints regarding data issues.
Johann-Mattis List for sharing his knowledge about alignments.

Additionally I am especially grateful to Marisa Delz, Steffen Suur-Nuuja and Peter Heinrich for their companionship, kindness, support during all of my studies.

Last but not least I would like to take the occasion and thank my family, especially my parents, and Angelina Linnemann for all their support.

Abstract

Comparing sequences is important for several fields of science; historical linguistics and computational biology are two of those. Computational biology has brought up *Pair Hidden Markov Models* to deal with this task. A Pair Hidden Markov Model is stochastic model for sequence comparison. The sequences which are compared in this field are DNA or protein sequences. Of course, there are no DNA or protein sequences in linguistics, but words can be seen as sequences of sounds. This offers a parallel between these two fields. Slightly modified Pair Hidden Markov Models can be used to compare sequences of sounds, i.e. words. The increasing amount of electronically available data has given rise to the necessity of automated techniques for sequence comparison in historical linguistics. Pair Hidden Markov Models offer the possibility to compare sequences automatically. Another algorithm for this purpose is the Needleman-Wunsch algorithm. This algorithm is designed to find the best alignment between two sequences.
As their name suggests, Pair Hidden Markov Models are developed from Hidden Markov Models. Therefore, algorithms developed for Hidden Markov Models can be transferred to Pair Hidden Markov Models. To test the performance of Pair Hidden Markov Models in sequence alignment, the alignment scores are used for a cognate classification task. The cognate clusters generated from the Pair Hidden Markov Model alignment scores are compared to cognate clusters based on Needleman-Wunsch alignment. In an eight-fold cross validation task the cognate clusters are compared. It turns out that Pair Hidden Markov Models outperform Needleman-Wunsch on the cognate classification task.
In comparison to the Needleman-Wunsch algorithm Pair Hidden Markov Models offer a variety of methods for sequence comparison. Hence, Pair Hidden Markov Models have to be taken into consideration for applications in historical linguistics where sequence comparison is important.

Zusammenfassung

Der Vergleich von Sequenzen aller Art ist in diversen wissenschaftlichen Bereichen wichtig. Historische Linguistik und Bioinformatik sind zwei dieser Bereiche. In der Bioinformatik wurden zu diesem Zweck *Pair Hidden Markov Models* entwickelt. Ein Pair Hidden Markov Model ist ein stochastisches Modell, um Sequenzen zu vergleichen. In der Bioinformatik werden DNA- und Proteinsequenzen verglichen. Jedoch gibt es in der Linguistik keine DNA- oder Proteinsequenzen, allerdings können Wörter als Sequenzen von Lauten aufgefasst werden. Diese Ähnlichkeit bildet eine Parallele zwischen der Bioinformatik und der Linguistik. Leicht verändert können Pair Hidden Markov Models dazu benutzt werden, Lautsequenzen, d.h. Wörter, zu vergleichen. Die wachsende Menge an elektronisch verfügbaren Daten machen automatische Methoden für den Vergleich von Sequenzen notwendig. Pair Hidden Markov Models bieten eine solche Möglichkeit. Eine andere Methode für solche Aufgaben ist der Needleman-Wunsch Algorithmus. Dieser Algorithmus wurde entwickelt, um das beste Alignment zwischen zwei Sequenzen zu finden.
Wie der Name suggeriert, sind Pair Hidden Markov Models eine Weiterentwicklung von *Hidden Markov Models*. Aus diesem Grund können Algorithmen, die für Hidden Markov Models entwickelt wurden, auf Pair Hidden Markov Models übertragen werden. Um die Leistungsfähigkeit von Pair Hidden Markov Models im Bereich der Sequenzalinierung zu testen, werden die Werte der einzelnen Alinierungen für Kognatenerkennung genutzt. Kognatgruppen, die durch die Pair Hidden Markov Model Alinierung gebildet wurden, werden mit Kognatgruppen verglichen, die durch Needleman-Wunsch Alinierung gebildet wurden. Unter Verwendung von achtfacher Kreuzvalidierung werden die Kognatgruppen verglichen. Pair Hidden Markov Models sind beim Problem der Kognatenklassifikation dem Needleman-Wunsch Alignment überlegen.
Im Vergleich zum Needleman-Wunsch Algorithmus bieten Pair Hidden Markov Models eine Vielzahl von Methoden für Sequenzvergleiche. Aus diesem Grund müssen Pair Hidden Markov Models in der historischen Linguistik berücksichtigt werden, wenn es um Sequenzvergleiche geht.

Contents

List of Figures

List of Tables

Chapter 1
Introduction

Automated techniques for word comparison and alignment are of increasing importance for linguists. The increasing amount of available data makes it necessary to have good automated techniques for word comparison and similar tasks. The relatively young field of bioiniformatics offers some methods which are also applicable for the needs of linguistics. In bioinformatics, protein and DNA sequences are compared. These sequences consist of discrete entities. Their type and their ordering can be compared with the parts of another sequence. This comparison allows conclusions of the relationship between these sequences. For the task of sequence comparison certain algorithms have been developed. To make these algorithms applicable for the needs of linguistics a few changes have to be made. The Needleman-Wunsch algorithm is such an algorithm which has been used in both fields. Another quite young approach to sequence comparison are Pair Hidden Markov Models. These models compare sequences on a stochastic basis. Similarities between sequences can thus be stated probabilistically.

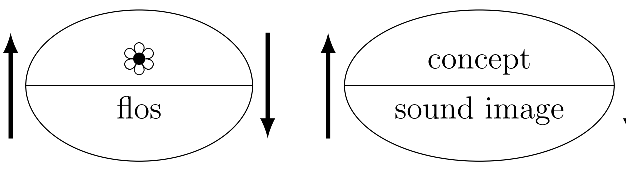
Chapter 2 introduces the concept of alignment and shows how this concept can be transferred to linguistics. This chapter also introduces the well known Needleman-Wunsch algorithm. The purpose of this algorithm is to compare and find the best alignment of two sequences. The simple version of this algorithm has certain shortcomings. To account for these shortcomings, substantial and structural extensions are also presented. In chapter 3, Pair Hidden Markov Models are presented. Pair Hidden Markov Models belong to the family of Markov Models. These models can be used to model stochastic processes. Pair Hidden Markov Models extend the notion of Markov Models to the task of sequence comparison. To get the idea of Pair Hidden Markov Models, Markov and Hidden Markov Models are introduced as well. Pair Hidden Markov models can be used to measure the likelihood that two sequences are related. The next chapter (chapter 4) compares the performance of the Needlemann-Wunsch algorithm and alignments carried out with Pair Hidden Markov Models on the basis of a cognate classification task. Cognate classification is important for historical linguistics. Cognates are used to determine the degree of the relationship between languages as well as for tasks of phylogenetic reconstruction. In the last chapter (chapter 5), the results of this thesis are discussed and summarized.

Chapter 2
Alignment

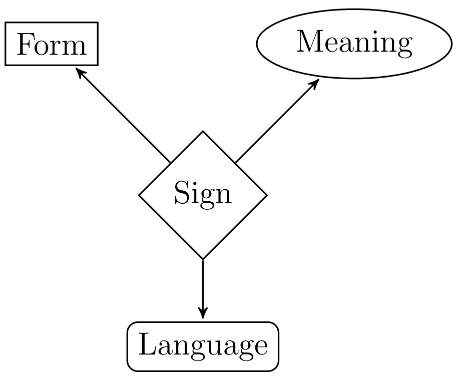
2.1 Signs and Sequences

In linguistics scholars are dealing with human language. But the subject of study is far from clearly defined. Following Saussure human language consists of two parts: *langue* (language) and *parole* (speaking) (c.f. Saussure, 1966, pp. 9-13). In this distinction, *langue* is a more or less abstract system of a language dealing with signs. *Parole* is the speakers utterance of sounds (c.f. Trask, 1999, p. 224). Since the utterance of sounds is a singular event, the focus of historical linguists is rather on *langue*. The *bilateral sign model* Saussure develops is based on the idea that a "linguistic sign unites [...] a concept and a sound image" (Saussure, 1966). In this model, the Latin sound image 'flos' (flower) is tied to the concept 'flower' (Q). This idea is shown in figure 2.1(a). In terms of Saussure, sound image and concept are called signifier (*signifiant*) and signified (*signifié*). The relation between the signifier and signified in this model is arbitrary. It just exists through social convention. Note that a motivated connection between signifier and signified is not excluded; e.g. onomatopoeia. Following List (2012b), the model of the linguistic sign gets extend by a language dimension which specifies the language a sign belongs to. In the example presented here, this dimension would bear the information 'Latin' (c.f. List, 2012b, p. 16). This extended model is shown in figure 2.1(b).

In historical linguistics, the form (signifier) aspect of the sign is very important. The *substance* and the *structure* part of the form are of great importance. The term substance refers to the physical properties and structure to the organization of the form. The substance aspect is usually addressed by a phonemic or phonetic transcription of the speech signal. By this method, the continuous speech signal is broken down into discrete units. These discrete units usually have a linguistic relevance. They are surely not reflected directly in the real utterance of speech



(a) Model of linguistic signs following Saussure



(b) Linguistic sign following List (2012b)

Figure 2.1: Two sign models

signals, which is simply due to the fact that real articulation shows effects like co-articulation. Yet, this breakdown is necessary for almost every linguistic application (c.f. List, 2012b, p. 18). By doing this reduction, one can either choose the *phonetic* or the *phonemic* perspective. While the phonetic perspective focusses on the material aspects of the sounds, i.e. its articulatory characteristics, the phonemic perspective concentrates on the functional level of sounds in the system of a language (c.f. Davenport & Hannahs, 1998, p. 3). The structural part of the form deals with the ordering of the segments which are based on the division described above. This ordering can best be described as a linear one, "i.e. as a *sequence of sounds*" (List, 2012b). The structural analysis of a sequence of sounds can be performed in two different ways: either *algebraic* or *substantial*. The algebraic analysis just compares two sounds and states whether they are identical or not. The substantial analysis compares two sound symbols based on their internal features (c.f. List, 2012b, p. 19).

As pointed out further above, linguistic signs have a sequential aspect. Sequences are also of high importance in computer science or biology. Therefore, many approaches to solve problems in sequence comparison were developed in these sciences. Some of these approaches can be transferred to problems in historical linguistics. At a first step of this transfer, it is necessary to clarify the term 'sequence' among others. Here, the terminology follows the one introduced by List (2012b). A definition for a sequence is given in Definition 1.

Definition 1 (Sequence)**:**

"Given an *alphabet* (a non-empty finite set, whose elements are called *characters*), a *sequence* is an ordered list of characters drawn from the alphabet. The elements of sequences are called *segments*. The *length* of a sequence is the number of its segments, and the *cardinality* of a sequence is the number its unique segments." (List, 2012b)

It is important to notice that the definition of sequences is adaptive for all kinds of sequences consequently, it is not restricted to sound sequences. As List points out, many of the structures we cope with in daily life can be seen as sequences; may it be a film with a sequence of pictures, or a book with a sequence of pages. They all have in common that their parts are ordered in a certain way. We can identify them by their segments and their ordering of these segments (c.f. List, 2012b, pp. 60). Given this definition, one can define the terms *subsequence*, *substring*, *prefix* and *suffix* (see Definition 2). For a linguist, the terms prefix and suffix are intuitively clear. It is important to note the difference between a subsequence and a substring.

Definition 2:

"Given two sequences s and t

(a) t is a *subsequence* $[t \lhd s]$ of s, if t can be derived from s by deleting some of the segments of s without changing the order of the remaining segments,

(b) t is a *substring* $[t \lessdot s]$ of s, if t is a subsequence of s and the derivation of t from s can be carried out by deleting only elements from the beginning and the end of s,

(c) t is a *prefix* of s, if t is a substring of s and the derivation of t from s can be carried out by deleting only elements from the end of s,

(d) t is a *suffix* of s, if t is a substring of s and the derivation of t from s can be carried out by deleting only elements from the beginning of s."

