Eberhard Karls Universität Tübingen
Mathematisch-Naturwissenschaftliche Fakultät
Wilhelm-Schickard-Institut für Informatik

Bachelorarbeit Bioinformatik

Bigramm-Alignierung und ihre Anwendung in der historischen Linguistik

Nancy Retzlaff
15. August 2013

Gutachter

Dr. Kay Nieselt
Wilhelm-Schickard-Institut für Informatik
Universität Tübingen

Betreuer

Dr. Lydia Steiner

Härtelstr. 16-18
Bioinformatik
Universität Leipzig

Dr. Christian Höner zu Siederdissen
Währingstr. 17, A-1090 Wien, Österreich
Institut für Theoretische Chemie
Universität Wien

Retzlaff, Nancy:
Bigramm-Alignierung und ihre Anwendung in der historischen Linguistik
Bachelorarbeit Bioinformatik
Eberhard Karls Universität Tübingen
Bearbeitungszeitraum: 15. April 2013 – 15. August 2013

Zusammenfassung

Die vorliegende Arbeit beschäftigt sich mit der Frage nach dem optimalen Scoring-Modell, unter Betrachtung von Bigrammen. Dazu sollen in den ersten Kapiteln Grundlagen erklärt werden, auf welche sich Auswertung und Diskussion beziehen.

Nach einer kurzen „Einleitung“ werden im Kapitel „Linguistischer Hintergrund“ ein Einblick in die vergleichende Sprachwissenschaft gegeben und reguläre Wortveränderungen eingeführt, sowie verschiedene frühe Wortalignment-Algorithmen vorgestellt.

Der „Bioinformatische Hintergrund“ befasst sich mit komplexeren Alignierungsalgorithmen, wie die nach Needleman-Wunsch, Smith-Waterman und Gotoh.

Im „Methoden“-Teil werden dann zum einen die den Algorithmen zugrundeliegenden Scoring-Modelle erläutert und zum anderen ein Einblick in die Theorie der Formalen Sprachen gewährt.

Auf die im Verlauf der Arbeit entstandenen Ergebnisse wird dann in „Auswertung“ und „Diskussion“ eingegangen und Bezüge zur Linguistik hergestellt.

Danksagung

An dieser Stelle möchte ich zuerst Prof. Peter Stadler danken, bei dem man mit Ideen und Kritik Unterstützung und Feedback finden konnte.

Desweiteren bedanke ich mich bei Dr. Kay Nieselt dafür, dass sie mir die Freiheit und Möglichkeit gegeben hat, dieses spannende Thema zu untersuchen.

Danke an Lydia und Christian, die mich immer unterstützt und aufgebaut haben – auch wenn der Stock in meinem Rücken manchmal schmerzte.

Vielen Dank auch an Christoph; deine vielen Sandwich-Kritiken haben mir sehr geholfen.

An meine Familie und Freunde, welche da und geduldig waren während der Arbeitsphase: danke!

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

Der Gedanke, dass es zwischen ausgewählten Sprachen innerhalb einiger Wörter Ähnlichkeiten gibt, ist nicht neu. Bereits Gottfried Wilhelm Leibniz (1646 – 1716) stellte dies fest:

> „Höchst bemerkenswert aber ist die Tatsache, daß in einem großen Teil unseres Kontinents in den gegenwärtigen Sprachen die Spuren einer alten, weitverbreiteten Sprache vorhanden sind; […].“ [Arens1969, S.97]

Dieses Zitat macht deutlich, dass schon zu Beginn der Zeit der Aufklärung Theorien entstanden, nach denen Sprachen eine gemeinsame Wurzel haben könnten. Leibniz macht dies am Beispiel der eurasischen Sprachfamilie deutlich:

> „[…] Alle diese [die britische, gallische, deutsche] Sprachen stammen aus einer Quelle und können als Variationen der nämlichen Sprache angenommen werden […].“ [Arens1969, S.69]

Leibniz nahm an, dass diese Ursprache von einem Volk gesprochen wurde, welches sich zu Zeiten der Völkerwanderung an ökologisch günstigen Stellen niederließ und dort seine Wanderung stoppte.

Leibniz war, wie Alargov bemerkte [Alargov2007], anscheinend auch der Sprachwandel bereits bewusst:

> „Einige Hunderte ähnlicher Menschen könnten leicht in ihren Nachkommen eine neue Sprache hervorbringen, so wie die Sprache der heutigen Räter aus dem Italienischen oder Gallischen entstellt ist, wie beide ihrerseits wieder aus dem Lateinischen: solche wiederholte Entstellungen von Entstellungen löschen schließlich die ursprünglichen Grundzüge aus.“ [Arens1969, S.97]

Hier lassen sich deutlich die ersten Grundzüge der historischen Linguistik erkennen. Lyle Campbell schreibt am Beginn eines seiner Bücher:

> „What is historical linguistics? Historical linguists study language change.“ [Campbell2004, S.1]

Die historische Linguistik ist also die Wissenschaft, welche den Sprachwandel untersucht. Dabei stehen jüngere Lautverschiebungen genauso im Mittelpunkt wie Veränderungen der Sprachen über Jahrhunderte hinweg.

> “Approaching sound change from the procedural perspective, different mechanisms of sound change may be identified.“ [List2012, S.27]

Erste solche Studien wurden von Jacob Grimm (1785 - 1863) durchgeführt. Er entdeckte sogenannte *sound shifts* - Veränderungen in der Aussprache und damit über die Zeit auch in der Rechtschreibung - bei der Entwicklung vom Proto-Indoeuropäischen zum Proto-Germanischen. Diese Beobachtung wird *Grimm's Law* genannt.

Ein solches Gesetz kann jedoch lediglich über Sprach- und Wortvergleiche erkannt werden. Hier schlägt die historische Linguistik eine Brücke zu den Sequenzanalysemethoden der Bioinformatik, da sich in beiden ähnliche Methoden finden lassen, um Zeichenketten zu analysieren. Damit können Wörter in der Linguistik fast analog zu DNA-/ RNA- oder Proteinsequenzen in der Bioinformatik behandeln lassen. Eine solche Sequenzanalysemethode [Hamming1950] wurde bereits 1950 von Richard Hamming (1915 – 1998) entwickelt, welcher untersuchte, wie sich Bitsequenzen bei ihrer Verarbeitung verhalten. Dabei verwendete er „Error Detecting and Error Correcting Codes“. Diese Codes unterstützen das Erkennen und Korrigieren von Fehlern in der Übertragung von Bitsequenzen.

In der Literatur haben sich verschiedene Distanzmaße etabliert, wie zum Beispiel die Hamming-Distanz [Hamming1950] und der Levenshtein-Distanz [Levenshtein1966].

Erste Implementierungen zu Sequenzanalyseverfahren wurden bereits in ALINE [Kondrak2000], [Steiner2011] und LingPy [ListMoran2013] realisiert. Diese Ansätze basieren jedoch auf buchstabenweisen Vergleichen. Um den Kontext bei Lautänderungen mit einzubeziehen, wurde im Rahmen dieser Arbeit ein Bigrammansatz entwickelt, welcher in einer ähnlichen Weise bereits in BlastR [Busotti2011] zu finden ist.

Als Datengrundlage wurde die indoeuropäische Sprachfamilie gewählt, da diese bereits recht gut erforscht ist und sich die Methode hier gut überprüfen lässt. Die für die Auswertung wichtigen linguistischen Grundlagen werden im folgenden Kapitel erläutert.

Kapitel 2

Linguistischer Hintergrund

Die historische Linguistik beschäftigt sich mit Vergleichen von Sprachen auf Wortebene. Dabei ist aufgefallen, dass es ein paar reguläre Veränderungen der Wörter gibt, welche sehr häufig auftauchen. Diese können zum einen Deletionen sein (Vgl. Tabelle 1):

- *Syncope*: Verlust eines Vokals in der Mitte des Wortes.
- *Apocope*: Verlust eines Lautes, in der Regel eines Vokals, am Ende eines Wortes
- *Aphaeresis*: Verlust eines Lautes, in der Regel eines Vokals, am Anfang eines Wortes

Desweiteren gibt es ebenso Insertionen:

- *Prothesis:* Einfügen eines Lautes am Anfang eines Wortes
- *Anaptyxis:* Einfügen eines Vokals zwischen zwei Konsonanten
- *Excrescence:* Einfügen eines Konsonanten zwischen zwei anderen Konsonanten
- *Paragoge:* Einfügen eines Lautes, in der Regel eines Vokals, am Ende eines Wortes

Diese Definitionen sind [Campbell2004] entnommen.

Tabelle 1: Veränderung in der Position eines Wortes für einen Vokal *α*, Konsonant β und Buchstabensequenzen σ und τ, wobei ein Wort eine Kombination aus diesen ist.

Art der Veränderung	Anfang	Mitte	Ende
Deletion	Aphaeresis: $\alpha\sigma\tau \rightarrow \sigma\tau$	Syncope: $\sigma\alpha\tau \rightarrow \sigma\tau$	Apocope: $\sigma\tau\alpha \rightarrow \sigma\tau$
Insertion	Prothesis: $\sigma\tau \rightarrow \alpha\sigma\tau$	Anaptyxis: $\sigma\tau \rightarrow \sigma\alpha\tau$ Excrescence: $\sigma\tau \rightarrow \sigma\beta\tau$	Paragoge: $\sigma\tau \rightarrow \sigma\tau\alpha$

Grimm's Law

Diese Veränderungen behandeln jedoch nur die Verlängerung oder Verkürzung von Wörtern. Ein Beispiel für einen *sound shifts*, also Verschiebungen in der Aussprache, welche sich nicht zwingend auf eine Änderung in der Wortlänge auswirkt, wäre *Grimm's Law*. Dieses ist benannt nach seinem Entdecker Jacob Grimm, dem älteren der beiden Grimm Brüder. Er entdeckte bereits zu seinen Lebzeiten (1785 – 1863), dass es Regelmäßigkeiten in den Lautverschiebungen gab.

> „Grimm's Law is an extremely important set of sound changes in historical linguistics; it is intimately involved in the history of the comparative method […]. "
> Campbell, Historical Linguistics, S. 49

Die hier erwähnte *comparative method* beschäftigt sich mit den oben erwähnten Wortvergleichen, die Aufschlüsse über Verwandtschaft und die teilweise Rekonstruktion einer Protosprache ermöglicht.

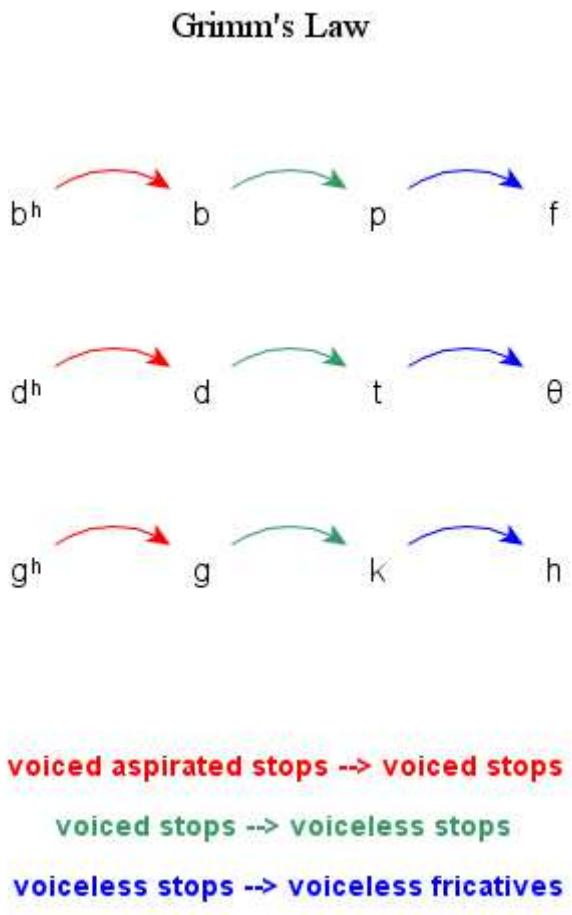

Abbildung 1: Sound shifts für Grimm's Law

Für die zeitliche Abfolge von *Grimm's Law* gibt es zwei mögliche Szenarien, in denen die Lautverschiebungen stattgefunden haben könnten. Da dies jedoch chronologisch nicht mehr nachvollziehbar ist, könnten zuerst die *voiced aspirated stops* zu *voiced stops* geworden sein und damit eine *push chain* ausgelöst haben. In einem anderen Szenario veränderten sich zuert die *voiceless stops* und wurden zu *voiceless fricatives*, wodurch eine *pull chain* in Gang kam. *Push chain* heißt in diesem Falle, dass die Verschiebung eines Lautes in Richtung eines anderen Lautes geschah, weshalb dieser sich ebenfalls verändern musste, um die Unterscheidung dieser Laute zu gewährleisten (zum Beispiel $b^h \rightarrow b$, weshalb sich der Lauts des bs verschieben musste, also $b \rightarrow p$, usw.). Im Fall der *pull chain* entstand ein Raum zwischen zwei Lauten (zum Beispiel: $p \rightarrow f$, Raum zwischen b und p wird größer). Um diese Lücke zu füllen, bewegt sich der b-Laut in Richtung des p-Lautes.