(List, 2012b)

2.2 Alignment

As pointed out above sequences are identified by their elements and how they are ordered. Therefore, two sequences can be compared by looking at their segments and if they have the same order. In the case of the film, both of them have first to be split in the pictures they consist of. If two films are the same, we get the same pictures in the same order. This method can also be applied to sound sequences. (1) shows a comparison of the sound sequence of the English word 'hand' (hEnd) and the corresponding German translation 'Hand' (hant).[1]

```
(1)   h E n d
      h a n t
```

From a naive point of view and just applying the 'film comparison' method, one has to judge these sequences as completely different. But there is no doubt that there is a correlation between these sequences, so it might be too strict to judge them as completely different. A naive method would be to count the numbers of unequal matches. In this example, this would yield 2 (E-a, d-t). Therefore, these strings have a distance of 2. This method is known as the *Hamming distance* which has been introduced by Hamming (1950).

The comparison of sequences can be done in two different ways, the *correspondence* or the *edit* perspective. The correspondence perspective distinguishes between *matches* and *empty matches*. If two segments correspond, it is called match and if a segment does not correspond to a segment of the other sequence, this is called an empty match. In the case of a match, there can either be a divergent or a uniform match. If two segments match and the are identical, this is a uniform match (e.g. h-h in (1)), if they match but are different, then it is divergent match (e.g. E-a in (1)). The edit perspective shows how much operations are needed to turn one sequence into the other. The two basic operations of the edit perspective are *substitution* and *indels* (*insertions* and *deletions*). This system of sequence comparison was first introduced by Levenshtein (1966). The substitution is equivalent to the divergent match in the correspondence perspective. The equivalent to the uniform match is called continuation. As pointed out, the two perspectives of sequence comparison are highly similar and therefore often

[1]The transcription system used here is the one used in the ASJP-Database (Wichmann et al., 2012). The system is explained in Brown, Holman, Wichmann, and Velupillai (2008).

used interchangeably. Nevertheless, the edit perspective is more accurate since it focusses on the way how a sequence is transformed into another. This also introduces a direction of the change and an order (c.f. List, 2012b, pp. 64-67). There are different ways to visualize the difference between sequences. A good overview is given in List (2012b). In the following, the *alignment* method is used to represent string comparison. The idea of an alignment is specified in Definition 3. Note that this definition limits an alignment to two sequences, so the result is a pairwise sequence alignment. Yet, it is possible to align more than two sequences, the result would then be a multiple sequence alignment. An extended definition can be found in List (2012b).

Definition 3 (Alignment)**:**
An alignment of two sequences S_1 and S_2 is obtained by writing the two sequences in two rows in such a manner that that matches and substitutions appear in the same column and empty matches are in the same column with a specific gap symbol (c.f. List, 2012b, p. 67; Gusfield, 1997, p. 216).

(2) is an example for an alignment. The four matches appear in the same column; one substitution h-k and three uniform matches o-o, r-r and n-n. The empty match u is aligned with the gap symbol -.

```
(2)   h o r n -
      k o r n u
```

(c.f. Jäger, 2013b, p. 2)

From the edit perspective the empty match is an insertion of u in the second sequence. There are different ways to measure the distance between two sequences. A simple scheme to score the difference between to sequences is the so called *Levenshtein distance*. This method assigns a certain score to the different edit operations to gain a measure for the distance (c.f. Levenshtein, 1966). A possible scoring scheme would be to assign a penalty of 0 to a match and a penalty of 1 to substitutions and empty matches. For (2), this scheme yields a distance of 2: three matches, one substitution and one indel operation. To find the best alignment between two sequences one could simply consider all possible alignments of two sequences and choose the one with the minimal Levenshtein distance. The problem for this approach is the high number of possible alignments. For two sequences of length n and m the number of possible alignments N is calculated by the formula 2.1.

$$N = \sum_{k=0}^{min\{n,m\}} 2^k \binom{m}{k} \cdot \binom{n}{k} \tag{2.1}$$

(c.f. Torres et al., 2003, p. 429)

This equation yields 681 possible alignments for “horn” and “kornu”. This might still be computationally feasible but for two sequences of length 10 there are 8097453 possible alignments. These are obviously to many alignments to compare them. Hence, it is not reasonable to obtain the best alignment just by comparison.

2.2.1 Needleman-Wunsch Algorithm

An efficient and popular algorithm to find the best alignment of two sequences is the so called *Needleman-Wunsch algorithm*, which was introduced by Needleman and Wunsch (Needleman & Wunsch, 1970). This algorithm belongs to the family of dynamic programming algorithms. The idea of dynamic programming is to reduce the solution to the solution of subproblems. For the edit distance between two strings this means that one has to compute the alignment for two strings of length m and n from the alignment of the substrings with length $m-1$ and $n-1$ (c.f. Gusfield, 1997, pp. 217). For the alignment of two sequences f and g of length m and n respectively, the Needelman-Wunsch algorithm computes the optimal alignment in four steps. It is important to notice that this algorithm does not compute the distance between two strings but their similarity, i.e. the higher the score, the more similar the two sequences are.

1. construct a matrix $A = (m + 1) \times (n + 1)$
2. initialize the matrix
3. recursively compute the values
4. trace back the optimal alignment

In this matrix, each segment of f is confronted with g, i.e. the value stored in A_{ij} is the score for the best alignment between the two substrings of f and g which end at

Match type	Score
Match	1
Substitution	-1
gap penalty	-1

Table 2.1: Example scoring scheme for Needleman-Wunsch algorithm

segment i and j respectively. The matrix is initialized with a zero value in the first cell of the first row, i.e. A_{00}. Then the first row and the first column are filled with multiples of the gap penalty. So the first row represents the alignment of g with a sequence of gaps and the first column an alignment of f with a sequence of gaps. As stated above as long as the values A_{i-1j-1}, A_{ij-1} and A_{i-1j} are known the value A_{ij} can be computed with the Needlemann-Wunsch algorithm (see equation 2.2). Based on equation 2.2, the matrix becomes filled recursively from top to bottom and from left to right (c.f. Durbin et al., 2001, pp. 19; List, 2012b, pp. 75).

$$A_{ij} = max \begin{cases} A_{i-1j-1} + \texttt{score}(f_i, g_i) \\ A_{i-1j} + \texttt{gap penalty} \\ A_{ij-1} + \texttt{gap penalty} \end{cases} \tag{2.2}$$

(c.f. Durbin et al., 2001, p. 20)

To know whether two segments match or whether there is an empty match for, one of the two segments one has to keep track of the origin of the value assigned to the specific cell. This method is called *traceback*. The traceback is done by reconstructing the alignment from the back. Starting from the final cell the algorithm moves back to the cell from which the value was derived, i.e. from A_{ij} to A_{i-1j-1}, A_{i-1j} or A_{ij-1}. Based on the movement, the two symbols are matched or aligned with a gap. If the movement occurs from A_{ij} to A_{i-1j-1}, f_i gets aligned to g_j; if the movement occurs to A_{i-1j}, f_i gets aligned to a gap and for A_{ij-1} and g_j respectively. Of course, it is important to have a scoring scheme which returns the values mentioned in equation 2.2. An example for a simple scoring scheme is given in table 2.1. There are several possibilities to extend the Needleman-Wunsch algorithm on the structural as well as on the substantial side.

Structural extensions

The basic Needleman-Wunsch algorithm performs a so called *global* alignment. A global alignment is characterized by its equal treatment of all segments of a sequence. This type of alignment does not distinguish whether a part of the sequence is a prefix, suffix or a substring. There are several ways to overcome this problem. The two most important extensions are *semi-global* and *local* alignment. In case of the two sequences “FATCAT” and “CATFISH”, a global alignment would result in (3). This alignment correctly align the two substrings “CAT” to each other but assigns a gap to all the other segments.

(3)
```
F A T C A T - - - -
- - - C A T F I S H
```
(c.f. List, 2012b, p. 80)

This alignment yields a similarity score of -4 and an edit distance of 7. However, it might be better in such a case to exclude the prefix “FAT” and the suffix “FISH” from the alignment; the semi-global alignment excludes them from the alignment (see (4)).

(4)
```
FAT | C A T | - - - -
- - - | C A T | FISH
```
(c.f. List, 2012b, p. 80)

This alignment procedure maintains a global view of the alignment but strips off ‘unnecessary’ prefixes and suffixes. The algorithm which computes the local alignment of two sequences is often called the *Smith-Waterman* algorithm; named after Smith and Waterman who published it in 1981 (Smith & Waterman, 1981). (5) shows the global (5-a) and the semi global (5-b) alignment of the two sequences “FATCAT” and “REDCAT”. Neither of these two alignments is really inadequate, nor is one of them optimal. One rather wants the alignment algorithm to ignore prefixes, which do not have a counterpart in the other sequence. The desired alignment is shown in (6).

(5) a. F A T C A T
R E D C A T

b. - - - - | F A T | CAT
REDC | - A T | - - -

(c.f. List, 2012b, p. 82)

(6) FAT | C A T
RED | C A T

(c.f. List, 2012b, p. 82)

Unlike semi-global alignment, the unmatched substrings, which are not contributing to the total alignment score, can occur in both strings when aligned locally. (c.f. Kondrak, 2002, p. 36). List (2012b) as well as Kondrak (2002) give a more detailed overview over structural extensions of the basic Needleman-Wunsch algorithm.

Substantial extensions

While structural extensions of the alignment algorithm are dealing with the question of how to integrate the structure of the sequence into the alignment, substantial extensions focus on the scoring functions of the algorithms. For a historical linguist a scoring scheme as shown in table 2.1 may look insufficient for a measurement of the similarity of two sequences. Substantial extensions try to tackle exactly these doubts. There are two points were the scoring scheme can be extended. More information can be added to the treatment of matches and substitutions or to the treatment of gaps.

A common way to extend the scoring scheme for matches and substitutions is to introduce a scoring matrix. In computational biology, an example for such a scoring matrix is the so called BLOSUM (BLock SUbstitution Matrix) (Henikoff & Henikoff, 1992). In the BLOSUM matrices log-odd ratio values for amino acid pairs $s(a, b)$ are stored.

$$s(a,b) = \frac{1}{\lambda} log \frac{p_{ab}}{q_a q_b} \tag{2.3}$$

(c.f. List, 2012b, p. 92)

Penalty	Conditions
0	Exact match of consonants or glides
5	Exact match of vowels (reflecting the fact that the aligner should prefer to match consonants rather than vowels if it must choose between the two)
10	Match of two vowels that differ only in length, or i and y, or u and w
30	Match of two dissimilar vowels
60	Match of two dissimilar consonants
100	Match of two segments with no discernible similarity
40	Skip preceded by another skip in the same word (reflecting the fact that affixes tend to be contiguous)
50	Skip not preceded by another skip in the same word

Table 2.2: Scoring scheme reported in Covington (1996)

The log-odds are calculated along the lines of equation 2.3. In this equation, λ serves as a scaling functor, p_{ab} is the frequency of a aligned with b, and q_a and q_b are the average frequencies of a and b appearing in any sequence (c.f. List, 2012b, p. 92).
A possible way to set up a linguistically sensible scoring scheme is to focus on the differences between sound classes. The scoring scheme presented in Covington (1996) tries to capture this intuition by distinguishing between consonants, vowels and glides. As Covington points out, the penalties were created by trial and error. Another possibility to create a scoring scheme is to represent sounds as vectors consisting of binary features. Gildea and Jurafsky (1996) as well as Nerbonne and Heeringa (1997) measure the distance between two sequences based on a feature system for sounds. Although such an approach seems to be more sensible for similarities and dissimilarities, Kondrak (2002) reports a high correlation between the two scoring schemes.

> The correlation between Covington's penalties and the average Hamming distances is very high (0,998), which demonstrates that feature-based phonology provides a theoretical basis for Covington's manually constructed distance function (Kondrak, 2002).

Among these two proposals, there is a scoring scheme which is based on multivalued features. In this method, features are no longer binary but can take different values on a certain scale. Though, it is still possible to get binary features, e.g. for nasality (Kondrak, 2000). Since all features are treated differently in this scheme, it is possible to weight certain features based on their importance for alignment analyses (List, 2012b).