Weitere Lautverschiebungen sind zum Beispiel:

- *Metathesis*: das Vertauschen von Lauten

- *Fusion*, das Verschmelzen von Lauten
- *Fission*, das Spalten von einem Laut in zwei separate Laute
- *Assimilation*, das Angleichen von zwei oder mehreren Lauten

Diese Veränderungen sind ebenfalls möglich, jedoch für diese Arbeit von keiner großen Relevanz, weshalb hier auf die genaue Erläuterung verzichtet wird.

Betrachtet man nun verschiedene Sprachen aus dem indoeuropäischen Sprachraum, werden diese *sound shifts* offensichtlich. Ein Beispiel hierfür ist ‚padre' (sp.: Vater), ‚vader' (holl.: Vater), ‚fader' (dän.: Vater) und ‚father' (engl.: Vater). Auf dieses wird später in der Auswertung noch näher eingegangen.

Nun kann man auch Aussagen über den Verwandtschaftsgrad zweier Sprachen durch Betrachtung ihrer Wörter treffen. Die Vergleichbarkeit resultiert daraus, dass Wörter sowohl geschrieben als auch gesprochen in Sequenzen vorliegen.

String Metriken

Ein erster Algorithmus zum Vergleich von Wörtern wurde von Wagner und Fischer 1974 implementiert. Dieser berechnet die Levenshtein Distanz zweier Strings, indem diese in ihre einzelnen Buchstaben zerlegt werden und dann buchstabenweise verglichen werden. Für Strings A und B wird hier überprüft, ob die Überführung von a_i zu b_j $(A\langle i\rangle \rightarrow B\langle j\rangle)$ günstiger ist als das Einfügen eines *gaps* (Λ) [WagnerFischer1974]:

1. $D[0,0] := 0;$
2. $\boldsymbol{for}\ i := 1\ \boldsymbol{to}\ |A|\ \boldsymbol{do}\ D[i,0] := D[i-1,0] + \gamma(A\langle i\rangle \rightarrow \Lambda);$
3. $\boldsymbol{for}\ j := 1\ \boldsymbol{to}\ |B|\ \boldsymbol{do}\ D[0,j] := D[0,j-1] + \gamma(\Lambda \rightarrow B\langle j\rangle);$
4. $\boldsymbol{for}\ i := 1\ \boldsymbol{to}\ |A|\ \boldsymbol{do}$
5. $\quad \boldsymbol{for}\ j := 1\ \boldsymbol{to}\ |B|\ \boldsymbol{do}\ \boldsymbol{begin}$
6. $\quad\quad m_1 := D[i-1,j-1] + \gamma(A\langle i\rangle \rightarrow B\langle j\rangle);$
7. $\quad\quad m_2 := D[i-1,j] + \gamma(A\langle i\rangle \rightarrow \Lambda);$
8. $\quad\quad m_3 := D[i,j-1] + \gamma(\Lambda \rightarrow B\langle j\rangle);$
9. $\quad\quad D[i,j] := \min(m_1, m_2, m_3);$
10. $\quad\quad \boldsymbol{end};$

Hier ist $\gamma(x \to y)$ die Kostenfunktion für das Ersetzen von x durch y. Sollte $x = y$ gelten, so ist $\gamma(x \to y) = 0$. Im Falle von $x \neq y$ wird $\gamma(x \to y) = 1$ gesetzt. In der Zelle $D[|A|, |B|]$ steht nach dem Ausführen des Algorithmus‘ die kleinste Distanz von A und B. Möchte man nun noch das optimale Alignment beider Strings berechnen, so benötigt die Funktion noch einen Teil, der allgemein als *backtracking* bezeichnet wird. Dieser geht von der Zelle mit dem optimalen Score aus denjenigen Weg durch die Tabelle, der die geringsten Kosten zwischen zwei Zellen enthält [WagnerFischer1974]:

1. $i := |A|; j := |B|;$
2. **while** $(i \neq 0 \,\&\, j \neq 0)$ **do**
3. **if** $D[i,j] = D[i-1,j] + \gamma(A\langle i\rangle \to \Lambda)$ **then** $i := i - 1;$
4. **else if** $D[i,j] = D[i,j-1] + \gamma(\Lambda \to B\langle j\rangle)$ **then** $j := j - 1;$
5. **else begin**
6. **print**$((i,j))$;
7. $i := i - 1; j := j - 1;$
8. **end**;

Ein mögliches Alignment zwischen engl. ‚monday‘ und dt. ‚Montag‘ würde dann wie folgt aussehen:

Tabelle 2: Berechnung der Wagner-Fischer-Distanz für das Beispiel ‚monday‘ (engl.) und ‚Montag‘ (dt.)

	Λ	M	O	N	T	A	G
Λ	**0**	1	2	3	4	5	6
M	1	**0**	1	2	3	4	5
O	2	1	**0**	1	2	3	4
N	3	2	1	**0**	1	2	3
D	4	3	2	1	**1**	2	3
A	5	4	3	2	2	**1**	2
Y	6	5	4	3	3	2	**2**

Das Ergebnis des *backtrackings* ist hier fett markiert. Daraus ergibt sich folgende Alignierung:

A: $M\ O\ N\ D\ A\ Y$

$\quad\quad\ |\ \ |\ \ |\quad\ \ |$

B: $M\ O\ N\ T\ A\ G$

Hier geben die Striche zwischen Buchstaben in String A und B an, welche Buchstaben gleich sind. Würde man das Scoringmodell etwas abwandeln und auch $D \rightarrow T$ als gültige Ersetzung zulassen, würde das Alignment einen Strich mehr zwischen diesen beiden Buchstaben besitzen. Formal betrachtet, werden die entsprechenden Buchstaben in einen Vektor V gespeichert, welcher aus Paaren (i, j) mit Indizes besteht, mit i als Index des Buchstabens in A, und j als Index des Buchstabens in Sequenz B. Für das oben gewählte Beispiel ergibt sich damit:

$$V = \big((1,1),(2,2),(3,3),(5,5)\big)$$

Bei der Betrachtung der entsprechenden Buchstaben fällt auf, dass einige ausgelassen werden. Hier stößt man schnell auf das Problem des *Substrings* und der *Subsequenz*. Bevor wir eine formale Definition geben, wollen wir das Phänomen zuerst anhand des Beispiels erklären. Ein Substring ist eine Sequenz innerhalb eines Strings, welche ohne jegliche Lücken auftaucht. Diese Konvention gilt bei der Subsequenz nicht. Deshalb ist auch jeder Substring automatisch eine Subsequenz, umgekehrt jedoch nicht. Also wäre $\big((1,1),(2,2),(3,3)\big)$ der größte gemeinsame Substring von A und B. V komplett betrachtet ist jedoch nur die gemeinsame Subsequenz.

Formal gesehen erhält man eine Subsequenz s aus einem String T durch Löschen von Buchstaben an beliebiger Stelle, ohne dabei die Reihenfolge zu verändern. Für einen Substring t gilt weiter, dass von T nur Buchstaben vom Anfang oder Ende gelöscht werden dürfen [List2012]:

- „*t* is a *subsequence* 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" [List2012]
- " *t* is a *substring* 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*" [List2012]

String Alignment unter phonetischer Transkription

Ähnlich zu dem Algorithmus von Wagner und Fischer hat auch Kondrak [Kondrak2000] einen Algorithmus implementiert, welcher unter dem Namen ALINE bekannt ist. Hier werden Daten, also Wörter aus verschiedenen Sprachen, in phonetischer Transkription als Input verwendet. Phonetische Transkription bedeutet, dass Wörter in eine Schreibweise gebracht werden, welche die Aussprache dieser am besten wiedergibt. In der Regel wird hier das Internationale Phonetische Alphabet (IPA) verwendet. Dieses bildet das Gegenstück zur orthographischen Schreibweise, in welcher zum Beispiel auch dieser Text verfasst ist. Dabei verwendet Kondrak zum Bewerten von Buchstaben ihren linguistischen Hintergrund, also den Ort im Stimmapparat, wo der dem Buchstaben entsprechende Laut gebildet wird und welche Bewegungen seine Aussprache dort verursacht.

> "The algorithm performs better on cognate alignment, in terms of accuracy and efficiency, than other algorithms reported in the literature." [Kondrak2000]

Kondrak überprüfte die Richtigkeit seines Algorithmus anhand der produzierten Alignments von *Kognaten*. Kognaten sind Wörter in verschiedenen Sprachen, welche denselben Ursprung haben. Betrachtet man also den Stammbaum der Sprachen, gab es eine Muttersprache, welche bereits ein Wort enthielt, das sich im Laufe der Zeit innerhalb der einzelnen Sprachen abgewandelt hat. Es bleibt jedoch ein Wortstamm zurück, welcher sich unverkennbar auf diese Muttersprache zurückführen lässt und deshalb die Bestimmung von Verwandtschaftsgraden einiger Sprachen lässt. Hier drängt sich die Frage auf, weshalb nicht alle Wörter aus Geschwistersprachen kognat sind. Die Antwort ist jedoch sehr simpel: aufgrund von Sprachkontakt. Dieser bewirkt, dass sich zwei fremde Sprachen sehr oft verändern und sich angleichen, was man Entlehnung nennt. Dabei können auch Wörter von einer Sprache in eine andere übertragen werden, welche man als Lehnwörter bezeichnet. Ein Beispiel für ein Lehnwort wäre das Englische ‚mountain', welches aus dem Französischen (‚montagne') entlehnt und dabei an die Schreibweise und Aussprache des Englischen angepasst wurde.

Für die Implementierung von ALINE gibt es verschiedene Möglichkeiten, das Alignment durchzuführen. Die oben vorgestellte Variante gibt ein globales Alignment zurück, wobei

aber auch eine lokale Alignierung möglich ist. Wie genau ein solcher lokaler Algorithmus aussieht, wird im Laufe des folgenden Kapitels beschrieben.

Ein Beispiel für die Sprachfamilie ist das Indogermanische, wozu unter anderem Deutsch, Englisch, Dänisch und Schwedisch gehören.

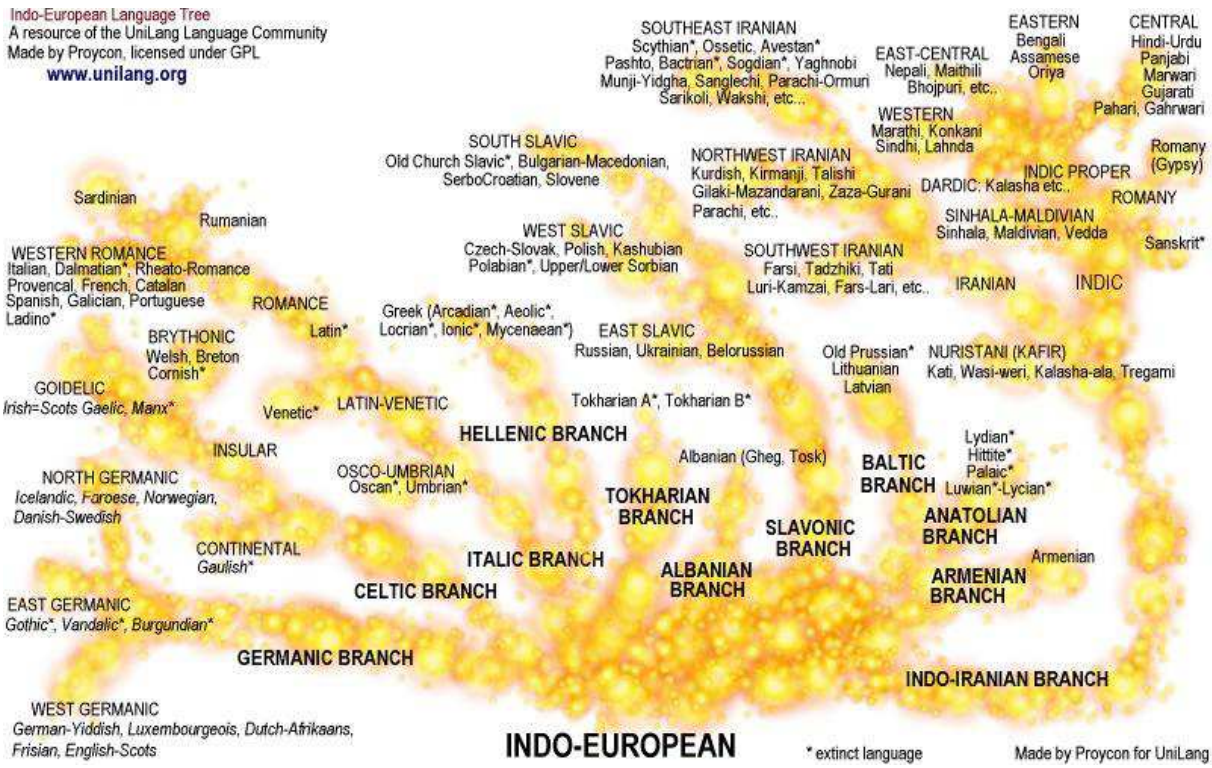

Abbildung 2: Stammbaum indoeuropäischer sprachen (www.unilang.org)

Wie man in Abbildung 2 sehen kann, gehören die meisten in Europa gesprochenen Sprachen (außer Finnisch, Ungarisch und Baskisch) zur indoeuropäischen Sprachfamilie [Jacob2003]. Einige Sprachen der indischen Untergruppe werden auch außerhalb von Europa gesprochen, haben ihre Wurzeln trotz dessen im Indoeuropäischen.

Kapitel 3

Bioinformatischer Hintergrund

Ausgehend von den ersten Entwicklungen mit der Berechnung der Hamming Distanz, in der zwei gleich lange Zeichenketten auf ihre Gleichheit untersucht werden, kam später der Levenshtein Algorithmus, welcher bereits das Einfügen von gaps erlaubte und somit auch Strings verschiedener Länge als Input zuließ.

> „The smallest unit of comparison is a pair of amino acids, one from each protein.“ [NeedlemanWunsch1970]

Wenn man sich nun biologische Sequenzen anschaut, ist die kleinste vergleichbare Einheit entweder ein Paar von Basen, beim Vergleich von DNA- oder RNA-Sequenzen, oder ein Paar aus zwei Aminosäuren. Dabei betrachtet man immer eine Base/ Aminosäure aus der ersten Sequenz und eine aus der zweiten. Dazu werden die Sequenzen vom 5‘-Ende beziehungsweise N-Terminus aus durchnummeriert und somit jede Base/ Aminosäure eindeutig beschreibbar gemacht. Zum Beispiel wäre A_i die i-te Stelle der Sequenz A, wie auch schon zuvor im Kapitel linguistischer Hintergrund beschrieben.

Auch hier sind Insertionen und Deletionen zu finden. Da biologische Sequenzen jedoch sehr lang sind, liegen diese fast immer in der Mitte der Sequenz. Im Vergleich zu linguistischen Sequenzen hat es deshalb hier wenig Sinn den Ort der Mutation relativ zum kompletten String zu betrachten.

Algorithmen zum Sequenzvergleich

Eine Methode zum Sequenzvergleich ist die von Needleman und Wunsch. Sie entwickelten einen Algorithmus, welcher für globale Alignments verwendet wird. Globales Alignment heißt, dass zwei Sequenzen derart aneinander gelegt werden, dass alle Einheiten möglichst gut aligniert werden. Wie in der linguistischen Einleitung werden auch hier die Sequenzen A und B in einem zweidimensionalem Array M aufgetragen und jede einzelne Zelle gefüllt:

1. $M[0,0] := 0$;
2. **for** $i := 1$ **to** $|A|$ **do** $M[i,0] := M[i-1,0] + g$;
3. **for** $j := 1$ **to** $|B|$ **do** $M[0,j] := M[0,j-1] + g$;
4. **for** $i := 1$ **to** $|A|$ **do**
5. **for** $j := 1$ **to** $|B|$ **do begin**
6. $match := M[i-1,j-1] + score(A_i, B_j)$;
7. $insert := M[i-1,j] + g$;
8. $delete := M[i,j-1] + g$;
9. $M[i, j] := \max(match, insert, delete)$;
10. **end**;

Dabei ist g der Wert, welcher für das Einfügen eines *gaps* addiert werden muss. Die Werte $match, insert$ und $delete$ beschreiben die Operationen, die zur Umformung von A in B vorgenommen werden müssen. Die letzte Funktion $score(A_i, B_j)$ sucht aus einer gegebenen Bewertungstabelle (engl. *scoring matrix*) den Wert für das Ersetzen von A_i durch B_j heraus.

Eine solche Bewertungstabelle könnte zum Beispiel die BLOSUM62 Matrix (siehe Abbildung 3) sein, welche für gewöhnlich für Berechnungen bei Aminosäuresequenzen verwendet wird.

	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	
	0	-1	1	0	2	1	1	2	1	2	0	0	2	4	1	5	1	2	-2	5	C
		2	0	-2	0	-1	0	0	0	1	0	0	0	1	0	1	-1	1	1	-1	S
C	9		2	-1	-1	-1	0	0	0	0	0	0	-1	0	-1	1	0	1	1	3	T
S	-1	4		2	-2	-1	-1	0	0	-1	-1	-1	1	1	0	-1	0	0	2	1	P
T	-1	1	5		2	-1	-2	-2	-1	0	0	1	1	0	0	1	0	1	1	2	A
P	-3	-1	-1	7		2	0	-1	-2	0	1	1	0	0	-1	0	-1	1	2	4	G
A	0	1	0	-1	4		3	-1	-1	0	0	1	-1	0	-1	0	-1	0	0	0	N
G	-3	0	-2	-2	0	6		2	-1	-1	-1	0	-1	0	0	0	0	2	1	3	D
N	-3	1	0	-2	-2	0	6		1	0	0	2	2	1	-1	0	0	2	2	4	E
D	-3	0	-1	-1	-2	-1	1	6		0	-2	0	1	1	-1	0	0	1	3	3	Q
E	-4	0	-1	-1	-1	-2	0	2	5		2	-1	0	1	0	-1	0	1	2	2	H
Q	-3	0	-1	-1	-1	-2	0	0	2	5		-1	-1	0	-1	1	0	1	3	-4	R
H	-3	-1	-2	-2	-2	-2	1	-1	0	0	8		1	-2	-1	1	1	2	3	1	K
R	-3	-1	-1	-2	-1	-2	0	-2	0	1	0	5		-2	-1	-1	0	1	2	4	M
K	-3	0	-1	-1	-1	-2	0	-1	1	1	-1	2	5		-1	1	0	0	1	3	I
M	-1	-1	-1	-2	-1	-3	-2	-3	-2	0	-2	-1	-1	5		-1	0	-1	1	2	L
I	-1	-2	-1	-3	-1	-4	-3	-3	-3	-3	-3	-3	-3	1	4		0	1	2	4	V
L	-1	-2	-1	-3	-1	-4	-3	-4	-3	-2	-3	-2	-2	2	2	4		-1	-2	1	F
V	-1	-2	0	-2	0	-3	-3	-3	-2	-2	-3	-3	-2	1	3	1	4		-1	2	Y
F	-2	-2	-2	-4	-2	-3	-3	-3	-3	-3	-1	-3	-3	0	0	0	-1	6		-1	W
Y	-2	-2	-2	-3	-2	-3	-2	-3	-2	-1	2	-2	-2	-1	-1	-1	-1	3	7		
W	-2	-3	-2	-4	-3	-2	-4	-4	-3	-2	-2	-3	-3	-1	-3	-2	-3	1	2	11	
	C	S	T	P	A	G	N	D	E	Q	H	R	K	M	I	L	V	F	Y	W	

Abbildung 3: BLOSUM62 Matrix mit oberer Dreiecksmatrix, welche die Distanzen für jedes Aminosäurenpaar besitzt, und unterer Dreiecksmatrix, welche die Ähnlichkeitswerte enthält [Henikoff1992]

Mit der unteren Dreiecksmatrix dieser BLOSUM-Matrix (Abbildung 3) ergibt sich dann folgende Tabelle (Tabelle 3) während des Needleman-Wunsch-Algorithmus‘ für die Sequenzen „Freitag“ (dt.) und „Friday“ (engl.) mit gap-Kosten von $g = -2$.

Tabelle 3: Berechnung der Needleman-Wunsch-Ähnlichkeit für das Beipsiel ‚friday‘ (engl.) und ‚Freitag‘ (dt.)

	-	F	R	E	I	T	A	G
-	**0**	-2	-4	-6	-8	-10	-12	-14
F	-2	**6**	4	2	0	-2	-4	-6
R	-4	4	**11**	**9**	7	5	3	1
I	-6	2	9	8	**13**	11	9	7
D	-8	0	7	11	11	**12**	10	8
A	-10	-2	5	9	10	11	**16**	14
Y	-12	-4	3	7	8	9	14	**13**

Das zugehörige Alignment enthält ein gap.

$$
\begin{array}{l}
F\ R\ E\ \ I\ T\ A\ G \\
|\ |\quad |\quad |\quad\ |\ \ |\ \ | \\
F\ R - I\ D\ A\ Y
\end{array}
$$

Vergleicht man den Algorithmus von Needleman und Wunsch mit dem zuvor im linguistischen Teil vorgestellten, fällt auf, dass man nicht mehr das Minimum der Werte bildet. Mittels der Wahl des Minimums wird die Distanz zweier Sequenzen berechnet, welche man möglichst gering halten möchte. Das Maximum hingegen wird verwendet, wenn eine möglichst große Ähnlichkeit zweier Strings berechnet werden soll.

> „Both distance and similarity measures have been designed for the comparison of pairs of biological molecules. […] These distance and/or similarity measures have been used by the biologist to obtain information about processes of molecular evolution." [SmithWaterman1981]

Dies hat auch Auswirkungen auf die Wahl der *gap*-Kosten und der Werte für ein Match. Versucht man die Distanz zweier Sequenzen zu ermitteln, werden die *gap*-Kosten positiv gewählt. Im Scoring-Modell werden hier Werte für Basen/ Aminosäuren umso kleiner, je mehr sie sich ähneln (siehe obere Dreiecksmatrix Abb. 3). Für den Wert des Alignments bedeutet dies, dass er möglichst gering sein soll. Die similarity soll jedoch einen möglichst großen Wert haben. Deshalb wird der Wert für das Einfügen eines *gaps* kleiner als null gewählt. Basen-/ Aminosäurepaare, die sehr ähnlich sind, bekommen einen sehr hohen Wert, wenn sie aligniert werden. Wie man in Abbildung 3 sehen kann, wurden hier für beide Möglichkeiten die Werte angegeben. Bei der Berechnung von Ähnlichkeit wird die untere Dreiecksmatrix verwendet, zur Ermittlung der Distanz, die obere.