So far, the penalty to open a gap is as high as the penalty to expand a gap. Because of this, there is no difference between opening three independent and discontinuous gaps and opening a gap with a length of three. Up to now, there are only linear gap penalties. The penalty for a gap of length l and a gap penalty of p is given in equation 2.4.

$$\begin{gathered}\texttt{Gap}(l) = -l \cdot p \\ \text{(c.f. Durbin et al., 2001, p. 16)}\end{gathered} \tag{2.4}$$

In contrast to such a linear measurement, it is possible to work with *affine gap penalties*. Affine gap penalties differentiate between the introduction and the extension of a gap. Usually, it is more expensive to open a gap than to extend it. Therefore the penalty for a gap of length l is given in equation 2.5. The gap opening penalty is d and the gap extension penalty is e.

$$\begin{gathered}\texttt{Gap}(l) = -p - (l-1) \cdot e \\ \text{(c.f. Durbin et al., 2001, p. 16)}\end{gathered} \tag{2.5}$$

2.2.2 Summary on Needleman-Wunsch Alignment

What was presented so far is the Needleman-Wunsch algorithm for pairwise alignment. The Needleman-Wunsch algorithm aligns two sequences based on a scoring scheme. There are several ways to improve the performance of the algorithm. The alignment can be improved by structural extensions of the algorithm. This method allows the algorithm to account for structural properties of the sequences, i.e. affixes. Another method to improve the performance of the algorithm is to make it more sensible for similarities and dissimilarities between the individual segments. These are so called substantial extensions to the algorithms.

Chapter 3
Pair Hidden Markov Models

For modelling stochastic processes often so called Markov Models are used. Pair Hidden Markov Models (PHMMs) are developed from Hidden Markov Models (HMMs) which are themselves developed from Markov Models. This chapter gives an introduction into these three types of models. First, the general ideas behind Markov Models are shown (section 3.1); second, there is a general introduction to HMMs (section 3.2), and third, PHMMs are introduced (section 3.3). The section about PHMMs also includes a short overview of the applications of PHMMs especially in linguistics.

3.1 Markov Models

A Markov Model[1] is a probabilistic model that models sequences of random variables. The probability of a random variable in a Markov chain does not depend on all the other variables before but only on the one immediately preceding this variable. This property is called the *Markov assumption*, named after the Russian mathematician Andrey Markov (1856-1922) (c.f. Jurafsky & Martin, 2009, p. 2009). Formally a Markov Model is defined as the triple $M = (S, \Pi, A)$ (see definition 4).

Definition 4 (Markov Model)**:**
$\lambda = (S, \Pi, A)$ with:

$S = \{s_1, \ldots, s_N\}$	set of states
$\Pi = \{\pi_i\},\ i \in S$	initial state probabilities
$A = \{a_{ij}\},\ i, j \in S$	state transition probabilities

(c.f. Manning & Schütze, 2003, p. 324)

[1]One might also find the terms Markov chain, Markov process or observed Markov Model for the kind of model described in this section.

The Markov assumption can then be formalized in the following way[2]:

$$\begin{aligned} & P(s_i|s_{i-1}) = P(s_i|s_1 \dots s_{i-1}) \\ & \qquad \text{with:} \\ & \qquad s_1 \dots s_i \in S \text{ and } i \leq N \\ & \qquad \text{(c.f. Jurafsky \& Martin, 2009, p. 209)} \end{aligned} \tag{3.1}$$

The state transition probabilities A of a Markov chain are written as a transition matrix. The properties in (3.2) have to hold for this transition matrix:

$$\begin{aligned} & a_{ij} = P(s_n = i|s_{n-1} = j) \\ & \quad \text{with:} \\ & \quad a_{ij} \geq 0 \text{ and } \forall i \sum_{j=1}^{N} a_{ij} = 1 \\ & \quad \text{(c.f. Manning \& Schütze, 2003, p. 318)} \end{aligned} \tag{3.2}$$

The initial state probabilities Π are defined as a vector characterizing the probability of s_i to be the first state of the Markov Model.

$$\begin{aligned} & \pi_i = P(x_1 = s_i) \\ & \quad \text{with:} \\ & \quad s_i \in S \text{ and } \sum_{i=1}^{N} \pi_i = 1 \\ & \quad \text{(c.f. Manning \& Schütze, 2003, p. 318)} \end{aligned} \tag{3.3}$$

3.2 Hidden Markov Models

In a Markov Model, there is no doubt about the state the chain is in, in a certain situation. This is due to the fact that this can be directly observed. In a Hidden Markov Model (HMM), one can not observe the actual state of the model. In case of a HMM, it is no longer possible to see the states but rather emissions which are emitted from the states of the HMM. Hidden Markov Models are then extensively defined by the five-tuple $\lambda = (S, K, \Pi, A, B)$ (see definition 5).

[2]Similar definitions can be found in Durbin et al. (2001) and Manning and Schütze (2003).

Definition 5 (Hidden Markov Model)**:**

$\lambda = (S, K, \Pi, A, B)$ with:

$S = \{s_1, \ldots, s_N\}$	set of states
$K = \{k_1, \ldots, k_M\}$	output alphabet
$\Pi = \{\pi_i\},\ i \in S$	initial state probabilities
$A = \{a_{ij}\},\ i, j \in S$	state transition probabilities
$B = \{s_i(k_t)\}\ s_i \in S; k_t \in K$	symbol emission probabilities

(c.f. Manning & Schütze, 2003, p. 324)

The definitions of S, Π and A are the same as above for the Markov Models. The symbols, which are emitted by the states, are listed in K. The probability that a certain symbol from the alphabet is emitted in a specific state, is stored in B in the way that $s_i(k_t)$ describes the probability of the symbol k_t being emitted in state s_i. There are two alternatives to define HMMs, *state emission HMMs* and *arc emission HMMs*. The type of HMMs described here is a state emission HMMs, i.e. the states are emitting the symbols. In an arc emission HMM, the symbols are emitted by changing the states, i.e. during the transition from state i to state $i+1$. Based on these parameters, a HMM creates an output sequence $O = (o_1, \ldots o_T)$ with $o_t \in K$. There is no lack of information if a HMM is only defined by the triple $\lambda' = (A, B, \Pi)$ since all states are coded in the transition matrix and the symbol emission probabilities inherit the output alphabet.

According to Rabiner (1989), there are three fundamental questions which an HMM should answer.

1. Given an observation sequence O and a model $\lambda = (A, B, \Pi)$, how probable is this sequence given λ, i.e. $P(O|\lambda)$?
2. Given an observation sequence O and a model $\lambda = (A, B, \Pi)$, what is the most likely state path through the HMM explaining O?
3. Which model parameters of $\lambda = (A, B, \Pi)$ will maximize $P = (O|\lambda)$?

(c.f. Rabiner, 1989, p. 261)[3]

To answer these questions there are specific dynamic programming algorithms for each of this question.

[3]See also Jurafsky and Martin (2009, p. 213) and Manning and Schütze (2003, p. 325).

> "For algorithms such as HMMs, the dynamic programming problem is generally described in terms of *trellises* (also called *lattices*). Here, we make a square array of states versus time, and compute the probabilities for being in each state at the preceding time in terms of the probabilities for being in each state at the preceding time instant." (Manning & Schütze, 2003)

Forward & Backward Algorithm

To solve the first problem there are two dynamic programming algorithms, the Forward Algorithm and the Backward Algorithm. For the Forward Algorithm stores the probability of ending up in state i after j observations in a so called *forward variable.*

$$\begin{aligned} \alpha_i(j) =& P(o_1, \ldots, o_{j-1}, s_t = i|\lambda) \\ & \text{(c.f. Manning \& Schütze, 2003, p.327)} \end{aligned} \tag{3.4}$$

This value $\alpha_i(j)$ will then at $(s_i,\ j)$ be stored in the trellis for the Forward Algorithm. This value expresses the total probability of landing in state i given the observation sequence $o_1, \ldots, o_{j-1}$. The values for the trellis are calculated by the algorithm described in Algorithm 1.

Algorithm 1 (Forward Algorithm)**:**

Initialization	$\alpha_i(1) = \pi_i b_i(o_1)$	$1 \leq i \leq N$
Recursion	$\alpha_i(j) = s_i(o_j) \sum_k \alpha_k(i-1) a_{kj}$	$1 \leq j \leq T; 1 \leq i \leq N$
Termination	$P(O|\lambda) = \sum_k \alpha_k(T)$	

(c.f. Manning & Schütze, 2003)[4]

The idea behind the Backward Algorithm is not as straightforward as for the Forward Algorithm. The Backward Algorithm calculates the backward probability $\beta_i(t)$ which indicates the probability of being in a certain state i at time t given

[4]Similar definitions can be found in Rabiner (1989); Jurafsky and Martin (2009)

the symbols $t+1$ to the end of the sequence.

$$\beta_i(t) = P(o_t, \ldots, o_T | s_t = i, \lambda) \tag{3.5}$$

(c.f. Manning & Schütze, 2003, p. 329)

The dynamic programming algorithm for the backward procedure is shown in Algorithm 2.

Algorithm 2 (Backward Algorithm)**:**

Initialization	$\beta_i(T+1) = 1$	$1 \le i \le N$
Recursion	$\beta_i(t) = \sum_{j=1}^{N} a_{ij} s_j(o_t) \beta_j(t+1)$	$1 \le t \le T; 1 \le i \le N$
Termination	$P(O\|\lambda) = \sum_{j=1}^{N} \pi_j \beta_j(1)$	

(c.f. Manning & Schütze, 2003)[5]

Viterbi algorithm

The second of the questions above asks for the most probable state path W through an HMM. To give an answer to this question, one needs to find the state $w_n \in W$ which will maximize $P(w_n|O,\lambda)$ for each n, $1 \le n \le N+1$. For a whole sequence of states this yields the following equation.

$$\arg\max_W P(W|O,\lambda) \tag{3.6}$$

(c.f. Manning & Schütze, 2003, p. 332)

It is sufficient to maximize this equation for a fixed sequence of observations.

$$\arg\max_W P(W,O|\lambda) \tag{3.7}$$

(c.f. Manning & Schütze, 2003, p. 332)

[5]Similar definitions can be found in Rabiner (1989); Jurafsky and Martin (2009).

An efficient algorithm to compute the most probable path is the *Viterbi algorithm*. It computes the probability $\upsilon_j(t)$ that the state sequence ends in state j.

$$\upsilon_j(t) = \max_{W_1,\dots,W_{t-1}} P(W_1 \dots W_{t-1}, o_1 \dots o_{t-1}, W_t = j|\lambda) \tag{3.8}$$

(c.f. Manning & Schütze, 2003, p. 332)

The algorithm computes this probability in a dynamic fashion like the Needleman-Wunsch algorithm (see section 2.2.1). As Algorithm 3 shows, the value $\upsilon_j(t)$ is calculated based on the value of $\upsilon_j(t-1)$.

Algorithm 3 (Viterbi algorithm)**:**

```
Initialization   $\upsilon_j(1) = \pi_j$                                              $1 \leq j \leq N$
Recursion        $\upsilon_j(t) = \max_{1\leq i\leq N} \upsilon_i(t-1)a_{ij}s_j(o_t)$            $1 \leq j \leq N$
                 Store pointer:
                 $\Upsilon_j(t) = \arg\max_{1\leq i\leq N} \upsilon_i(t-1)a_{ij}s_j(o_t)$        $1 \leq j \leq N$
Termination      $P(W,O|\lambda) = \max_{1\leq i\leq N} \upsilon_i a_{iN}$
                 $w_{N+1} = \arg\max_{1\leq i\leq N} \upsilon_i a_{iN}$
```

(c.f. Manning & Schütze, 2003)[6]

The value of $\upsilon_j(t)$ is calculated as the maximum of the product of transition probability, symbol emission probability and the probability of the preceding state. In order to reconstruct the optimal state path through the HMM, it is necessary to store a pointer to the state which maximizes $\upsilon_j(t)$. As for the Needleman-Wunsch algorithm, the optimal state path through the HMM is then reconstructed via a traceback.