Im Gegensatz zu den globalen Alignments, gibt es auch Methoden, welche lokal alignieren. Ein solcher Standard-Algorithmus aus der Bioinformatik ist der von Smith und Waterman. Diese kritisierten den Ansatz von Needleman und Wunsch:

> „In 1970 Needleman and Wunsch […] introduced their homology (similarity) algorithm. From a mathematical viewpoint, their work lacks rigor and clarity." [SmithWaterman1981]

Im selben Zug stellten sie ihren lokalen Alignment-Algorithmus vor, welcher nicht nur beim Ausfüllen der Tabelle für Sequenzen A und B Unterschiede aufweist, sondern auch beim *backtracking*. Hier wird der maximale Wert als Startpunkt gewählt:

1. $M[0,0] := 0;$
2. $\boldsymbol{for}\ i := 1\ \boldsymbol{to}\ |A|\ \boldsymbol{do}\ M[i,0] := 0;$
3. $\boldsymbol{for}\ j := 1\ \boldsymbol{to}\ |B|\ \boldsymbol{do}\ M[0,j] := 0;$
4. $\boldsymbol{for}\ i := 1\ \boldsymbol{to}\ |A|\ \boldsymbol{do}$
5. $\quad \boldsymbol{for}\ j := 1\ \boldsymbol{to}\ |B|\ \boldsymbol{do\ begin}$
6. $\quad\quad match := M[i-1,j-1] + score(A_i, B_j);$
7. $\quad\quad insert := M[i-1,j] + g;$
8. $\quad\quad delete := M[i,j-1] + g;$
9. $\quad\quad M[i,\ j] := \max(0, match, insert, delete);$
10. $\quad\quad \boldsymbol{end};$

In der Tabelle wird also kein Wert kleiner als 0 auftauchen, da man hier lokale Maxima herausfinden möchte und deshalb beim veränderten *backtracking* nur vom größten Wert bis zu einer 0 zurückkehren muss:

9. $maximum := 0; a := 0; b := 0;$
10. $\quad \boldsymbol{for}\ i := 1\ \boldsymbol{to}\ |A|\ \boldsymbol{do}$
11. $\quad \boldsymbol{for}\ j := 1\ \boldsymbol{to}\ |B|\ \boldsymbol{do\ begin}$
12. $\quad\quad \boldsymbol{if}\ M[i,j] \geq maximum\ \boldsymbol{then}\ a := i; b = j; maximum := M[i,j];$
13. $\quad \boldsymbol{while}\ (a \neq 0\ \boldsymbol{\&}\ b \neq 0)\ \boldsymbol{do}$
14. $\quad \boldsymbol{if}\ M[a,b] = M[a-1,b] + g\ then\ a := a-1;$
15. $\quad \boldsymbol{else\ if}\ M[a,b] = M\ [a,b-1] + g\ then\ b := b-1;$
16. $\quad \boldsymbol{else\ begin}$
17. $\quad\quad \boldsymbol{print}\ ((a,b));$
18. $\quad\quad a := a-1; b := b-1;$
19. $\quad\quad \boldsymbol{end};$

In a und b werden die Zeile und Spalte für die Zelle mit dem größten Wert gespeichert. Ausgehend von dieser Zelle wird dann bis zur nächsten 0 zurück gegangen. Dadurch wird ein Pfad beschrieben, welcher in der Summe nur ansteigt. Es kann natürlich zwischendurch auch vorkommen, dass ein Paar nicht gut zusammen passt. Dieses kann jedoch übergangen werden, da hier das Maximum noch nicht erreicht wurde.

Für den Smith-Waterman-Algorithmus ergibt sich dann für das oben gewählte Beispiel mit selben *gap*-Kosten und gleicher Bewertungsmatrix (Abbildung 3) die in Tabelle 4 dargestellte Scoring-Tabelle.

Tabelle 4: Berechnung der Smith-Waterman-similarity für das Beispiel ‚friday' (engl.) und ‚Freitag' (dt.)

	-	F	R	E	I	T	A	G
-	**0**	0	0	0	0	0	0	0
F	0	**6**	4	2	0	0	0	0
R	0	4	**11**	**9**	7	5	3	1
I	0	2	9	9	**13**	11	9	7
D	0	0	7	11	11	**12**	10	8
A	0	0	5	9	10	11	**16**	14
Y	0	3	3	7	8	9	14	**13**

Wie schon im vorangegangenen Kapitel erwähnt, hat Kondrak für die Benutzer seines Programms ALINE einen ähnlichen lokalen Ansatz ebenfalls implementiert.

Beide Algorithmen kann man noch erweitern, da es bei biologischen Sequenzen häufiger zum Verlust längerer Ketten kommt als nur zum Verlust einer Base/ Aminosäure. Hierfür wurde eine Modell zur Bewertung von *gap*-Kosten entwickelt, welches diesen Umstand gesondert betrachtet: die affinen *gap*-Kosten. Dabei wird das Einfügen vieler einzelner *gaps* stärker bestraft als das Einfügen ganzer *gap*-Reihen:

$$gapcost = gapopen + k * gapextension$$

Hier werden also die kompletten Kosten ($gapcost$) berechnet aus dem Strafwert, welcher beim ersten Einfügen eines *gaps* gezahlt werden muss ($gapopen$). Hinzu kommen dann für jedes weitere *gap* eine Strafe ($gapextension$), welche in der Regel geringer ist als der Wert für das $gapopen$. k steht hier nur für die Anzahl der eingefügten *gaps*.

Mit diesem Modell arbeitet auch der Gotoh-Algorithmus [Gotoh1982], der hier noch als letzte Methode für das Alignieren von Sequenzen A und B genannt werden soll:

1. $M[0,0] := 0;$
2. $\boldsymbol{for}\ j := 1\ \boldsymbol{to}\ |B|\ \boldsymbol{do}\ D[0,j] := -\infty;$
3. $\boldsymbol{for}\ i := 1\ \boldsymbol{to}\ |A|\ \boldsymbol{do}\ M[i,0] = D[i,0] = g(i);$
4. $\boldsymbol{for}\ i := 1\ \boldsymbol{to}\ |A|\ \boldsymbol{do}\ I[i,0] := -\infty;$
5. $\boldsymbol{for}\ j := 1\ \boldsymbol{to}\ |B|\ \boldsymbol{do}\ M[0,j] = H[0,j] := g(j);$
6. $i := |A|; j := |B|;$
7. $\boldsymbol{while}\ (i \neq 0\ \&\ j \neq 0)\ \boldsymbol{do}$
8. $M[i,j] := \max\begin{pmatrix} M[i-1,\ j-1] + match(A_i, B_j), \\ I[i,j], \\ D[i,j] \end{pmatrix};$
9. $I[i,j] := \max\begin{pmatrix} M[i,j-1] + gapopen + gapextension, \\ I[i,j-1] + gapextension \end{pmatrix};$
10. $D[i,j] := \max\begin{pmatrix} M[i-1,j] + gapopen + gapextension, \\ D[i-1,j] + gapextension \end{pmatrix};$
11. $\boldsymbol{end};$

Dieser Algorithmus füllt bei seiner Ausführung nicht nur eine Tabelle, wie die zuvor vorgestellten Algorithmen, sondern drei Tabellen gleichzeitig. In den Matrizen I und D werden die Kosten für das Einfügen von *gaps* gespeichert. Diese sind jedoch davon abhängig, ob zuvor bereits eine Insertion oder Deletion stattgefunden hat. In der Matrix M wird der gesamte Score vermerkt und geschaut, ob im aktuellen Schritt das Einfügen eines *gaps* günstiger ist, als das Bewerten von (A_i, B_j). Erreicht man die Kanten der *gap*-Matrizen, soll vermieden werden, dass noch beliebig viele *gaps* eingefügt werden können, weshalb diese per default auf $-\infty$ gesetzt werden.

Im folgenden Kapitel soll auf die Methoden eingegangen werden, welche in der Arbeit verwendet wurden.

Kapitel 4

Methoden

In dieser Arbeit wurde der Needleman-Wunsch-Algorithmus mit linearen Gap-Kosten verwendet [NeedlemanWunsch1970]. Im Rahmen dieser Bachelorarbeit wurde dieser Algorithmus derart modifiziert, dass nicht nur einzelne Symbole erkannt werden können, sondern immer zwei aufeinanderfolgende. Hierdurch bringt man das Wissen über den vorangehenden Buchstaben mit ein, was im Folgenden als Bigramm bezeichnet wird. Dadurch ergibt sich die Möglichkeit einen Kontext aufzubauen.

Eine Implementierung im bioinformatischen Kontext existiert bereits bei BlastR [Bussotti2011]. Das Ziel von BlastR ist es, nicht-kodierende RNA-Sequenzen zu finden. Dabei wird nach sogenannten *high-scoring pairs* (HSP) gesucht. Diese sind kurze Sequenzen von RNA-Sequenzen, welche bei häufigem Auftreten zwischen Anfragesequenz und Einträgen in der Datenbank auf eine Ähnlichkeit dieser beiden Sequenzen schließen lässt.

Scoring Modelle und ihre Implementierung

Die hier erzeugte Bewertungs-Matrix enthält ermittelte Scores für Dinukleotide, welche mittels statistischer Berechnungen erzeugt wurden. Die Betrachtung von Dinukleotiden erhöht die Sensivität. Für einen Score von Dinukleotid α in Sequenz A und Dinukleotid β in Sequenz B, werden zum einen die Häufigkeiten beider Dinukleotide benötigt und zum anderen die Häufigkeit α aligniert mit β berechnet, nachdem beide Sequenzen A und B optimal aneinander gelegt wurden. Daraus ergibt sich nun eine zu erwartende Häufigkeit für das Paar (α, β). Dies bedeutet im biologischen Kontext, mit welcher Rate α durch β substituiert wird. Im Zusammenhang mit der Linguistik gibt dieser Wert über die Wahrscheinlichkeit Auskunft, mit welcher α durch β ersetzt wurde.

$$Score(\alpha,\beta) = \log\left(\frac{Häufigkeit\big(Paar(\alpha,\beta)\big)}{Häufigkeit(\alpha) * Häufigkeit(\beta)}\right)$$

Hier kann man nun auch Aussagen über den erhaltenen Score treffen. Ein negativer Score bedeutet, dass die relative Häufigkeit von α und β verglichen mit ihren einzelnen Häufigkeiten sehr gering war und, weshalb der berechnete Quotient kleiner als 1 wurde. Dieser Quotient ist jedoch immer in der Menge von $\mathbb{R}_+$, denn das Vorkommen eines Bigrammes kann nie negativ werden. Für die komplette Scoring-Funktion bedeutet dies, dass sie jedoch wieder in $\mathbb{R}$ liegt, denn der Logarithmus ist gerade als Funktion beschrieben, welche von $\mathbb{R}_+$ auf $\mathbb{R}$ abbildet. Eine der Eigenschaften des Logarithmus‘ ist etwa die Abbildung von kleinen Zahlen zwischen 0 und 1 auf einen sehr großen Raum, wodurch diese besser separiert werden können, was numerisch relevant ist (wie unten beschrieben).

Eine weitere Eigenschaft des Logarithmus ist, dass der Logarithmus des Produktes zweier Zahlen gleich der Summe der Logarithmen der einzelnen Zahlen, für Zahlen größer gleich 0 ist.

$$\log(xy) = \log(x) + \log(y)$$

Das Problem, das hier entstehen könnte, ist numerischer Natur. Da sehr viele kleine Zahlen bei der Berechnung des Scores miteinander multipliziert werden müssten, verwendet man den log-odds-Score. Hierbei wird das Produkt vieler Zahlen ersetzt durch die Summe der Logarithmen, was numerisch stabiler ist. Die Begründung hierfür liegt in der Verwendung des IEEE 754 Standards zur Darstellung reeller Zahlen. Dieser approximiert reelle Zahlen nur mit endlicher Genauigkeit und kann deshalb das Produkt kleiner Zahlen, wie sie bei Wahrscheinlichkeiten vorkommen, oft nur ungenau darstellen.

Die Dinukleotide werden im Anschluss mit einem eindeutigen Buchstaben aus einem Alphabet mit 16 Elementen bezeichnet. Bevor nun also aligniert werden kann, müssen beide Sequenzen zuvor in dieses Alphabet überführt werden.

Der zuvor vorgestellte statistische Ansatz wurde auch in dieser Bachelorarbeit verwendet. Da die Wortlisten jedoch aus 618 Buchstaben bestanden, wurde auf eine Transkription der Buchstabenpaare verzichtet, da sonst 618^2 neue Zeichen für die Ersetzung eingeführt werden müssten.

Das in der Arbeit verwendete Scoringmodell berechnet zuerst die relativen Häufigkeiten eines jeden vorkommenden Bigramms:

$$rf_{\alpha}^{L} = \frac{n_{\alpha}^{L}}{n^{L}}$$

Wobei n_α^L die absolute Häufigkeit des Bigramms α in Sprache L ist, und n^L ist die Anzahl aller Bigramme γ in Sprache L, also

$$n^L = \sum_\gamma n_\gamma^L$$

Wobei n_γ^L der Anzahl aller nicht alignierten Bigrammen entspricht.

Analog wird die relative Häufigkeit für alignierte Bigrammpaare berechnet.

$$rf_{\alpha,\beta}^{L_1,L_2} = \frac{n_{\alpha,\beta}^{L_1,L_2}}{n^{L_1,L_2}}$$

Dabei gibt $n_{\alpha,\beta}^{L_1,L_2}$ die Häufigkeit an, welche Bigramme in den Sprachen L_1 und L_2 vorkommen und n^{L_1,L_2} die Anzahl aller Brigramme in L_1 und L_2.

Zum Schluss bildet man dann für den eigentlichen log-odds-Score σ noch den Logarithmus aus den zuvor berechneten relativen Häufigkeiten, analog zu Busotti et al. [Bussotti2011]:

$$\sigma_{L_1,L_2}(\alpha,\beta) = \log\left(\frac{rf_{\alpha,\beta}^{L_1,L_2}}{rf_\alpha^{L_1} * rf_\beta^{L_2}}\right)$$

In der Regel würde man hier einen Pseudocount einfügen, um bei Berechnungen 0 als Teiler zu vermeiden. In diesem Ansatz werden jedoch nur Bigrammpaare gespeichert, welche für Wörter aus verschiedenen Sprachen, jedoch mit gleicher Bedeutung, auch gesehen wurden. Das ausschließliche Betrachten von Wörtern mit gleicher Bedeutung soll schon zu Beginn sicherstellen, dass auch wirklich korrespondierende Paare von Bigrammen später im Modell auftauchen. Würden hier alle Wörter untereinander verglichen werden, könnten keine Entsprechungen von Paaren gefunden werden.

Dieser statistische Ansatz spricht denjenigen Bigrammpaaren einen sehr hohen Informationsgehalt zu, welche sehr selten vorkommen. Erste Auswertungen eines solchen unbeschränkten Scoringmodells zeigten jedoch, dass hier auch Bigrammpaare einen sehr guten Wert bekamen, welche keine linguistische Relevanz hatten. Dies führte dazu, dass im verwendeten Scoringmodell nur Bigrammpaare betrachtet werden, welche mindestens drei Mal vorkamen.

Bei dem Alignment kann es vorkommen, dass sogenannte *gaps* eingefügt werden müssen. Hierfür muss ein Bestrafungswert eingefügt werden. Dieser wird so gewählt, dass er kleiner

ist als der schlechteste Score für ein Alignment (-4,12428), damit lieber ein schlechteres Buchstabenpaar aligniert wird, als Wörter gar nicht zu alignieren. Daher fiel die Wahl des Bestrafungswertes für ein *gap* auf -4,2.

Da Wörter im Vergleich zu biologischen Sequenzen recht kurz sind, ist das Modell für die affinen gap Kosten möglicherweise nicht notwendig, denn das Einfügen vieler gaps würde bedeuten, dass beide Wörter möglicherweise recht wenig miteinander verwandt sind.

Alignmentalgorithmen als Grammatiken

Das eigentliche Alignment erfolgt mittels einer Variante des ADP (Algebraic Dynamic Programming [GiegerichMeyer2002]), ADPfusion [HzS2012]. Auf einem Input wird hier ein Algorithmus ausgeführt. Das heißt explizit, dass zuvor der Algorithmus in Form einer regulären Grammatik überführt wird, mit der dann ADPfusion kompiliert. Das Ergebnis ist dann also die Grammatik ausgeführt auf dem Input. Für die Bewertung von Alignments wird vor der Ausführung noch eine Algebra angegeben. Diese definiert, wie verschiedene Alignments bewertet werden.

Beim Berechnen eines Algorithmus werden drei Hauptschritte unterschieden [HzS2012]:

> “ADP on the other hand separates three concerns: the construction of the search space, evaluation of each candidate (or correct parse) available within this search space, and efficiency via tabulation of parts of the search space using annotation [13] of the grammar.” [HzS2012]

Zuerst muss der Suchraum für eine Eingabe erzeugt werden. Dies bedeutet konkret für den Fall von Wortvergleichen, es werden die Bäume aller möglichen Alignments zweier Wörter erstellt; im worst case exponentiell viele. Dies bewerkstelligt eine entsprechende Grammatik, auf welche später noch eingegangen wird. Innerhalb des Suchraumes werden nun einzelne Alignments mittels der Algebra ausgewertet. Da dieses in exponentieller Zeit läuft, bedient man sich „Bellman’s principle of Optimality“, welches das Grundprinzip der dynamischen Programmierung bildet:

> „An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with

> regard to the state resulting from the first decision."
> *– Bellman, 1957*

Im Falle eines Alignments bedeutet dies, dass die Entscheidung, welche zu Beginn getroffen wurde, auch die restlichen Schritte beeinflusst. Man kann also die Bäume einfacher auswerten, wenn die linken Teilbäume und damit auch gleiche Schritte am Anfang der Alignierung gleich sind und diese in einer Matrix gespeichert werden; ganz nach dem Prinzip der *memoization*. Hier verschmelzen das Aufstellen des Suchraumes und die Auswertung der Kandidaten eines Inputs, sodass jetzt nur noch Teilbäume betrachtet werden müssen, welche in dieser Konstellation noch nicht existieren und gleichzeitig den höchsten Wert durch die übergebene Algebra erzielen.

Dies bedeutet für den bereits genannten Needleman-Wunsch-Algorithmus, dass die Berechnungen für den Score eines Alignments der Matrix *M* wie folgt aussieht:

$$M(i,j) = \max \begin{cases} M(i-1,j) + gap\ penalty \\ M(i-1,j-1) + Score(w_1(i), w_2(j)) \\ M(i,j-1) + gap\ penalty \end{cases}$$

Für ein Alignment zweier Wörter w_1 und w_2 mit $|w_1| = n$, $|w_2| = m$, $i \leq n, j \leq m$, werden in der Tabelle die bisherigen Scores von links nach rechts und von oben nach unten eingetragen. Die *gap penalty* bezeichnet hier die Kosten, welchen man aufwenden muss, um ein *gap* einzufügen, also das Überspringen einer Stelle in einem Wort. $w_1(i)$ steht hier für den i-ten Buchstaben des Wortes w_1; analog für $w_2(j)$.

Die Grammatik, welche hier benötigt wird, muss also in der Lage sein, beide Wörter umzuschreiben. Dazu bedient man sich eines Konstrukts aus der theoretischen Informatik. Formal betrachtet ist eine Grammatik G ein 4-Tupel $G = (N, T, P, S)$, mit der Menge aus Nonterminalen *N*, den Terminalsymbolen *T*, den Produktionsregeln *P*, und einem Element *S* aus *N*, mit welchem die Produktionsregeln beginnen. Terminalsymbole $t \in T$ sind hierbei kurze Strings aus Unicode-Zeichen, welche hier aus Gründen der Einfachheit entweder als Buchstaben oder Character bezeichnet werden. In der Regel wählt man für die Nonterminale Groß- und für Terminale Kleinbuchstaben. Da die Symbole in der Menge der Terminale jedoch bereits durch die einzelnen Buchstaben der betrachteten Wörter vorgegeben sind, lässt sich diese Konvention nicht zur Gänze erfüllen. Aus Gründen der Vereinfachung wird c als Platzhalter für mögliche Buchstaben verwendet. Um nun ein Wort einlesen zu können, muss

eine Grammatik nichts mehr leisten als einen Buchstaben lesen zu können, oder ihn einmal zu überspringen, was im Hinblick auf das Alignment später wichtig wird.

Produkte von Grammatiken

Der folgende Ansatz ist Höner zu Siederdissen et al. [HzS] entnommen. Dort wird das formale Produkt linearer Grammatiken definiert. Für diese Arbeit ermöglicht es, die Beschreibung einer Grammatik für den eindimensionalen Fall. Die Grammatik S, die gesucht wird, sieht also wie folgt aus:

$$S = (N = \{X\}, \quad T = \{c\}, \quad P = \{X \to Xc \mid X\}, \quad X).$$

Da jedoch beide Wörter gleichzeitig umgeschrieben werden müssen, wendet man eine Multiplikation auf diese Grammatik an. Dazu bildet man das Kreuzprodukt aus den Produktionsregeln. Hier kann es jedoch passieren, dass eine Produktion das Alignment nicht verlängert. Außerdem muss ein Abbruchkriterium eingefügt werden, weshalb zwei weitere Grammatiken L und D eingeführt werden.

$$L = (N = \{X\}, \quad T = \emptyset, \quad P = \{X \to X\}, \quad X).$$

$$D = (N = \{X\}, \quad T = \{\varepsilon\}, \quad P = \{X \to \varepsilon\}, \quad X).$$

Da diese Grammatiken multipliziert mit sich selbst nach dem eigentlich Grammatikprodukt von S erst wichtig werden, wird ihr Produkt mit sich selbst benötigt [HzS]:

$$2 * L = \left(N = \left\{\binom{X}{X}\right\}, \quad T = \emptyset, \quad P = \left\{\binom{X}{X} \to \binom{X}{X}\right\}, \quad \binom{X}{X}\right).$$

$$2 * D = \left(N = \left\{\binom{X}{X}\right\}, \quad T = \left\{\binom{\varepsilon}{\varepsilon}\right\}, \quad P = \left\{\binom{X}{X} \to \binom{\varepsilon}{\varepsilon}\right\}, \quad \binom{X}{X}\right).$$

Somit ergibt sich für das Grammatikprodukt GP folgende Einschränkung:

$$GP = S \otimes S + 2 * D - 2 * L$$

Die hier verwendete Addition ist mengentheoretisch die Vereinigung, die Differenz bildet das Komplement.

Mittels all dieser Grundlagen lässt sich nun das eigentliche Grammatikprodukt für den Needleman-Wunsch-Algorithmus NW ermitteln:

$$NW = (N_{NW}, T_{NW}, P_{NW}, S_{NW}).$$

Wobei gilt:

$$N_{NW} = \left\{\binom{X}{X}\right\},$$

$$T_{NW} = \left\{\binom{c}{c}, \binom{c}{\varepsilon}, \binom{\varepsilon}{c}, \binom{\varepsilon}{\varepsilon}\right\},$$

$$P'_{NW} = \left\{\binom{X}{X} \to \binom{X}{X}\binom{c}{c} \mid \binom{X}{X}\binom{c}{\varepsilon} \mid \binom{X}{X}\binom{\varepsilon}{c} \mid \binom{X}{X}\right\} \cup \left\{\binom{\varepsilon}{\varepsilon}\right\} / \left\{\binom{X}{X}\right\}$$
$$= \left\{\binom{X}{X} \to \binom{X}{X}\binom{c}{c} \mid \binom{X}{X}\binom{c}{\varepsilon} \mid \binom{X}{X}\binom{\varepsilon}{c} \mid \binom{\varepsilon}{\varepsilon}\right\}$$

$$S_{NW} = \left\{\binom{X}{X}\right\}$$

Nur erhält man auch Ableitungen wie $\binom{c}{\varepsilon}$. Dies bedeutet, dass hier ein Buchstabe in w_2 übersprungen, also ein gap eingefügt, wird; analog für $\binom{\varepsilon}{c}$ in w_1. Deshalb muss P'_{NW} noch umgeschrieben werden zu:

$$P_{NW} = \left\{\binom{X}{X} \to \binom{X}{X}\binom{c}{c} \mid \binom{X}{X}\binom{c}{-} \mid \binom{X}{X}\binom{-}{c} \mid \binom{\varepsilon}{\varepsilon}\right\}$$

Dabei liefert nun c auch den Vorgänger – falls vorhanden – mit, der dann beim Bigrammvergleich wichtig wird. Um auch mögliche Anfangsbuchstaben und Endbuchstaben vergleichen zu können, wurden im Scoring Modell vor jedem Wort „^“ und am Ende eines jeden Wortes „$“ eingefügt.

Die Algebra schaut nun nach den erkannten Bigrammen in der Bewertungstabelle und berechnet für jeden Eintrag in der Tabelle den maximalen Wert aus, welcher sich in der Zelle der Bewertungstabelle finden lässt, welche sich „unten rechts“ befindet. Mittels Backtracking wird dann das optimale Alignment ermittelt und ausgegeben.

Kapitel 5

Auswertung

Zum Erstellen des Scoring-Modells wurden indoeuropäische Sprachen von der IDS-Datenbank (The Intercontinental Dictionary Series) eingelesen und wie in den Methoden beschrieben ausgewertet.

> „The purpose of the IDS is to establish a database where lexical material across the continents is organized in such a way that comparisons can be made." [IDS]

Die Daten sind derart aufbereitet, dass Sprachvergleiche recht leicht machbar sind. Zu diesem Zweck wurden für die enthaltenen Sprachen die Wörter mit ihrer Bedeutung eingetragen. Dies ermöglicht die Suche nach einer Bedeutung in allen Sprachen und der Vergleich der eingetragenen Wörter.

Die verwendeten Sprachen wurden mit Hilfe von Ethnologue.com direkt zu ihren Sprachuntergruppen, wie sie in Abbildung 2 zu sehen sind, eingeordnet:

Tabelle 5: verwendete Sprachen mit Zuordnung zu Sprachfamilien und Compilern. Großteil der Daten abrufbar auf http://lingweb.eva.mpg.de/ids/, hier jedoch aus privater Quelle, weshalb unveröffentlichte Daten enthalten sind.