Baum-Welch-Training

The third question looks for the most probable model parameters given an observation sequence. This is to find the model parameters which will maximize

[6] Similar definitions can be found in Rabiner (1989); Jurafsky and Martin (2009).

the probability $P(O|\lambda)$.

$$\arg\max_{\lambda} P(O|\lambda) \tag{3.9}$$

(c.f. Manning & Schütze, 2003, p. 333)

Normally, this maximization is done with a training sequence. Sadly, there is no analytic method to maximize this continuous function. So, an iterative hill-climbing algorithm has to be used (c.f. Manning & Schütze, 2003; Durbin et al., 2001). The algorithm used in this case for HMMs is the so called *Baum-Welch algorithm* or *Forward-Backward algorithm* (Baum, Petrie, Soules, & Weiss, 1970). This algorithm is a part of the family of the so called *Expectation-Maximization algorithms* (EM algorithms).[7] The EM algorithm iteratively calculates maximum likelihood estimates. For a HMM, this is to estimate the most probable model parameters, i.e. transition and emission probabilities. An EM algorithm generally performs two steps: an estimation and a maximization step. In case of the Baum-Welch algorithm, this is initially to compute probable paths based on a given distribution of transition and emission values and second to calculate new transition and emission values based on the paths. This process gets repeated until a stop criterion is reached. This procedure does not guarantee to find the global maximum, i.e. the overall best model parameters, but the log-likelihood of the model increases up to a local maximum. Which local maximum will be reached, highly depends on the initial model parameters (c.f. Durbin et al., 2001, p. 63). To calculate the new parameters of the model one wants to know how probable a certain transition or emission is given the training sample. So the transition probability, one wants to calculate, is the probability of a transition from s_i to s_j given observation t, $p_t(i,j)$.

$$\begin{aligned} p_t(i,j) &= P(W_t = i, W_{t+1} = j | O, \lambda) \\ &= \frac{P(W_t = i, W_{t+1} = j, O | \lambda)}{P(O|\lambda)} \end{aligned} \tag{3.10}$$

(c.f. Manning & Schütze, 2003, p. 333)

The denominator in this equation can be calculated from the Forward and the Backward algorithm. The probabilities $P(O|\lambda)$ given in Algorithm 1 and

[7]See McLachlan and Krishnan (1997) for an introduction into this type of algorithms.

Algorithm 2 are just special instances of equation 3.11.

$$\begin{aligned} P(O|\lambda) &= \sum_{i=1}^{N} \alpha_i(t)\beta_i(t) \\ &\text{with } 1 \leq t \leq T+1 \\ &\text{(c.f. Manning \& Schütze, 2003, p. 331)} \end{aligned} \tag{3.11}$$

As the alternative name of this algorithm suggests, these probabilities are calculated with the help of the Forward and the Backward algorithm. The probability $p_t(i,j)$ can be calculated from the forward variable for state s_i, i.e. $\alpha_i(t)$, and the backward variable for state s_j, i.e. $\beta_j(t+1)$. Together with equation 3.11, equation 3.12 can be derived from equation 3.10.

$$\begin{aligned} p_t(i,j) &= \frac{\alpha_i(t)a_{ij}s_i(k_t)\beta_j(t+1)}{\sum_{m=1}^{N}\alpha_m(t)\beta_m(t))} \\ &= \frac{\alpha_i(t)a_{ij}s_i(k_t)\beta_j(t+1)}{\sum_{m=1}^{N}\sum_{n=1}^{N}\alpha_m(t)a_{mn}s_n(t)\beta_n(t+1)} \\ &\text{(c.f. Manning \& Schütze, 2003, pp. 333)} \end{aligned} \tag{3.12}$$

For the updated transition probability $\hat{a}_{ij}$, this can for example simply be done by dividing the transition from state i to state j through all transitions leaving state i (see equation 3.13).

$$\begin{aligned} \hat{a}_{ij} &= \frac{\sum_{t=1}^{T} p_t(i,j)}{\sum_{t=1}^{T}\sum_{n=1}^{N} p_t(i,n)} \\ &\text{(c.f. Durbin et al., 2001, p. 62;} \\ &\text{Manning \& Schütze, 2003, pp. 334)} \end{aligned} \tag{3.13}$$

A new emission probability $\hat{s}_i(k_l)$ gets calculated similarly. The probability that k_t gets emitted in state s_i is the emissions of k_t in state s_i divided through all emissions of s_i (see equation 3.14).

$$\begin{aligned} \hat{s}_i(k_l) &= \frac{\sum_{\{t:o_t=l,1\leq t\leq T\}} p_t(i,j)}{\sum_{t=1}^{T} p_t(i,j)} \\ &\text{(c.f. Durbin et al., 2001, pp. 62;} \\ &\text{Manning \& Schütze, 2003, p. 335)} \end{aligned} \tag{3.14}$$

3.3 Pair Hidden Markov Models

Pair Hidden Markov Models (PHMMs) are based on Hidden Markov Models. But instead of emitting only a single sequence of symbols, a PHMM emits a pair of symbols. This type of Markov Models was first described in Durbin et al. (2001). They were invented to get a probabilistic model for pairwise alignment of DNA sequences. It derives from the property of PHMMs of emitting pairs of sequences that there is a probability distribution over pairs of symbols for the states. Additionally, there are states which emit pairs of symbols and gaps. A PHMM is formally defined in Definition 6.

Definition 6 (Pair Hidden Markov Model)**:**
$\lambda = (S, K, \Pi, A, B, B')$ with:

$S = \{s_1, \ldots, s_N\}$	set of states
$K = \{k_1, \ldots, k_M\}$	output alphabet
$\Pi = \{\pi_i\},\ i \in S$	initial state probabilities
$A = \{a_{ij}\},\ i, j \in S$	state transition probabilities
$B = \{s'_i(k_u k_v)\}\ s'_i \in S'; S' \subset S; k_u, k_v \in K$	emission probabilities for pairs of symbols
$B' = \{s''_j(k_w)\}\ s''_j \in S''; S'' \subset S; S' \cap S'' = \emptyset$	emission probabilities for symbols and gaps

Following this definition a PHMM is defined as the six-tuple $\lambda = (S, K, \Pi, A, B, B')$. Similar to HMM's, this six-tuple can be reduced to a quadruple $\lambda' = (A, B, B', \Pi)$. The argumentation why this reduction is legit for HMMs carries through for PHMMs since a PHMM emits pairs of sequences and not a single sequence. Therefore, there are two observation sequences $O = (o_1, \ldots, o_T)$ and $O' = (o'_1, \ldots, o'_R)$.

In principle, Definition 6 allows as many states as one wants to have. However, it has been shown that one only needs one state which emits pairs of symbols and two states emitting only single symbols, one state for each string. Additionally, there can be a 'begin' and an 'end' state. For example, in the PHMM in figure 3.1 state **M** emits aligned pairs of symbols, state **X** emits a symbol in the first string aligned with a gap in the second and vice versa for state **Y**. In the following, the

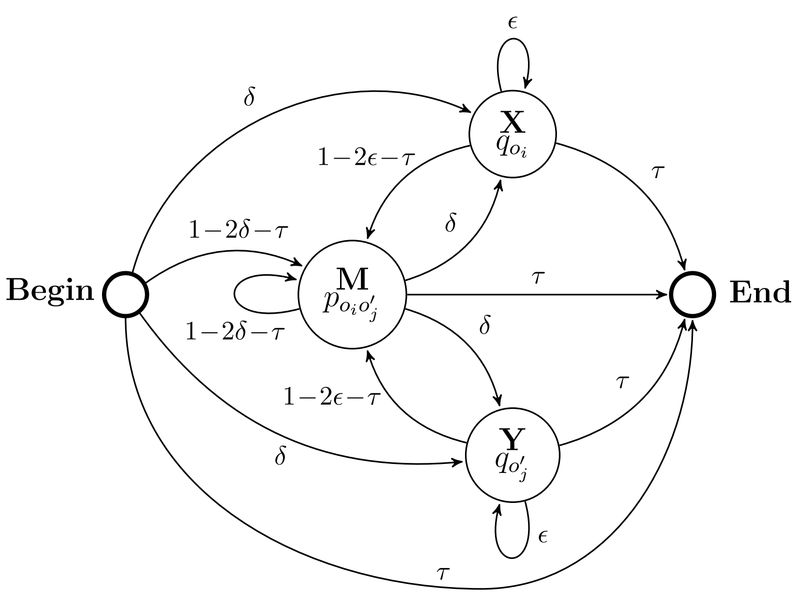


Figure 3.1: Pair HMM (c.f. Durbin et al., 2001, p. 82)

probability $s_M(o_i, o'_j) \in B$ is abbreviated to $p_{o_i o'_j}$, $s_X(o_i)$ to q_{o_i} and $s_Y(o'_j)$ to $q_{o'j}$.[8] The model described so far suits well for the needs of bioinformatics. For linguistic applications such as alignment this model falls a little bit to short. By taking a closer look at the model it becomes apparent that a transition from one gap state to the other is not possible. But in a linguistic application this transition is needed. (1) gives an example of an alignment between the Italian ('due') and the Spanish ('dos') word for 'two'. One would not hesitate to align the first two sounds of both words, but there is no evidence for a historical sound correlation between the Italian 'e' and the Spanish 's'. So it is most suitable to align both sounds to a gap (c.f. Mackay & Kondrak, 2005, p. 43).

(1) d u e -
    d o - s

(c.f. Mackay & Kondrak, 2005, p. 43)

This result can not be achieved from the model described above. To account for this, Mackay and Kondrak (2005) decided to allow a transition between the states **X** and **Y**. Figure 3.2 shows the model as it is described by Mackay and Kondrak (2005). With this model, it is possible to achieve an alignment like the one in (1).

[8]This notation is also the one which is in line with the literature about Pair Hidden Markov Models.

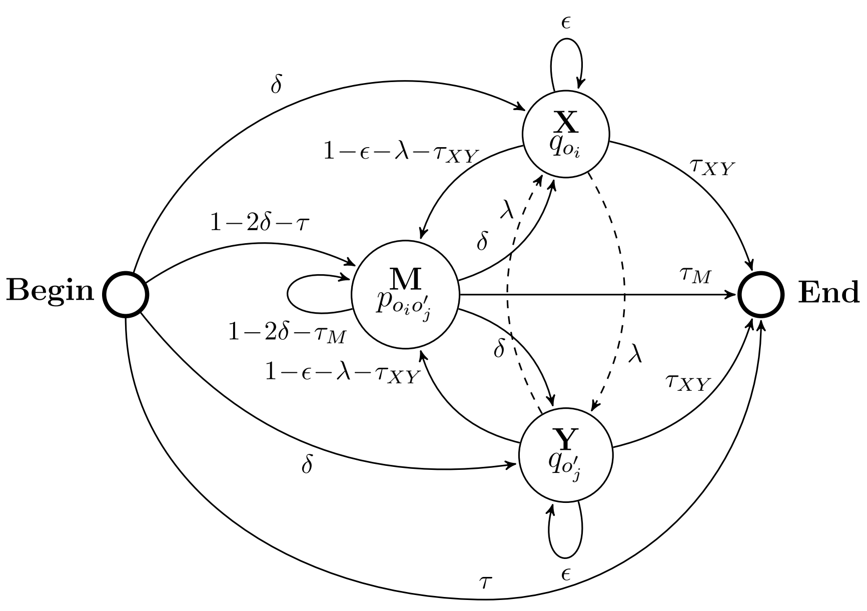


Figure 3.2: Pair HMM (c.f. Mackay & Kondrak, 2005, p. 44)

Another change that Mackay and Kondrak made is to split the transition parameter τ to the end state in to parameters. In the original model, it is as likely to go from the match state to the end state as to go from the two states **X** and **Y**. This is an unnecessary constraint for linguistic applications.

> For cognates, and other word similarity tasks, it may be that similar words are more or less likely to end in gaps or matches. The modification preserves the symmetry of the model while allowing it to capture how likely a given operation is to occur at the end of an alignment (Mackay & Kondrak, 2005).

These changes however keep the symmetry of the model. These 'linguistic' PHMMs have proven to be successful in cognate detection and word similarity tasks (see Mackay & Kondrak, 2005; Mackay, 2004) and dialectological applications (see Wieling, Leinonen, & Nerbonne, 2007). Since Pair Hidden Markov Models are an extension of Hidden Markov Models, the questions we asked above regarding a HMM are also relevant for PHMMs. To answer these questions algorithms similar to the ones for HMMs are used.