Sprachfamilie	Sprache	Compiler
Slawisch	Bulgarian	Cynthia Vakareliyska, Kevork Horissian
	Czech	Václav Blažek
	Polish	Stanisław Puppel
	Russian	Jules F. Levin
	Sorbian_Lower	
Germanisch	Danisch	W. W. Schuhmacher
	Dutch	Bernard Bichakjian, Hendrikje Hettema
	English	various

	German_Standard	Meredith Lee
	Swedisch	Erik V. Gunnemark, Laila Kollmann
	Yiddish_Eastern	Paul Glasser
Romanisch	French	n.A.
	Catalan-Valencian-Balear	Joan Rafel (Universitat de Girona)
	Galician	
	Latin	Philip Baldi
	Italian	Martin Maiden, University of Oxford
	Portuguese	Various scholars
	Romanian	Maria Manoliu-Manea
	Sardinian_Logudorese	
	Spanisch	Joan Rafel (Iberian Spanish), Various scholars
	Venetian	
Armenisch	Armenian_Western	Dora Sakayan
Baltisch	Lithuanian	Ramutė Plioplys
	Latvian	Juris Cibuļs
Keltisch	Breton	Jean Le Dû
	Gaelic_Irish	
	Welsh	Andrew Hawke
Indisch	Hindi	
Albanisch	Albanian_Tosk	Leonard Newmark
Iranisch	Farsi_Western (Persian)	John R. Perry
	Judeo-Tat	
	Romani_Vlax	Donald Kenrick
Griechisch	Greek (Greek (Modern))	Maria Tsigou, Université René Descartes, Paris

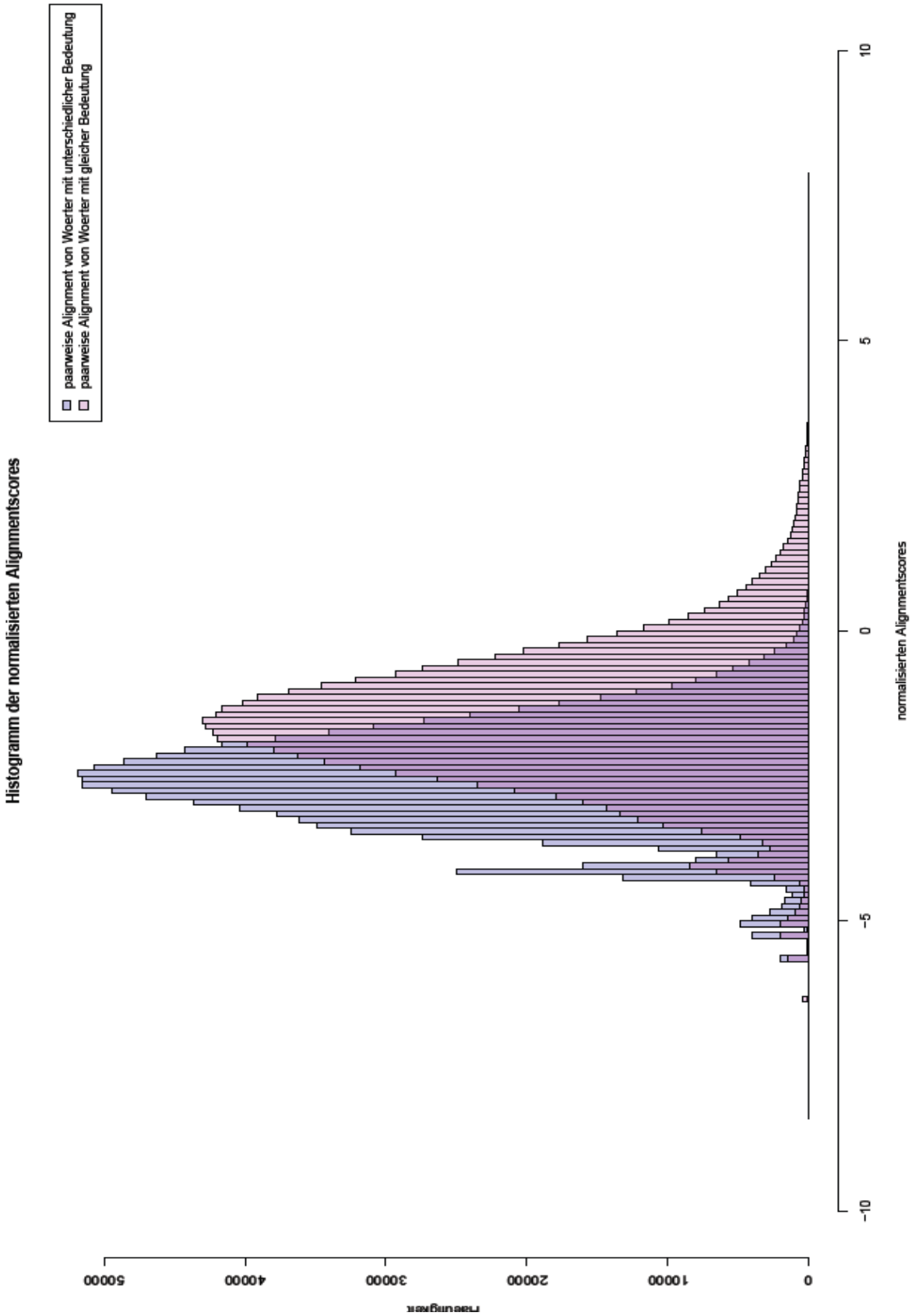

Abbildung 4: Histogramm der normalisierten Alignmenscores.

Für Latein muss an dieser Stelle erwähnt werden, dass dies eine ausgestorbene Sprache ist und diese zur romanischen Untergruppe hinzugezählt wurde, obwohl diese als Vorgänger dieser Untergruppe gilt.

Abbildung 4 zeigt die Ergebnisse der Scores, welche bei der Alignierung aller Daten entstand. Dabei wurde für Wörter mit unterschiedlicher und mit gleicher Bedeutung zwischen den Ähnlichkeitswerten unterschieden. Auf der x-Achse wurden die „normalisierten Alignmentscores“ aufgetragen. Normalisiert heißt, dass nach der Addition der einzelnen Scores für die Bigramme durch die Anzahl der Buchstaben im längsten Wort geteilt wird. Dabei zählen die angehangenen Zeichen ‚^‘ und ‚$‘ nicht zu der Wortlänge. Auf der y-Achse lässt sich dann zum Score die entsprechende Häufigkeit ablesen. Da sehr viele Alignments und damit auch Scores erzeugt wurden, was es leider nicht möglich alle in das Statistik-Tool R einzulesen. Deshalb wurden etwa gleich viele, zufällig ausgewählte Scores für die Alignment der Wörter mit verschiedener Bedeutung verwendet wie für die von gleicher Bedeutung zur Verfügung standen. Außerdem soll an dieser Stelle erwähnt werden, dass ein Buchstabe nicht zwingend für einen Buchstaben steht, wie man es erwarten würde, da diese Transkription nicht aussagekräftig genug wäre. Man kann sehen, dass Alignments von Wörtern mit gleicher Bedeutung in der Regel besser bewertet werden, als Alignments mit Wörtern untschiedlicher Bedeutung.

Im Kapitel „Linguistischer Hintergrund“ wurde bereits Grimm’s Law vorgestellt. Das Score-Modell wurde auch darauf untersucht. Auffällig hier sind Scores zum Beispiel für das englische ‚t h‘:

Tabelle 6: log-odds Score für Bigrammpaare aus Dänisch (Danish), Holländisch (Dutch), Englisch und Deutsch (German_Standard), welche mit englischem ‚t h‘ als ein Teil des Bigrammpaars

Sprache 1	Bigramm 1	Sprache 2	Bigramm 2	log-odds Score
Danish	^ t	English	t h	1.21336
	d n			2.50613
	d s			0.714372
	e d			1.28236
	r d			0.847903
Dutch	^ d	English	t h	1.16552
	a d			1.88399
	d e			1.29004
	d i			1.04036
	d o			0.727268
	d r			0.634895
	d u			1.04036

	e d			1.31344
English	t h	German_Standard	^ d	1.50233
			a d	0.955782
			d $	0.722167
			d a	1.41531
			d e	0.952691
			d i	0.642125
			d r	1.77199
			d u	1.50233
			e d	1.19217
			i d	1.37449
			r d	1.33527

Es fällt auf, dass ein Brigrammpaar sehr häufig einen hohen Score bekommt, wenn ‚t h' im Englischen mit einem ‚d' im Dänischen, Deutschen und Holländischen vorkommt. Hier hilft der Bigramm-Score, da man gezielt nach ‚t h' suchen kann. Für andere Lautverschiebungen ist dies nicht so einfach, da die zu betrachtenden Laute nur durch einen einzelnen Buchstaben kodiert werden. Ein Buchstabe als Teil des Bigramms lässt mehr Rauschen zu, wodurch diese Daten nur schwer durchsuchbar sind.

Wortalignments am Beispiel ‚Vater'

Anhand des Beispiels ‚Vater' in den Sprachen Dänisch, Niederländisch (Dutch), Englisch und Spanisch sollen hier noch Verwandtschaft von Sprachen und die schon beschriebenen Lautverschiebungen deutlich gemacht werden.

1. ID1: 10806 WORD1: ^ f a d e r $ LANGUAGE1: Danish
   ID2: 14041 WORD2: ^ f a t h e r $ LANGUAGE2: English
   SCORE: 5.20 NSCORE: 0.87

```
f a d - e r
f a t h e r
```

2. ID1: 10806 WORD1: ^ f a d e r $ LANGUAGE1: Danish
   ID2: 12355 WORD2: ^ v a d e r $ LANGUAGE2: Dutch
   SCORE: 11.02 NSCORE: 2.20

   ```
   f  a  d  e  r
   v  a  d  e  r
   ```

3. ID1: 12355 WORD1: ^ v a d e r $ LANGUAGE1: Dutch
   ID2: 14041 WORD2: ^ f a t h e r $ LANGUAGE2: English
   SCORE: 5.55 NSCORE: 0.92

   ```
   v  a  d  e  -  r
   f  a  t  h  e  r
   ```

4. ID1: 14041 WORD1: ^ f a t h e r $ LANGUAGE1: English
   ID2: 44671 WORD2: ^ p a d r e $ LANGUAGE2: Spanish
   SCORE: 0.21 NSCORE: 0.03

   ```
   f  a  t  h  e  r
   p  a  d  r  e  -
   ```

5. ID1: 12355 WORD1: ^ v a d e r $ LANGUAGE1: Dutch
   ID2:44671 WORD2: ^ p a d r e $ LANGUAGE2: Spanish
   SCORE: 4.66 NSCORE: 0.93

   ```
   v  a  d  e  r
   p  a  d  r  e
   ```

6. ID1: 10806 WORD1: ^ f a d e r $ LANGUAGE1: Danish
   ID2: 44671 WORD2: ^ p a d r e $ LANGUAGE2: Spanish
   SCORE: 3.27 NSCORE: 0.65

   ```
   f  a  d  e  r
   p  a  d  r  e
   ```

Man kann sehen, dass diese Alignments einen zumindest positiven Score trotz des Einfügens eines *gaps* bekommen. Exemplarisch wurde hierfür ein Phylogramm erstellt. In diesem werden durch den Neighbour-Joining-Algorithmus anhand der gegebenen Daten (Tabelle 7) die Verwandtschaftsbeziehungen berechnet und dann in einem Baum eingetragen:

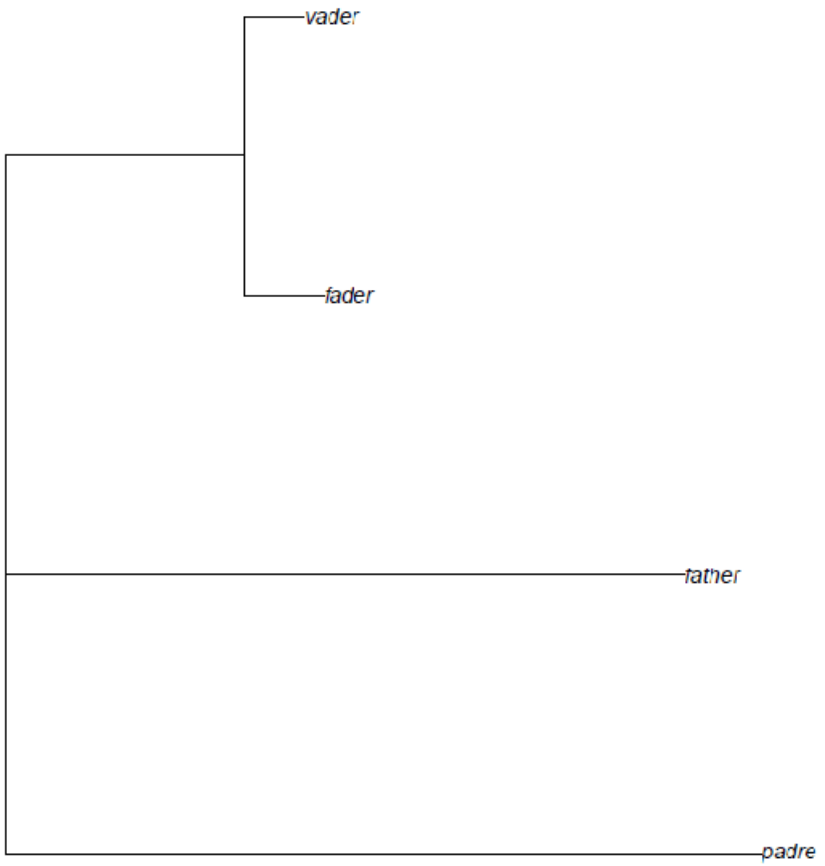

Abbildung 5: Phylogramm für Neighbour-Joining der Daten aus Tabelle7. Es ist deutlich ersichtlich, dass ‚fader' und ‚vader' zu sich ähnlicher sind als ‚father' und ‚padre' zu sich selbst oder zu der oberen Gruppe.