Forward & Backward algorithm

Similar to the Forward and Backward algorithm for HMMs, these algorithms compute the probability of a pair of observation sequences given a model λ. In case of PHMMs, this observation sequence is an alignment rather than just a series of symbols or pairs of symbols. So, the equation in (3.15) returns a probability indicating if O and O' are related "according to the [P]HMM by *any* alignment" (Durbin et al., 2001).[9]

$$P(O, O') = \sum_{\mu} P(\mu(O, O')) \qquad (3.15)$$

(c.f. Durbin et al., 2001, p. 87)

As for the Forward algorithm for HMMs the Forward algorithm for PHMMs computes the the forward variable α which is stored in a trellis. The forward variable $\alpha^k(i,j)$ then indicates the probability of ending up in state k after aligning the two sequences O and O' up to observation i and j respectively.

$$\begin{gathered} \alpha^k(i,j) = P(\mu(o_1, \ldots, o_i, o'_1, \ldots, o'_j), s_t = k|\lambda) \\ o_1, \ldots, o_i \lessdot O \text{ and } o'_1, \ldots, o'_j \lessdot O' \end{gathered} \qquad (3.16)$$

The trellis in case of PHMMs has one dimension more then the trellis for HMMs. This is due to the fact that there are two sequences of observations and not only a single one. Therefore, the superscript of the variable α indicates one of the three non-silent states shown in figure 3.2.[10] The dynamic programming algorithm is described in Algorithm 4.
As for the case of HMMs, there is also an algorithm for PHMMs which calculates the backward variable β. It is shown in Algorithm 5. This algorithm follows the same idea as for the HMMs, namely to return the backward variable $\beta^k(i,j)$, which indicates the probability of being in state k, given the sequences O and O' are aligned from observation $i+1$ and $j+1$ to the end of the sequences respectively.

$$\begin{gathered} \beta^k(i,j) = P(\mu(o_{i+1}, \ldots, o_n, o'_{j+1}, \ldots, o'_m)|s_t = k, \lambda) \\ o_{i+1}, \ldots, o_n \lessdot O \text{ and } o'_{j+1}, \ldots, o'_m \lessdot O' \end{gathered} \qquad (3.17)$$

[9] In the following, $\mu(O, O')$ will indicate an alignment between O and O'.
[10] The $\bullet$ indicates that the respective action is performed for all three states.

Algorithm 4 (Forward algorithm for PHMMs)**:**

Initialization

$$\alpha^M(0,0) = 1-2\delta-\tau_M,\ \alpha^X(0,0) = \alpha^Y(0,0) = \delta$$
$$\text{All } \alpha^\bullet(i,-1), \alpha^\bullet(-1,j) = 0$$

Recursion $0 \le i \le n, 0 \le j \le m$ except (0,0)

$$\begin{aligned}
\alpha^M(i,j) = \; & p_{o_i o'_j}[(1-2\delta-\tau_M)\alpha^M(i-1,j-1)+ \\
& (1-\epsilon-\lambda-\tau_{XY})(\alpha^X(i-1,j-1) \\
& +\alpha^Y(i-1,j-1))]; \\
\alpha^X(i,j) = \; & q_{o_i}[\delta\alpha^M(i-1,j) + \epsilon\alpha^X(i-1,j) + \lambda\alpha^Y(i-1,j)]; \\
\alpha^Y(i,j) = \; & q_{o'_i}[\delta\alpha^M(i,j-1) + \epsilon\alpha^X(i,j-1) + \lambda\alpha^Y(i,j-1)].
\end{aligned}$$

Termination

$$P(O,O'|\lambda) = \tau_M\alpha^M(n,m) + \tau_{XY}(\alpha^X(n,m) + \alpha^Y(n,m)).$$

(c.f. Mackay, 2004, p. 46)

Algorithm 5 (Backward algorithm for PHMMs)**:**

Initialization

$$\beta^M(n,m) = \tau_M,\ \beta^X(n,m) = \beta^Y(n,m) = \tau_{XY}$$
$$\text{All } \beta^\bullet(i,m+1), \beta^\bullet(n+1,j) = 0$$

Recursion $0 \le i \le n, 0 \le j \le m$ except (n,m)

$$\begin{aligned}
\beta^M(i,j) = \; & (1-2\delta-\tau_M)p_{o_{i+1}o'_{j+1}}\beta^M(i+1,j+1)+ \\
& \delta(q_{o_{i+1}}\beta^X(i+1,j) + \beta^Y(i,j+1)); \\
\beta^X(i,j) = \; & (1-\epsilon-\lambda-\tau_{XY})p_{o_{i+1}o'_{j+1}}\beta^M(i+1,j+1)+ \\
& \epsilon q_{o_{i+1}}\beta^X(i+1,j) + \lambda q_{o'_{j+1}}\beta^Y(i,j+1); \\
\beta^Y(i,j) = \; & (1-\epsilon-\lambda-\tau_{XY})p_{o_{i+1}o'_{j+1}}\beta^M(i+1,j+1)+ \\
& \epsilon q_{o'_{j+1}}\beta^Y(i,j+1) + \lambda q_{o_{i+1}}\beta^X(i+1,j);
\end{aligned}$$

Termination

$$P(O,O'|\lambda) = (1-2\delta-\tau_M)\beta^M(0,0) + \delta(\beta^X(0,0) + \beta^Y(0,0))).$$

(c.f. Mackay, 2004, p. 47)

Viterbi algorithm

The most probable state path W through a HMM was computed with the help of the Viterbi algorithm. In case of PHMMs, the most probable state path is the optimal alignment between two sequences. Finding this path means to maximize the probability of a state path given a model and two sequences.

$$\arg\max_W P(W|O,O',\lambda) \tag{3.18}$$

As for HMMs, it is sufficient to maximize this probability for a fixed pair of sequences.

$$\arg\max_W P(W,O,O'|\lambda) \tag{3.19}$$

As in the case of HMM, the Viterbi algorithm computes this probability dynamically. The value in each cell of the trellis represents the value of the single most optimal alignment leading to this respective cell.

Algorithm 6 (Viterbi algorithm for PHMMs)**:**

Initialization

$$\upsilon^M(0,0) = 1 - 2\delta - \tau_M,\ \upsilon^X(0,0) = \upsilon^Y(0,0) = \delta$$
$$\text{All } \upsilon^\bullet(i,-1), \upsilon^\bullet(-1,j) = 0$$

Recursion $0 \le i \le n, 0 \le j \le m$ except $(0,0)$

$$\upsilon^M(i,j) = p_{o_i o'_j} max \begin{cases} (1-2\delta-\tau_M)\upsilon^M(i-1,j-1), \\ (1-\epsilon-\lambda-\tau_{XY})\upsilon^X(i-1,j-1), \\ (1-\epsilon-\lambda-\tau_{XY})\upsilon^Y(i-1,j-1) \end{cases}$$

$$\upsilon^X(i,j) = q_{o_i} max \begin{cases} \delta\upsilon^M(i-1,j), \\ \epsilon\upsilon^X(i-1,j), \\ \lambda\upsilon^Y(i-1,j) \end{cases}$$

$$\upsilon^Y(i,j) = q_{o'_j} max \begin{cases} \delta\upsilon^M(i,j-1), \\ \epsilon\upsilon^X(i,j-1), \\ \lambda\upsilon^Y(i,j-1) \end{cases}$$

Termination

$$P(W,O,O'|\lambda) = max(\tau_M\upsilon^M(n,m), \tau_{XY}\upsilon^X(n,m), \tau_{XY}\upsilon^Y(n,m)).$$

(c.f. Mackay, 2004, p. 45)

Baum-Welch-Training

Parameter estimation for PHMMs is widely done with Baum-Welch-Training as in the case of HMMs. The probability to go from state h to state k depends on the two observations o_i and o'_j in the two observation sequences O and O' respectively.

$$\xi_{o_i,o'_j}(h,k) = \frac{\alpha_h(o_i,o'_j)a_{rs}e_k(\diamond 1)\beta_k(\diamond 2,\diamond 3)}{\sum_{l=1}^{N}\alpha_l(o_i,o'_j)\beta_l(o_i,o'_j)} \tag{3.20}$$

(c.f. Wieling, 2007, p. 48)

The value of $\diamond 1$, $\diamond 2$ and $\diamond 3$ depends on state k. If k is the substitution state $e_k(\diamond 1)$ resolves to $p_{o_{i+1}o'_{j+1}}$ and to $q_{o_{i+1}}$ or $q_{o'_{j+1}}$ for the states **X** and **Y** respectively. If k is the substitution state $(\diamond 2,\diamond 3)$ resolves to $(i+1,j+1)$ and to $(i+1,j)$ or $(i,j+1)$ for the states **X** and **Y** respectively. (c.f. Wieling, 2007, p. 48). The new transition values for the PHMM are computed along the lines of equation 3.21. The new transition probabilities are calculated on all sequence pairs (w) in the training set.

$$\hat{a}_{hk} = \frac{\sum_w \sum_{i=1}^{\tilde{T}} \sum_{j=1}^{\tilde{R}} \xi^w_{o_i,o'_j}(h,k)}{\sum_w \sum_{i=1}^{\tilde{T}} \sum_{j=1}^{\tilde{R}} \sum_{k=1}^{N} \xi^w_{o_i,o'_j}(h,k)} \tag{3.21}$$

(c.f. Wieling, 2007, p. 49)

The value of $\tilde{T}$ and $\tilde{R}$ depend on the final emitting state:

- if **M** is the final emitting state $\tilde{T} = T - 1$ and $\tilde{R} = R - 1$,
- if **X** is the final emitting state $\tilde{T} = T - 1$ and $\tilde{R} = R$,
- if **Y** is the final emitting state $\tilde{T} = T$ and $\tilde{R} = R - 1$.

Equation 3.14 was used to calculate the new emission probabilities for a HMM. This equation can be transformed to equation 3.22 to calculate the probability of going through state h.

$$\hat{s}_h(o_i, o'_j) = \frac{\alpha_h(o_i, o'_j)\beta_h(o_i, o'_j)}{\sum\limits_{n=1}^{N} \alpha_n(o_i, o'_j)\beta_n(o_i, o'_j)} \tag{3.22}$$

(c.f. Wieling, 2007, p. 49)

By using this equation it is possible to calculate the new emission probabilities with the help of equation 3.23. In this equation, $\diamond 1$ and $\diamond 2$ vary depending on m. If h is state **M** or state **Y** $\diamond 1$ yields $o_i = o_k$ and if h is state **M** or **X** $\diamond 2$ yields $o'_j = o'_k$. In state Y is $\diamond 2$ undefined. This behaviour is due to the fact that in state Y o' is not present and in state X o (Wieling, 2007, p. 49).

$$\hat{s}_m(o_k) = \frac{\sum\limits_{w} \sum\limits_{\substack{o_i=0 \\ \diamond 1}}^{T} \sum\limits_{\substack{o'_j=0 \\ \diamond 2}}^{R} s_m^w(o_i, o'_j)}{\sum\limits_{w} \sum\limits_{o_i=0}^{T} \sum\limits_{o'_j=0}^{R} s_m^w(o_i, o'_j)} \tag{3.23}$$

(c.f. Wieling, 2007, p. 49)

Random Model

The random model R (see figure 3.3) independently produces two sequences o and o' of length n and m respectively. With probability $1 - \eta$ the state **X** loops back onto itself and emits the sequence o. The same mechanism applies for state **Y** and sequence o'. The silent state in between the states **X** and **Y** mediates between the two. There is also a begin state **B** and an end state **E**. Within this model it is possible to have sequences of length 0. Along the lines of equation 3.24 the probability of two sequences according to R is calculated.

$$\begin{aligned} P(o, o'|R) &= \eta(1-\eta)^n \prod_{i=1}^{n} q_{o_i} \eta(1-\eta)^m \prod_{j=1}^{m} q_{o'_j} \\ &= \eta^2(1-\eta)^{n+m} \prod_{i=1}^{n} q_{o_i} \prod_{j=1}^{m} q_{o'_j} \end{aligned} \tag{3.24}$$

(c.f. Durbin et al., 2001, p. 84)

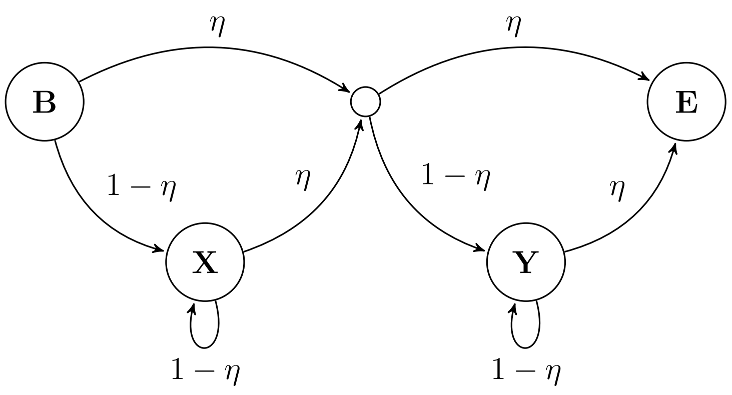


Figure 3.3: Random Model (c.f. Durbin et al., 2001, p. 83)

The exact value of η depends on the expected length L of the sequences o and o'.

$$\eta = \frac{1}{L+1} \tag{3.25}$$

(c.f. Borodovsky & Ekisheva, 2006, p. 106)

3.3.1 More on Pair Hidden Markov Models

Pair Hidden Markov Models are a probabilistic approach to the alignment problem. This probabilistic approach allows for a measurement of the reliability of an alignment and to look for other, maybe also suboptimal, alignments. "Indeed by weighting all alternatives probabilistically, we will be able to score the similarity of two sequences independent of any specific alignment." (Durbin et al., 2001, p. 80) This property of the PHMM makes it a very useful tool for alignment analysis. Originally developed for computational biology PHMMs were successfully applied for alignment of genes (see e.g. Knudsen and Miyamoto (2003); Meyer and Durbin (2002); Durbin et al. (2001)). For sequence comparison in linguistics, PHMMs are not that widely used yet.

Mackay and Kondrak (2005) use a PHMM for automated cognate detection tasks. The model used in the work of Mackay and Kondrak was trained on a set of known cognates. The authors show that the use of PHMMs for the cognate detection task outperforms all previous used automated techniques. They also show that a Viterbi algorithm which incorporates log-odds is the most successful one for the

task of cognate detection.
Based on the model proposed in Mackay and Kondrak (2005), Wieling et al. (2007) induce differences between sound segments. Trained on dialect data, the PHMM in this task was used to measure the distance between two dialects. The distance was calculated by averaging this distance between the word pairs of the two dialects. Based on the distances, the authors build distance matrices which show a high correlation to the respective matrices created by slightly modified Levensthein alignment. The dialectal maps generated with the use of these matrices are highly similar. Although the PHMM approach is more sophisticated, the Levensthein approach yields similar results. As to the authors, this is due to the richness of the used database, which compensates for the missing sensitivity in Levensthein alignment.
Clark (2001) makes use of PHMMs for stochastic string transduction. This model was designed to learn morphological processes. The model in this application was built to detect certain morphological classes, e.g. for English past tense inflection or German nominal plural inflection

Chapter 4
Alignment and word comparison

4.1 Alignment and cognates

Originally developed for bioinformatics Needleman-Wunsch alignment has proven to be useful for tasks of historical linguistics. For historical linguistics, there are obviously no DNA or protein sequences which can be compared. The DNA or protein sequences of linguistics actually are words or sound sequences. Needleman-Wunsch alignment was used in Jäger (2013b) to infer a phylogenetic tree.[1] This inference can be done in two ways. The *character based* method and the *distance based* method. The character based method represents each language or organism as a vector. Values for a given set of discrete entities are stored in this vector. In the distance based method, a distance is calculated for each pair of taxonomic units (c.f. Jäger, 2013a, p. 1; Baldauf, 2003, p. 349). Phylogenetic algorithms construct trees from this data. In case of the character based method, the algorithms compute a tree which is based on the vectors. By doing this, the algorithms optimize certain criteria, e.g. maximal posterior likelihood of all mutations or minimal number of mutations (c.f. Jäger, 2013a, p.1). Algorithms for the distance based method construct a tree by optimizing the match between the leaf distance in the tree and the respective distance stored in the matrix. Both of these methods have their advantages and disadvantages. The character based method provides more information than the distance based method which merely yields a tree. Otherwise, the distance based method is more suitable when dealing with large data sets.
Needleman-Wunsch alignment can also be used for cognate detection tasks. Cognates are very important for the comparative method. A working definition of cognates is given in definition 7.

[1]"Phylogenetics is the science of estimating the evolutionary past, in the case of molecular phylogeny, based on the comparison of DNA or protein sequences." (Baldauf, 2003, p. 345)

Definition 7 (Cognate)**:**
Two words are *cognates*, if they both descent from a common ancestor.(c.f. List, 2012b, p. 41)

The comparative method tries to reconstruct protolanguages from known languages which are related. Cognate detection is also important for phylogenetic inference. In the character based method cognate classes are used as suitable values (e.g. Gray & Atkinson, 2003). This points out the importance of cognate detection for historical linguistics. As there are large data sets available it is necessary to have an automatized method for cognate detection. The `LexStat` method, which was introduced by List (2012a), successfully detects cognates. A sophisticated conversion procedure transfers all words in the input into sound classes. Sonority profiles of these words are also identified. In the next step language specific scoring schemes are estimated and then used for the calculation of the pairwise distance of all word pairs for the respective two languages. Based on these distances, the words are then clustered into cognate sets. (List, 2012a)
In the study presented here, another but similar approach is used to identify cognate clusters. Possible cognates are aligned with the log-odds version of the Viterbi algorithm for PHMMs. The score of this algorithm is used to cluster the words into cognate sets. As Mackay and Kondrak report in their paper from (2005) this algorithm is the most suitable for that task (Mackay & Kondrak, 2005).

4.2 Results

4.2.1 Parameter Estimation

The Pair Hidden Markov Model in this study is trained on set of cognates for the Germanic languages extracted from the Automated Similarity Judgement Program database (ASJP database) (Wichmann et al., 2012). There are 41 Germanic languages in the ASJP database. The ASJP database is a database consisting of a 40 item Swadesh list for 6139 languages. Each of the Germanic languages was paired with every other of the Germanic languages except with itself yielding 820

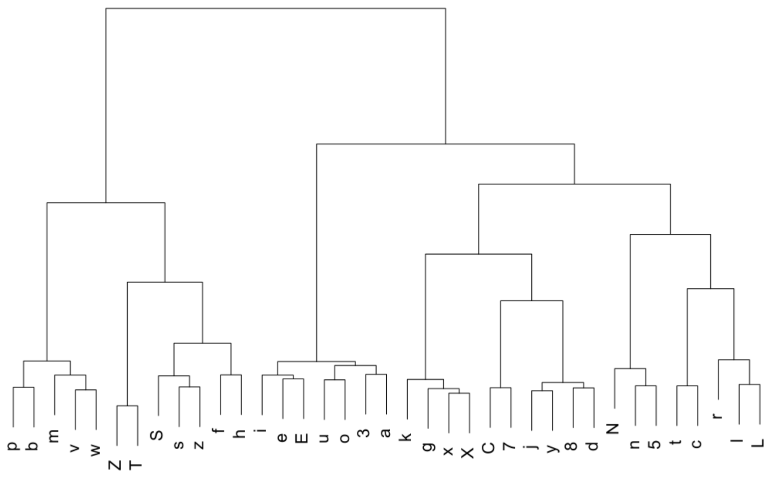


Figure 4.1: Hierarchical clustering of the Germanic sounds derived by PHMM parameter estimation

language pairs.[2] Each word pair with the same meaning was treated as probable cognates and thus used for the training algorithm. If a language misses an entry for a certain concept, this word pair is excluded from training. In case a language has multiple entries for a word each combination of words with this entry is used for the training. Applying this method results in a training set of 38071 word pairs. To avoid order and position specific effects, the mirror pair of each word pair is also used for training, i.e. the pair $[word_1, word_2]$ as well as the pair $[word_2, word_1]$. There are 1369 substitution probabilities, 74 insertion and deletion probabilities and 5 transition probabilities that have to be estimated. A Pair Hidden Markov Model as described in section 3.3 was trained using the Baum-Welch algorithm.[3] The substitution probabilities for the Germanic languages are displayed in a hierarchical clustering shown in figure 4.1. The cluster shows that the vowels form a cluster of their own and that also the consonants which are similar to one another are grouped together. Table 4.1 displays some substitution probabilities generated by the Baum-Welch algorithm. Although all these values are all fairly small, the values in the diagonals of all these tables are significantly higher than the uniform

[2]The number of unique pairs out of a given set can be calculated by the binomial coefficient $\binom{n}{k}$. This is $\binom{41}{2}$ for this case.

[3]The original implementation used in Mackay and Kondrak (2005) and which was extended by Wieling et al. (2007) was kindly made available by Martijn Wieling. To get a better interaction with the methods used in LingPy and the software used for the EVOLAEMP project, the algorithms were reimplemented in Python 2.7.

	a	**e**	**i**	**o**	**u**
a	0.0234	0.0064	0.0039	0.0097	0.0020
e	0.0064	0.0135	0.0097	0.0047	0.0014
i	0.0039	0.0097	0.0267	0.0036	0.0065
o	0.0097	0.0047	0.0036	0.0206	0.0112
u	0.0020	0.0014	0.0065	0.0112	0.0204

(a) Vowel substitution probabilities

	p	**b**	**t**	**d**	**k**	**g**
p	0.0015	0.0024	0.0004	0.0002	0.0001	0.0000
b	0.0024	0.0325	0.0023	0.0003	0.0014	0.0001
t	0.0004	0.0023	0.0602	0.0076	0.0005	0.0009
d	0.0002	0.0003	0.0076	0.0089	0.0002	0.0006
k	0.0001	0.0014	0.0005	0.0002	0.0143	0.0021
g	0.0000	0.0001	0.0009	0.0006	0.0021	0.0032

(b) Consonant substitution probabilities

Table 4.1: Substitution probabilities

substitution probability. [4] This reflects the intuition that the probability of identity substitution should be higher than that one of a chance substitution.

Since the amount of data for training and testing is relatively small, the PHMM used by Wieling et al. (2007) was trained on almost 100 million wordpairs; eight-fold cross validation was used to get more reliable results. In general, it should be the case that the test data is completely different from the training data. For the cognate detection task described here, this is not entirely possible. The languages, which are used for the training, are all related to one another since they all belong to the Germanic languages. Therefore it is possible that there are words in the training set which are very similar or even nearly identical to the items in the test set. This problem is also reported by Mackay (2004). It cannot be mastered by deleting all words from the training data which are similar to the words in the test set. In fact, there would be nearly no training data then. As Mackay (2004) reports, this problem has to be overcome by the mass of data in the training set compared to the amount of data in the test set. By eight-fold cross validation, these are 630 language pairs within the training set compared to 10 in the test set.

[4]The uniform substitution probability $\frac{1}{1369} \approx 0.00073$

4.2.2 Alignment Results

To measure how well the PHMM aligns the words the result of the alignment has to be compared to a gold standard. But in fact, there is no information stored in the ASJP database about alignments. So, there is no straightforward possibility to compare the alignment carried out by the Needleman-Wunsch algorithm and the log-odds version of the Viterbi algorithm for PHMMs. To account for this problem cognate clusters within the ASJP database were estimated with the `LexStat` method. `LexStat` clusters the cognates with a version of the UPGMA cluster algorithm; originally proposed by Sokal and Michener (1958). The version used here is the one used by List (2012a, 2012b). This version differs from the original version by having an additional parameter specifying a threshold for pairwise distances. "The threshold itself reflects the maximally allowed average distance between all word pairs in a given cluster." (List, 2012b, p. 174) List (2012a) reports that a threshold of 0.6 is the best for estimating cognate clusters. Using this parameter the ASJP data were clustered into cognate sets. These clusters were then used as the basis for the comparison of Needleman-Wunsch and Viterbi alignment.