Tabelle 7: Berechnete similarity Scores für ‚Vater' in den Sprachen Danish, Englisch, Niederländisch und Spanisch, welche den erzeugten Alignments entnommen wurden.

	Fader	father	vader	padre
fader	3.13	0.87	2.2	0.65
father	0.87	3.08	0.92	0.03
vader	2.2	0.92	2.81	0.93
padre	0.65	0.03	0.93	2.94

Man kann sehen, dass ‚vader' und ‚fader' einen sehr hohen *similarity score* erhalten (Tabelle 7) und deshalb auch im Phylogramm als verwandt betrachten können. Im Vergleich dazu scheinen ‚padre' und ‚father' weder besonders verwandt zueinander noch zu ‚vader' und ‚fader' zu sein. ‚father' sollte jedoch recht nah verwandt zum Dänischen sein, was jedoch aufgrund der hier gewählten hohen Kosten für das Einfügen eines gaps nicht möglich ist.

Hier werden auch noch einige der schon im Linguistischen Hintergrund beschriebenen *sound shifts* deutlich:

a. Spanisch enthält noch einen Anlaut ‚p‘, welcher jedoch komplett ausgetauscht wird durch einen ‚f‘-Laut; also: $p \rightarrow f$.
b. Des weiteren kann man die Verschiebung von ‚d‘ hin zu ‚th‘ (Lautschrift ‚θ‘) sehen. Unter der Betrachtung des deutschen Wortes ‚Vater‘, kann man auch noch den Zwischenschritt, welcher bereits im Linguistischen Hintergrund genannt wurde erkennen. Es ergibt sich also eine Kette der Lautverschiebungen: $d \rightarrow t \rightarrow \theta$.

Leider enthielt der Datensatz das Wort für Vater nicht in Latein. Mittels einer phonetischen Transkription wäre hier vielleicht noch ein weiterer *sound shift* sichtbar geworden.

Schwächen der momentanen Alignments

Weiterhin ist bekannt, dass es bei der Entwicklung der spanischen Sprache zu einer Prothesis von ‚e‘ bei den Wörtern kam, welche mit ‚st‘ begannen. Die folgenden Beispiele wurden für das Sprachpaar Italienisch und Spanisch gewählt, da diese sehr nah miteinander verwandt sind:

7. Stern: „stella“ (ital.) – „estrella“ (span.)
   ID1: 25518 WORD1: ^ s t e l l a $ LANGUAGE1: Italian
   ID2: 44614 WORD2: ^ e s t r e l l a $ LANGUAGE2: Spanish
   SCORE: 0.88 NSCORE: 0.11

   ```
   s - t - e l l a
   e s t r e l l a
   ```

8. Stall: stalla (ital.) – establo (span.)
   ID1: 25650 WORD1: ^ s t a l l a $ LANGUAGE1: Italian
   ID2: 44756 WORD2: ^ e s t a b l o $ LANGUAGE2: Spanish
   SCORE: 0.32 NSCORE: 0.05

   ```
   s t a l l a -
   e s t a b l o
   ```

9. Statue: statua (ital.) – estatua (span.)

   ID1: 26275 WORD1: ^ s t a t u a $ LANGUAGE1: Italian

   ID2: 45485 WORD2: ^ e s t a t u a $ LANGUAGE2: Spanish

   SCORE: 9.82 NSCORE: 1.40

   ```
   s - t a t u a
   e s t a t u a
   ```

10. Dumm: stupido (ital.) – estúpido (span.)

    ID1: 26748 WORD1: ^ s t u p i d o $ LANGUAGE1: Italian

    ID2: 46044 WORD2: ^ e s t uÌ p i d o $ LANGUAGE2: Spanish

    SCORE: 2.87 NSCORE: 0.36

    ```
    s t u p - i d o
    e s t uÌ p i d o
    ```

Anhand dieser Alignierungen kann man sehen, dass das Modell hier fehlschlägt. Das Auszählen ergab für ‚s t' im Italienischen und ‚s t' im Spanischen einen sehr guten Wert. Jedoch erzielte zum einen das Bigrammpaar ‚s t' – ‚e s' für ‚s t' in Italienisch und ‚e s' in Spanisch und zum anderen das Bigrammpaar ‚^ s' – ‚^ e' für dieselben Sprachen ebenfalls einen hohen Wert. Bei der Auswertung der erstellten Tabelle während des Alignments war es also günstiger an einer Stelle ein gap einzufügen, wo die Score für diese Bigrammpaar nicht so hoch war. Dies sieht man ganz deutlich in Alignment 10: ‚ uÌ' (was für ‚ú' codiert) wurde nicht besonders häufig gesehen, weshalb die Scores hierfür sehr schlecht sein sollten. An dieser Stelle ist es für den gesamten Score günstiger das gap an dieser Stelle einzufügen, als den Anfang mit besseren Scores zu zerstören. Ein Lösungsvorschlag für dieses Problem soll in der Diskussion erfolgen.

Kapitel 6

Diskussion

Im vorangegangenen Kapitel wurde gezeigt, dass Alignments für Wörter mit gleicher Bedeutung in der Regel besser bewertet werden als Wortalignierungen mit verschiedener Wortbedeutung. Im Anschluss wurden Beispiele für Alignments mit gleicher Wortbedeutung herausgegriffen und ausgewertet. Dabei konnte man anhand der ‚Vater'-Alignierungen ein positives Beispiel sehen, wohingegen die Untersuchung der Prothesis von ‚e' im Spanischen zur Entwicklung einer komplexeren Grammatik führte, welche diese Besonderheiten besser modellieren kann.

Im Teil „Diskussion zu einzelnen Punkten" werden Gründe aufgezeigt, welche zu schlechteren Alignments geführt haben könnten. Im letzten Abschnitt „Erweiterungsmöglichkeiten" erfolgt ein Ausblick auf mögliche Projekte, die auf dieser Arbeit aufbauen könnten.

Diskussion zu einzelnen Punkten

gap-Scores

Die Wahl der gap-Scores erwies sich als eher ungeschickt, da Alignments entstanden, welche kaum zusammenpassten. Dies war der Fall, da der gap- und default-Wert so gewählt wurde, dass sich beim Alignieren ausschließlich an die Bewertungsmatrix gehalten wird. Das Einfügen von gaps oder Bigrammen, die noch nicht gesehen wurden, soll möglichst vermieden werden. Das stellte sich beim Erstellen eines Alignments mit unterschiedlichen Wortlängen als Hindernis heraus. Die Verwendung eines gaps war teuer. In Folge dessen werden zwei fast gleiche Wörter sehr wahrscheinlich einen schlechteren Wert bekommen, als beim Alignieren zweier Wörter, die sich weniger ähnlich sind. Der Score des Alignments würde als aussagen, dass zwei Wörter gleicher Länge sich immer ähnlicher sind, als möglicherweise verwandte Wörter mit unterschiedlicher Anzahl an Buchstaben.

Um andere gap-Scores zu testen, wurde das Alignment für einige ausgewählte Sprachen wiederholt. Dabei wurden drei weitere Varianten verwendet:

a. $g_{-1} = -1$
b. $g_{-2} = -2$
c. $g_{-3} = -3$

Dies lässt das Auftreten einer Insertion oder Deletion wesentlich häufiger zu. Das Problem hier, ist die Art der Trainingsdaten. Bei Verwendung von bereits alignierten Wörtern, kann der *gap*-Score direkt aus den Daten berechnet werden. Bei nicht alignierten Daten, wie die hier verwendeten, muss dieser geschätzt oder experimentell bestimmt werden. Letztes wurde mit den neuen *gap*-Scores versucht.

***default*-Wert**

Eine mögliche Folge der Wahl von einem sehr schlechten *default*-Wert könnten auch die Scores von Wörtern mit unterschiedlicher Bedeutung sein. Allerdings muss dieser Wert kleiner sein, als der schlechteste Wert der ausgezählten Bigrammpaare. Der Grund hierfür liegt in der Wahrscheinlichkeit mit der ein Bigrammpaar beim Berechnen eines Alignments auftritt. Wurde dieses Paar also zuvor nicht gesehen, so ist es auch sehr unwahrscheinlich, dass es hier vorkommt.

Scoring-Modell

Es wurden lediglich log-odds Scores von Bigrammpaaren berechnet, bei denen jedes einzelne Bigramm mindestens drei Mal in der betrachteten Sprache vorkommt. Da es jedoch immer noch sehr viel Rauschen gibt, könnte man versuchen den Wert für das minimale Vorkommen eines Bigramms in einer Sprache noch geschickter zu wählen.

Algorithmus

Der verwendete Needleman-Wunsch-Algorithmus ist ein globaler Algorithmus zur Berechnung von Alignments, welcher keine affinen *gap*-Modelle zulässt. Da gerade beim Vergleichen von Wörtern mit unterschiedlicher Länge nicht die Differenz der Länge als Hindernis gelten sollte, wäre ein affines *gap*-Modell ein alternativer Lösungsansatz.

Grammatik

Die verwendete Grammatik erweist sich problematisch, was das Einfügen von *gaps* betrifft; diese werden bisher „blind“ eingefügt. Das soll heißen, wenn es günstiger ist, ein *gap* einzufügen, anstelle der Betrachtung eines Bigramms, wird dieses ohne Betrachtung seines Kontextes getan. Eine im Rahmen dieser Arbeit von mir entwickelte Grammatik könnte diese Hürde überwinden und unterscheiden zwischen zuvor gesehenen Character, $gapopen$ oder $gapextend$. Hier sollte direkt ein affines *gap*-Modell möglich sein:

$$G'_{new} = (N'_{new}, T'_{new}, P'_{new}, S'_{new})$$
$$N'_{new} = \{S'_{new}, C, G\}$$
$$T'_{new} = \{c, , -\}$$
$$P'_{new} = \begin{Bmatrix} S'_{new} \rightarrow Cu \mid G-, \\ Cu \rightarrow Cuu \mid G-u \mid \varepsilon u, \\ G- \rightarrow Cu- \mid G- \mid \varepsilon-, \\ G \rightarrow Cu \mid G \mid \varepsilon \end{Bmatrix}$$

Die verwendeten Terminalsymbole sind c, für beliebige Buchstaben, $-$, für ein *gap open*, und , für *gap extension*. Bei den Produktionsregeln stehen die verwendeten Regeln genau für das Einfügen eines Characters (C) oder eines *gaps* (G).

Diese Grammatik ist zwar in dieser Form kontextsensitiv, jedoch fällt bei genauerem Hinsehen auf, dass hier auch eine Überführung in eine reguläre Form möglich ist. In der Regel ist das Überprüfen, ob zwei Grammatiken dieselbe Sprache beschreiben, ein NP-hartes Problem, wenn mindestens eine Sprache davon nicht regulär ist.