The alignment scores of the Needleman-Wunsch algorithm were created along the lines described by Jäger (2013b). In a first step words of possibly related languages are aligned using the Levensthein algorithm. Then substitution scores are derived by the use of equation 4.1, where $p(a, b)$ is the probability that the sounds a and b are aligned and $q(a)$ and $q(b)$ are the probabilities that a and b occur in the first and second word respectively. This substitution matrix was then used to align the words with the Needleman-Wunsch algorithm. All word pairs then having an alignment score > 0 were used to compute $p_1(a, b)$. This procedure was repeated three times. As Jäger (2013b) reports, further iterations do not lead to significant changes.

$$s(a, b) = log \frac{p(a, b)}{q(a)q(b)} \tag{4.1}$$

(Jäger, 2013b)

Based on these substitution scores all possible cognates of the Germanic languages are aligned and their pairwise alignment score was calculated. An example for such alignment scores is given in table 4.2. Words, which are similar to one another, have a high score and vice versa. The Alsatian word *nom3* and the Bernese German

	Alsatian	**EF**	**NLW**	**Afrikaans**	**BG**	**FV**
	nom3	*nom3*	*nam*	*nam*	*nome*	*nEm3*
Alsatian	5.2174	5.2174	-3.5607	-3.5607	2.0487	1.3894
EF	5.2174	5.2174	-3.5607	-3.5607	2.0487	1.3894
NLW	-3.5607	-3.5607	1.518	1.518	0.8375	-2.6751
Afrikaans	-3.5607	-3.5607	1.518	1.518	0.8375	-2.6751
BG	2.0487	2.0487	0.8375	0.8375	6.2741	-0.8497
FV	1.3894	1.3894	-2.6751	-2.6751	-0.8497	4.6736

Table 4.2: Pairwise alignments scores created by Needleman-Wunsch alignment (EF: Eastern Frisian, NLW: Northern Low Saxon, BG: Bernse German, FV: Frans Vlaams)

	Alsatian	**EF**	**NLW**	**Afrikaans**	**BG**	**FV**
	nom3	*nom3*	*nam*	*nam*	*nome*	*nEm3*
Alsatian	11.5190	11.5190	7.8706	7.8706	10.3862	10.3628
EF	11.5190	11.5190	7.8706	7.8706	10.3862	10.3628
NLW	7.8706	7.8706	9.6216	9.6216	6.8376	8.0553
Afrikaans	7.8706	7.8706	9.6216	9.6216	6.8376	8.0553
BG	10.3862	10.3862	6.8376	6.8376	11.5602	9.2299
FV	10.3628	10.3628	8.0553	8.0553	9.2299	11.4201

Table 4.3: Pairwise Alignment scores created by Viterbi log-odds alignment (EF: Eastern Frisian, NLW: Northern Low Saxon, BG: Bernse German, FV: Frans Vlaams)

word *nome*, both denoting the concept "name", only differ in one sound pair (*3* - *e*). This leads to a higher similarity score than for the Alsatian and Frans Vlaams (*nEm3*) word pair. Already in this similarity matrix one can observe that there are certain words which tend to form clusters. The UPGMA cluster algorithm indeed puts Northern Low Saxon and Afrikaans together in one cluster.[5] Another cluster is formed by Bernse German, Alsatian and Eastern Frisian. Frans Vlaams forms its own cluster. A trained linguist however would put all the words together in one cluster.

The Viterbi log-odds algorithm contrariwise, assigns the values displayed in table 4.3 to the same word pairs. Although the general pattern in both matrices look the same, the clustering algorithm puts all the words in one cluster if confronted with the scores from the Viterbi algorithm. If translated into a distance matrix, the differences between the values in table 4.2 result in higher distances than the differences in table 4.3. This results in a more adequate clustering for the

[5]For the UPGMA clustering algorithm these similarity scores have to be transformed into distances.

Viterbi data than for the Needleman-Wunsch data.
In order to eliminate effects of the clustering algorithm, the parameters of the clustering algorithm are held constant throughout the whole evaluation procedure. The classification into cognate sets was done for each of the test sets. These clusters were then compared with the clusters generated by the `LexStat` method. For a threshold value of 0.8 the highest precision and recall values were achieved. An overview of these values is given in table 4.4. The comparison of the cognate clusters was done by the use of *pair precision* (equation 4.2) and *pair recall* (equation 4.3). Both methods use the amount of cognate pairs in the test set (P_t) and the number of cognate pairs in the reference set (P_r) for the calculation of the respective values. These methods compute the precision and the recall of all cognate pairs. The pair precision (PP) reflects the amount of false positive decisions made by the clustering. One can conclude: the higher the pair precision, the lower the number of false positives.

$$PP = \frac{|P_t \cap P_r|}{|P_t|} \tag{4.2}$$

(List, 2012b)

The pair recall (PR) score reflects how many false negatives were produced by the algorithm. The smaller the amount of false negatives, the higher is the pair recall score.

$$PR = \frac{|P_t \cap P_r|}{|P_r|} \tag{4.3}$$

(List, 2012b)

The *pair F-score* (PFS) is computed as a combination of pair recall and pair precision. This measure uses the geometric mean to combine precision and recall (Carstensen et al., 2004).

$$PFS = 2 \cdot \frac{PP \cdot PR}{PP + PR} \tag{4.4}$$

(List, 2012b)

Using this measurements the evaluation of the different test sets for the Viterbi log-odds alignment returns the values shown in table 4.4. The respective values for Needleman-Wunsch alignment are shown in table 4.5. The tables show that the

	Precision	**Recall**	**F-score**
Testset 1	0.7417	0.9008	0.7699
Testset 2	0.7675	0.96	0.8122
Testset 3	0.6638	0.8525	0.6933
Testset 4	0.6942	0.865	0.7244
Testset 5	0.69	0.7738	0.6745
Testset 6	0.7246	0.8983	0.7527
Testset 7	0.6388	0.8233	0.6506
Testset 8	0.7375	0.9525	0.7914
Mean	0.7072	0.878	0.7336

Table 4.4: Results of cognate classification using the PHMM Viterbi log-odds algorithm

	Precision	**Recall**	**F-score**
Testset 1	0.3529	0.6588	0.4174
Testset 2	0.3933	0.7229	0.4394
Testset 3	0.2838	0.61	0.3145
Testset 4	0.5213	0.7067	0.5555
Testset 5	0.3517	0.75	0.4256
Testset 6	0.3725	0.6525	0.4189
Testset 7	0.2563	0.4733	0.2877
Testset 8	0.4493	0.8248	0.519
Mean	0.6671	0.3642	0.4134

Table 4.5: Results of cognate classification using the Needleman-Wunsch alignment

values for precision and recall as well are higher for the Viterbi log-odds alignment than for the Needleman-Wunsch alignment. This is also visible in the respective F-scores. These values indicate that the Viterbi log-odds algorithm outperforms Needleman-Wunsch alignment in cognate recognition (all $p's < 0.007$).

The Forward algorithm can be used to calculate the similarity between two strings, too. Instead of calculating the score for the best alignment, this algorithm calculates the sum of all possible alignments. Alignments are highly influenced by the length of the strings. Therefore, it is necessary to correct for this issue. The log-odds version of the Viterbi algorithm has a built in correction for length through the connection with the random model. The simple Forward algorithm does not have this property. To account for this problem, there are two options to modify the Forward algorithm: either to correct the final score for length, or to use log-odds scores as substitution probabilities. The correction for length is done by

	Precision	**Recall**	**F-score**
Testset 1	0.4083	0.1429	0.1840
Testset 2	0.6833	0.3071	0.3694
Testset 3	0.5708	0.2358	0.2860
Testset 4	0.6542	0.3538	0.4210
Testset 5	0.5479	0.2083	0.2635
Testset 6	0.5625	0.2546	0.2861
Testset 7	0.3750	0.1667	0.1961
Testset 8	0.7925	0.4774	0.5409
Mean	0.5743	0.2683	0.3184

Table 4.6: Results of cognate classification using the Forward algorithm

	Precision	**Recall**	**F-score**
Testset 1	0.5713	0.3179	0.3502
Testset 2	0.7521	0.4408	0.4759
Testset 3	0.5333	0.2613	0.2696
Testset 4	0.6604	0.4204	0.4608
Testset 5	0.6958	0.3546	0.4069
Testset 6	0.5000	0.1788	0.2383
Testset 7	0.4083	0.1813	0.2228
Testset 8	0.8521	0.5234	0.5960
Mean	0.6217	0.3348	0.3775

Table 4.7: Results of cognate classification using the log-odds version of the Forward algorithm

multiplying the final alignment score with the normalizing factor $\frac{1}{C^n}$ where C is a constant and n the length of the longest string (Mackay & Kondrak, 2005). A value of 0.008 for C serves best for the data used here. The results of the clustering carried out with this method are shown in table 4.6. The means of precision, recall and F-Score are all below the respective values generated by Needleman-Wunsch and Viterbi log-odds alignment. Table 4.7 shows the values of the cognate classification task based on log-odds version of the Forward algorithm. The mean values shown here are closer the values in table 4.5 than the previous ones. Nevertheless, these values are below the ones for the log-odds version of the Viterbi algorithm (c.f. table 4.4). These results are in line with those presented by Mackay and Kondrak (2005).

Chapter 5
Discussion

Automated alignment techniques have proven themselves to be useful in bioniformatics and in linguistics as well. This thesis compares how well a stochastic model for sequence comparison, a Pair Hidden Markov Model, performs in contrast to an established algorithm for sequence comparison, the Needleman-Wunsch algorithm. The success of the PHMM in computational biology has encouraged linguists to adapt this model for linguistic purposes. This thesis has shown that it is indeed fruitful to use this model for linguistic purposes. In an eight-fold cross validation experiment words from the ASJP-database are aligned and then clustered into cognate sets. Evaluated against a gold standard created with the `LexStat` algorithm the log-odds version of the Viterbi algorithm for PHMMs yields better results than the Needleman-Wunsch algorithm. The Forward algorithm for PHMMs produces results which are slightly below the ones of Needleman-Wunsch alignment. As discussed below, the Forward algorithm is expected to improve phylogenetic reconstruction, but it performs worse in comparison to Needleman-Wunsch alignment in the cognate classification task. This may be due to the data-set. Since all languages in consideration are closely related, all the words are closely related as well. In contrast to the Viterbi algorithm, the Forward algorithm sums over all alignments. The close relationship between the languages thus inflates the Forward score and the overall difference between the alignment scores shrinks. The increasing amount of similarity complicates the task for the clustering algorithm. In contrast, the respective single best alignment scores are not inflated by this close relationship that much. Therefore the task for the clustering algorithm gets easier. A higher threshold for UPGMA algorithm improves the result of the cognate clusters for the Forward algorithm.[1] This supports the hypothesis that the alignment scores of the Forward algorithm are inflated by the close relationship. In applications with a broader set

[1]A threshold of 0.98 yields slightly better results for both versions of the Forward algorithm on cognate clustering than Needleman-Wunsch for a threshold of 0.8.

of data the Forward algorithm is expected to improve the results. Due to these results, the scope of Forward algorithm seems to be more in applications where overall relatedness is more important than specific relatedness.
The good performance of Pair Hidden Markov Models in comparison to Needleman-Wunsch is certainly due to the trained substitution probabilities. The substitution probabilities for PHMMs are generated by Baum-Welch training. This parameter estimation procedure is of course more sophisticated than the estimation of the log-odds for the Needelman-Wunsch algorithm. This makes the PHMM more sensible to subtle sound differences.

Pair Hidden Markov Models, especially the Viterbi algorithm, outperform Needleman-Wunsch alignment on cognate detection. The Forward algorithm does not lead to an improvement of cognate classification. As discussed, the scope of the Forward algorithm is not in areas of cognate detection. But nevertheless, this algorithm may prove its usefulness in other applications. The PHMM offers more variability to the task of sequence comparison, since different algorithms can be used.

5.1 Further issues

In applications of computational historical linguistics, Needleman-Wunsch alignment can be used to derive distance measures between languages. To measure the distance between two languages, Jäger (2013b, 2013a) uses a Swadesh list for each language and compare these two. The Needleman-Wunsch algorithm is designed to return the single most probable alignment for two sequences. Words are compared pairwise and these values are stored in a matrix. The values resulting from Needleman-Wunsch alignment in the diagonal of this matrix are the scores for words with the same meaning. For two related languages the values in the diagonal should differ from the off-diagonal values since a high similarity leads to a small distance. These distance matrices for languages are used for phylogenetic reconstruction. As shown by Jäger (2013b), the phylogenetic reconstruction based on distance matrices, which result from a weighted Needleman-Wunsch alignment, outperforms distance based methods which are based on a normalized Levensthein distance (Jäger, 2013b). Needleman-Wunsch alignment is set-up to always find the best alignment between two strings. In contrast to the single best alignment with Needleman-Wunsch, the Forward or the Backward algorithm for Pair Hidden Markov Models computes a score for all alignments between two strings. This alignment score of the Forward or Backward algorithm can be compared via log-odds ratio to a null model. This null model yields a probability that two sequences occur as a pair while there is no underlying relationship (c.f. Wieling, 2007, p. 50). So, the log-odds ratio give a likelihood that two sequences are related by not assuming some unspecified alignment. It is important to notice that this method does not presume a specific (best) alignment. By the use of this technique the alignment should be more sensible for close and distant relationships between languages. Given that this is the case and given the fact, that PHMM methods outperform Nedlemann-Wunsch alignment for cognate recognition, phylogenetic reconstruction based on PHMM alignments should yield better results than reconstructions based on Needleman-Wunsch alignment. This issue has to be tackled in further research.

5.2 Conclusion

The previous chapters have given an overview of the general idea of sequence comparison (chapter 2), a stochastic model for sequence comparison (chapter 3) and an evaluation of an experiment performed with this model (chapter 4). The experiments were carried out on the basis of the ASJP-database.
In chapter 2, it has been shown that sequence comparison is important for linguistic applications. Since there is an increasing amount of word lists available, it is necessary to have automated techniques for sequence comparison and related tasks. After pointing out the sequential nature of the linguistic sign, the basic idea of alignment was presented. The well known Needleman-Wunsch algorithm for sequence alignment was introduced afterwards. To account for certain drawbacks of this algorithm structural and substantial extensions as well were introduced.
Chapter 3 introduces Pair Hidden Markov Models. Pair Hidden Markov Models belong, as the name suggests, to the family of Markov Models. Pair Hidden Markov Models were first brought up in the context of bioinformatics and were successfully used there for the alignment of DNA sequences. Since Pair Hidden Markov Models are an extension of Hidden Markov Models, the algorithms for Hidden Markov Models can be translated for the needs of Pair Hidden Markov Models. These

algorithms were shown and explained.
The experiment presented in chapter 4 compares the performance of alignment carried out with Pair Hidden Markov Models and Needlemann-Wunsch alignment. In an eight-fold cross validation experiment, the words in the test set are aligned and then clustered into cognate sets. These clusters were evaluated against a gold standard which was created with the `LexStat` algorithm. The experiments have shown that the alignments carried out with Pair Hidden Markov Models serve better as an input for the clustering algorithm than the ones of Needleman-Wunsch.

This performance of the Pair Hidden Markov Model for cognate clustering tasks is encouraging for further applications. In contrast to standard Needleman-Wunsh alignment Pair Hidden Markov Models bring more variability to the table. Thus, the stochastic method for sequence comparison is able to improve results in phylogenetic applications. As pointed out above, the use of alignments carried out with Pair Hidden Markov Models may lead to an improvement of distance based phylogenetic reconstruction. Phylogenetic reconstruction is a topic that has gained more and more interest in the last years. Therefore, the results presented here contribute to this line of research.

References

Baldauf, S. L. (2003, June). Phylogeny for the faint of heart: a tutorial. *Trends in Genetics*, *19*(6), 345-351.

Baum, L. E., Petrie, T., Soules, G., & Weiss, N. (1970). A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains. *The Annals of Mathematical Statistics*, *41*(1), 164–171.

Borodovsky, M., & Ekisheva, S. (2006). *Problems and solutions in biological sequence analysis* (1st ed.). Cambridge University Press.

Brown, C. H., Holman, E. W., Wichmann, S., & Velupillai, V. (2008). Automated classification of the world's languages: A description of the method and preliminary results. *STUF – Language Typology and Universals*, *61*(4), 285-308.

Carstensen, K.-U., Ebert, C., Endriss, C., Jekat, S., Klabunde, R., & Langer, H. (Eds.). (2004). *Computerlinguistik und sprachtechnologie : eine einführung* (2nd ed.). München: Elsevier, Spektrum Akad. Verl.

Clark, A. (2001, July). Learning morphology with Pair Hidden Markov Models. In *Proc. of the student workshop at the 39th annual meeting of the association for computational linguistics* (p. 55-60). Toulouse, France.

Covington, M. A. (1996). An algorithm to align words for historical comparison. *Computational Linguistics*, *22*(4), 481-496.

Davenport, M., & Hannahs, S. J. (1998). *Introducing phonetics and phonology.* London, New York, Sydney, Auckland: Arnold.

Durbin, R., Eddy, S., Krogh, A., & Mitchison, G. (2001). *Biological sequence analysis: probabilistic models of proteins and nucleic acids* (Repr. ed.). Cambridge: Cambridge Univ. Press.

Ende, M. (1996). *Momo oder die seltsame Geschichte von den Zeit-Dieben und von dem Kind, das den Menschen die gestohlene Zeit zurückbrachte : ein Märchen-Roman* (41st ed.). Stuttgart: Thienemann.

Gildea, D., & Jurafsky, D. (1996). Learning bias and phonological-rule induction. *Computational Linguistics*, *22*(4), 497-530.

Gray, R. D., & Atkinson, Q. D. (2003, November). Language-tree divergence times support the anatolian theory of indo-european origin. *Nature*, *426*(6965), 435-439.

Gusfield, D. (1997). *Algorithms on strings, trees and sequences : computer science and computational biology.* Cambridge: Cambridge Univ. Press.

Hamming, R. W. (1950). Error Detecting and Error Correcting Codes. *Bell System Technical Journal*, *26*(2), 147–160.

Henikoff, S., & Henikoff, J. G. (1992, November). Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences of the United States of America*, *89*(22), 10915–10919.

Jäger, G. (2013a). *Evaluating distance-based pyhlogenetic algorithms for automated language classification.* (manuscript, University of Tübingen)

Jäger, G. (2013b). *Phylogenetic inference from word lists using weighted alignment with empirically determined weights.* (manuscript, University of Tübingen and Swedish Collegium for Advanced Study)

Jurafsky, D., & Martin, J. H. (2009). *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition* (2. ed., [Pearson International Edition] ed.). Upper Saddle River, NJ [u.a.]: Prentice Hall, Pearson Education International.

Knudsen, B., & Miyamoto, M. M. (2003). Sequence alignments and pair hidden markov models using evolutionary history. *Journal of Molecular Biology*, *333*(2), 453 - 460.

Kondrak, G. (2000). A new algorithm for the alignment of phonetic sequences. In *Proceedings of the 1st north american chapter of the association for computational linguistics conference* (pp. 288–295). Stroudsburg, PA, USA: Association for Computational Linguistics.

Kondrak, G. (2002). *Algorithms for language reconstruction.* Unpublished doctoral dissertation, Graduate Department of Computer Science, University of Toronto, Toronto.

Levenshtein, V. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, *10*, 707-710.

List, J.-M. (2012a, April). Lexstat: Automatic detection of cognates in multilingual wordlists. In *Proceedings of the eacl 2012 joint workshop of lingvis & unclh* (pp. 117–125). Avignon, France: Association for Computational Linguistics.

List, J.-M. (2012b). *Sequence comparison in historical linguistics.* Unpublished doctoral dissertation, Heinrich Heine Universität Düsseldorf.

Mackay, W. (2004). *Word similarity using pair hidden markov models. master's thesis.* Unpublished master's thesis, University of Alberta.

Mackay, W., & Kondrak, G. (2005). Computing word similarity and identifying cognates with pair hidden markov models. In *Proceedings of the ninth conference on computational natural language learning* (pp. 40–47). Stroudsburg, PA, USA: Association for Computational Linguistics.

Manning, C. D., & Schütze, H. (2003). *Foundations of statistical natural language processing* (6. print. with corr. ed.). Cambridge, Mass. [u.a.]: MIT Press.

McLachlan, G. J., & Krishnan, T. (1997). *The em algorithm and extensions.* New York [u.a.]: Wiley.

Meyer, I. M., & Durbin, R. (2002). Comparative ab initio prediction of gene structures using pair hmms. *Bioinformatics*, *18*(10), 1309-1318.

Needleman, S. B., & Wunsch, C. D. (1970, March). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, *48*(3), 443–453.

Nerbonne, J., & Heeringa, W. (1997). Measuring dialect distance phonetically. In *Proceedings of the third meeting of the acl special interest group in computational phonology* (pp. 11–18).

Rabiner, L. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*(2), 257-286.

Saussure, F. d. (1966). *Course in general linguistics* (C. Bally, Ed.). New York, Toronto, London: McGraw-Hill.

Smith, T., & Waterman, M. (1981). Identification of common molecular subsequences. *Journal of Molecular Biology*, *147*(1), 195 - 197.

Sokal, R. R., & Michener, C. D. (1958). A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, *28*, 1409–1438.

Torres, A., Cabada, A., & Nieto, J. J. (2003, December). An exact formula for the number of alignments between two DNA sequences. *DNA sequence : the journal of DNA sequencing and mapping*, *14*(6), 427–430.

Trask, R. L. (1999). *Key concepts in language and linguistics.* London, New York: Routledge.

Wichmann, S., Müller, A., Velupillai, V., Wett, A., Brown, C. H., Molochieva, Z., . . . Grant, A. (2012). *The asjp database (version 15).*

Wieling, M. (2007). *Comparison of dutch dialects.* Unpublished master's thesis, University of Groningen, Groningen.

Wieling, M., Leinonen, T., & Nerbonne, J. (2007). Inducing sound segment differences using pair hidden markov models. In J. Nerbonne, M. Ellison, & G. Kondrak (Eds.), *Proceedings of ninth meeting of the acl special interest group in computational morphology and phonology* (pp. 48–56). Stroudsburg, PA, USA: Association for Computational Linguistics.

Appendix A

Content of the Pair Hidden Markov Model implementation

File 1 (PairHMMFunctions.py)**:**
This is a Python class for Pair Hidden Markov Models. It incorporates an implementation of the Forward, the Backward, the Baum-Welch, the Viterbi and a log-odds version of the Viterbi and the Forward Algorithm.
It is depreciated to call this class directly. Please use the `MainPairHMM` class.

File 2 (MainPairHMM.py)**:**
This class is the main class for the Pair HMM stuff. It passes the work to all the other functions. The following functions can be called:

- Forward Algorithm
- Backward Algorithm
- Baum-Welch Algorithm
- Random model
- Viterbi Algorithm
- Log-odds version of the Viterbi Algorithm
- Log-odds version of the Forward Algorithm

File 3 (NewMain.py)**:**
Script for the single process execution of the Baum Welch training.

File 4 (NewMain(wrapper).py)**:**
Child file for the MPI implementation. This file calls the Baum-Welch implementation

File 5 (Alignment.py)**:**
This file performs the Alignment analysis.

File 6 (AuxFunctions.py)**:**
File holding a lot of functions, that were needed.

File 7 (AlignFunctions.py)**:**
File holding functions needed for the Alignment analysis.

File 8 (familySpecificLogOdds.py)**:**
Script creating log-odds for language families, based on the ASJP-matrix.

File 9 (asjpcognates.py)**:**
Script extracting the cognates out of the ASJP-matrix.

File 10 (ConvergenceChecker.py)**:**
File holding the functions, which checks if the Baum-Welch algorithm has converged

File 11 (DataPreparation.py)**:**
Some wrapper functions for the preparation of the training data.