Die Produktionsregeln P'_{new} müssen wie folgt abgeändert werden:

$$P_{new} = \begin{Bmatrix} S_{new} \rightarrow C \mid O, \\ C \rightarrow Cu \mid Ou \mid \varepsilon u, \\ O \rightarrow C- \mid E- \mid \varepsilon-, \\ E \rightarrow C \mid E \mid \varepsilon \end{Bmatrix}$$

Damit ergibt sich also folgende lineare Grammatik G_{new}, welche noch für weitere Alignierungen verwendet wird:

$$G_{new} = (N_{new}, T_{new}, P_{new}, S_{new})$$
$$T_{new} = \{c, , -\}$$
$$N_{new} = \{S_{new}, C, O, E\}$$

Methoden

Im vorherigen Kapitel wurden Alignments gezeigt, welche die Prothesis von ‚e' am Beginn eines spanischen Wörtes beinhalten. Diese waren jedoch nicht zufriedenstellend, da am Anfang der Alignierungen falsche Paare entstanden. Dieses Problem könnte man lösen, indem man vor dem Auszählen der Bigrammpaare ein erstes Alignment durchführt. Hier sollen die Wörter zuerst in eine Transkription überführt werden, welche lediglich Vokale und Konsonanten betrachtet. Dies heißt explizit, dass jeder Vokal in einen Character v überführt wird; analog jeder Konsonant (consonant) in c. Leider kann es für ein paar Buchstaben vorkommen, dass diese nicht eindeutig zugeordnet werden können, weshalb hier noch ein weiterer Character o (für other) hinzugefügt wird.

Auch dieses Vorgehen birgt weitere Probleme. Hindi zum Beispiel ist in seiner orthographischen Transkription in der Datenbank enthalten. Dies macht es besonders schwer, da hier einige Zeichen Kombinationen aus Vokalen und Konsonanten sind. Das erfordert also zuerst das Spalten dieser Zeichen in eine Form, welche die Zuordnung zu den genannten Kategorien überhaupt ermöglicht.

Die meisten Probleme hier ließen sich umgehen, wenn die Daten in der bereits erwähnten IPA-Transkription vorhanden wären. Hier zeigt sich jedoch ein weiteres Problem. Für den empirischen Ansatz, die Bigrammpaare auszuzählen, werden sehr viele Daten benötigt um ein repräsentatives Ergebnis zu erzielen. Auf der anderen Seite steht das Problem, dass nur wenige Daten in IPA-Transkription vorhanden sind und man nur mit guter Sprachkenntnis Wörter in diese überführen kann.

ADPfusion ist bereits jetzt in der Lage für einen gewählten Wertebereich alle Alignments auszugeben, welche einen Score in diesem Intervall erzielt haben. Das Prinzip funktioniert ähnlich dem von Eppstein [Eppstein1998], dessen Ansatz auf der Graphentheorie beruht. Für die erzeugte Score-Tabelle bedeutet das insbesondere, dass jede Zelle im Graphen als Knoten repräsentiert ist. Der Score, welcher addiert wird, um während des Algorithmus von einer Zelle zur nächsten zu kommen, wird hier als Distanz zwischen den Knoten aufgetragen. Für die l besten Alignments lässt sich dieses Problem nun auf die l kürzesten Pfade in dem Graphen reduzieren.

Erweiterungsmöglichkeiten

Kognatendetektierung

Eine erste Möglichkeit einer Erweiterung wäre das Clustering von Wörtern. Hier können Kognaten mittels *fourway* Alignierung ermittelt werden. *Fourway* Alignierung bedeutet hier nichts weiter, als dass nicht mehr paarweise Alignments erzeugt werden, sondern Vergleiche von vier Wörtern auf einmal. Diese Option ist ebenfalls mit ADPfusion möglich, da hier lediglich ein weiterer Schritt der Grammatikproduktion, wie sie bereits vorgestellt wurde, durchgeführt werden muss. Aus paarweiser Alignierung wird also vierfache Alignierung.

Sprachverwandtschaften

Des Weiteren können Sprachverwandtschaften bestimmt werden. Diese ermittelt man aus dem durchschnittlichen Alignment-Score für die Alignierungen der Wörter beider Sprachen. Wie bereits für das ‚Vater'-Beispiel gezeigt, führt man daraufhin den Neighbour-Joining-Algorithmus für alle Sprachen aus. Im besten Falle kann man hier eine Übereinstimmung zu den in der Auswertung aufgeführten Subfamilien der indoeuropäischen Sprachfamilien finden. Dieses ließe sich auch noch einmal mittels *bootstrapping* überprüfen. Hier würden Wörter mit gleicher Bedeutung aller Sprachen untereinander aligniert werden und immer zufällig ein Wort ersetzt werden. Anschließen wird jedes dieser Alignments der Neighbour-Joining-Algorithmus ausgeführt und ein Phylogramm erstellt. Hat man dies oft genug durchgeführt, wird für alle erzeugten Bäume der stabilste Baum berechnet, wie es in der Masterarbeit von Aberer [Aberer2011] bereits vorgestellt wurde. Dieser Baum sollte dann mit großer Wahrscheinlichkeit das bestmögliche Phylogramm sein. Hier lassen sich auch Teilbäume herausfinden, welche nur eine geringe Wahrscheinlichkeit für ihre aktuelle Platzierung im Gesamtbaum haben. Dies lässt weitere Schlussfolgerungen über Sprachverwandtschaften zu.

K-gramme

Der in dieser Arbeit verwendete Ansatz betrachtet Bigramme. Die vorgestellten Alignierungsalgorithmen verwenden in der Regel Unigramme. Formal betrachtet, kann man dies allgemeiner schreiben als *k-gram*, wobei für gewöhnlich $k = 1$ verwendet wird. In dieser

Arbeit setzte man $k = 2$. Es wäre interessant zu sehen, ob sich die Scores für ein anderes *k* verbessern oder verschlechtern.

Für das Scoring-Modell wäre es ebenfalls von Vorteil, könnte man *k* beim Auszählen variabel halten. Für den hier aufgetretenen Fall, dass ‚t h' als Bigramm in Englisch mit dem Unigramm ‚d' in Dänisch, Holländisch und Deutsch korrelieren sollte, könnte man hier den in der Arbeit entdeckten Fehler durch $k = \{1, 2\}$ vermeiden. Es ist auch an größere Werte für k zu denken, deren obere Grenze jedoch erst experimentell bestimmt werden muss.

Sound shifts

Die in der Auswertung beschriebenen *sound shifts* wurden per Hand untersucht. Ein guter Ansatz zur Automatisierung wäre die Verwendung eines *Bayesschen Netzes*. Hier wird für beliebige Variablen überprüft, ob diese unabhängig voneinander sind. Sind sie es nicht, bilden diese einen Teilgraph im Netzwerk. Das bedeutet insbesondere, dass nur Knoten oder Variablen miteinander verbunden sind, welche in Abhängigkeit zueinander stehen. Dies könnte auch hilfreich bei der Detektion von *sound shifts* sein. Ein solcher Ansatz wurde bereits in Tübingen im Rahmen des EVOLAEMP-Projektes entwickelt und in LingPy [ListMoran2013] eingefügt.

Sprachfamilien

Sobald die Alignments für die indoeuropäische Sprachfamilie zufriedenstellend sind, ist vorgesehen, weitere Sprachfamilien zu untersuchen. Diese sind:

1. Mataco-Guaicuruan: enthält sieben Sprachen und wird in Teilen von Bolivien, Argentinien, Paraguay und Brasilien gesprochen
2. Panoan: diese Sprachefamilie hat ihren Standort in den nördlichen Anden. Es liegen hier vier Sprachen vor.
3. Tzesic: Die Daten dieser kaukasischen Sprachfamilie beinhalten ebenfalls sieben Sprachen.

Es wird davon ausgegangen, dass die Herangehensweise Sprachfamilien zu analysieren, überall dieselbe ist.

Kapitel 7

Bibliographie

Hausarbeit

[Alargov2007] Alargov, Luben. "Geschichte der Etymologie-Von der Antike bis ins 19. Jahrhundert." (2007).

Buch

[Arens1969] Arens, Hans. "2 1969 Sprachwissenschaft. Der Gang ihrer Entwicklung von der Antike bis zur Gegenwart." *Orbis Academicus.) Freiburg/München*.

[Campbell2004] Campbell, Lyle (2004). *Historical linguistics* (2nd ed. ed.). Cambridge: MIT Press.

Paper

[Bussotti2011] Bussotti, Giovanni, et al. "BlastR—fast and accurate database searches for non-coding RNAs." *Nucleic acids research* 39.16 (2011): 6886-6895.

[Eppstein1998] Eppstein, David. "Finding the k shortest paths." *SIAM Journal on computing* 28.2 (1998): 652-673.

[GiegerichMeyer2002] Giegerich, Robert, and Carsten Meyer. "Algebraic dynamic programming." Algebraic Methodology And Software Technology. Springer Berlin Heidelberg, 2002. 349-364.

[Gotoh1982] Gotoh, Osamu. "An improved algorithm for matching biological sequences." *Journal of molecular biology* 162.3 (1982): 705-708.

[Hamming1950] Hamming, Richard W. "Error detecting and error correcting codes." *Bell System technical journal* 29.2 (1950): 147-160.

[Henikoff1992] Henikoff, Steven, and Jorja G. Henikoff. "Amino acid substitution matrices from protein blocks." *Proceedings of the National Academy of Sciences* 89.22 (1992): 10915-10919.

[HzS2012] Höner zu Siederdissen, Christian. "Sneaking around concatMap: efficient combinators for dynamic programming." *Proceedings of the 17th ACM SIGPLAN international conference on Functional programming*. ACM, 2012.

[HzS] Höner zu Siederdissen, Christian, Ivo L. Hofacker, and Peter F. Stadler. "How to Multiply Dynamic Programming Algorithms." (accepted Paper)

[Kondrak2000] Kondrak, Grzegorz. "A new algorithm for the alignment of phonetic sequences." *Proceedings of the 1st North American chapter of the Association for Computational Linguistics conference*. Association for Computational Linguistics, 2000.

[Levenshtein1966] Levenshtein, Vladimir I. "Binary codes capable of correcting deletions, insertions and reversals." *Soviet physics doklady*. Vol. 10. 1966.

[ListMoran2013] List, J.-M. and S. Moran (2013): “An open source toolkit for quantitative historical linguistics”. In: Proceedings of the ACL 2013. System Demonstrations. (Sofia, Bulgaria, Aug. 4–9, 2013). Association for Computational Linguistics.

[NeedlemanWunsch1970] Needleman, Saul B., and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins." *Journal of molecular biology* 48.3 (1970): 443-453.

[SmithWaterman1981] Smith, Temple F., and Michael S. Waterman. "Comparison of biosequences." *Advances in Applied Mathematics* 2.4 (1981): 482-489.

[Steiner2011] Steiner, Lydia, Peter F. Stadler, and Michael Cysouw. "A pipeline for computational historical linguistics." *Language Dynamics and Change* 1.1 (2011): 89-127.

[WagnerFischer1974] Wagner, Robert A., and Michael J. Fischer. "The string-to-string correction problem." *Journal of the ACM (JACM)* 21.1 (1974): 168-173.

Master thesis

[Aberer2011] Aberer, Andre J. *Advanced Methods for Phylogenetic Post-Analysis*. Diss. Master’s thesis, TU/LMU Munich, 2011.

Dissertation

[List2012] List, Johann-Mattis. “Sequence Comparism in Historical Linguistics.” Düsseldorf (2012)

[Kondrak2002] Kondrak, Grzegorz. “Algorithms for Language Reconstruction.” Toronto (2002)

Webseiten

[Jacob2003] Jacob, Stefan. “Vom Indogermanischen zum Deutschen.” http://www.stefanjacob.de/Geschichte/Unterseiten/Sprachgeschichte.htm, 2003. (Zugriff: 6. August 2013)

[IDS] Key, Mary R., and Comrie, Bernard. “The Intercontinental Dictionary Series.” http://lingweb.eva.mpg.de/ids/ (Zugriff: 12.8.2013)

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Bachelorarbeit selbständig und nur mit den angegebenen Hilfsmitteln angefertigt habe und dass alle Stellen, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen sind, durch Angaben von Quellen als Entlehnung kenntlich gemacht worden sind. Diese Bachelorarbeit wurde in gleicher oder ähnlicher Form in keinem anderen Studiengang als Prüfungsleistung vorgelegt.

Ort, Datum Unterschrift