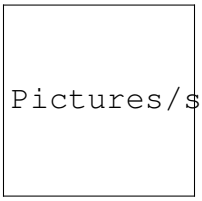
Diplomarbeit zur Erlangung des akademischen Grades
eines Diplom-Informatikers
Julius-Maximilians-Universität Würzburg



Alignment von mittelhochdeutschen und neuhochdeutschen Wörterbucheinträgen

Maria Haubner
geb. am 16. April 1986

Betreuer: Prof. Dr. Dietmar Seipel
Lehrstuhl für Informatik I
(Effiziente Algorithmen und wissensbasierte Systeme)
Institut für Informatik
Abgabedatum: 27. Juni 2013

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie die diesen Quellen und Hilfsmitteln wörtlich oder sinngemäß entnommenen Ausführungen als solche kenntlich gemacht habe. Die Arbeit habe ich bisher keiner anderen Prüfungsbehörde vorgelegt.

Würzburg, den 27. Juni 2013

Danksagung

An dieser Stelle möchte ich mich herzlich bei Prof. Dr. Dietmar Seipel bedanken. Nicht nur hat er mir eine interessante Aufgabe gestellt, in der ich das Wissen aus meinem Nebenfach Linguistik in meine Diplomarbeit einbringen konnte. Er hat mir auch in anregenden Besprechungen immer neue Blickwinkel auf das Thema gezeigt und mir vor Augen geführt, wie anspruchsvoll und lohnend es sein kann, interdisziplinär zu arbeiten. Ohne ihn und seine ausgezeichnete Betreuung der vergangenen Monate wäre diese Diplomarbeit nie zu Stande gekommen.

Weiterhin möchte ich mich bei vielen Freunden und Bekannten bedanken, die ich nicht namentlich aufzählen werde, die mich immer wieder gefragt haben, was ich da eigentlich mache. Ihnen zu erklären, worum es in meiner Diplomarbeit geht und an welchem Problem ich gerade knoble, hat oft dazu beigetragen, dass ich die Probleme lösen konnte und mit neuem Elan an die Arbeit ging.

Zu guter Letzt gilt mein Dank meiner Familie, die mich immer unterstützt und ohne deren Rückhalt ich nicht an diesen Punkt gekommen wäre.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Diese Arbeit befasst sich mit der Alignierung von mittelhochdeutschen und neuhochdeutschen Wörterbucheinträgen.

Da sich das Neuhochdeutsche aus dem Mittelhochdeutschen entwickelt hat, ist die Grundidee, dass man über Vergleiche versucht, jeweils die Wörterbucheinträge zu finden, die sich am meisten ähneln. Man kann dann davon ausgehen, dass der neuhochdeutsche Wörterbucheintrag die Entwicklung des mittelhochdeutschen Eintrags ist. Auf den auf diese Weise automatisch erhoben Daten kann man dann linguistische Analysen durchführen.

Alignierung bedeutet also in diesem Fall, dass man versucht, einem Wörterbucheintrag (einem sogenannten Lemma) aus den dem Mittelhochdeutschen ein Lemma aus dem Neuhochdeutschen zuzuordnen. Als Grundlage dieser Zuordnung werden für diese Arbeit das *Mittelhochdeutsche Handwörterbuch* von Matthias Lexer und das *Deutsche Wörterbuch* der Brüder Grimm verwendet.

Beide Wörterbücher spiegeln nur ansatzweise den Wortumfang der damals gesprochenen Sprache wider. Der Lexer umfasst ca. 80.000 Einträge, während der Grimm ca. 300.000 Einträge verzeichnet. Dementsprechend finden sich Lemmata in einem der Wörterbücher, die im anderen nicht auftauchen und für die sich somit keine Zuordnung finden lässt.

Auch die Annahme, dass alle Einträge des Lexers in dem wesentlich umfassenderen Wörterbuch der Brüder Grimm eine Entsprechung haben sollten, bestätigt sich nicht. Der Lexer belegt alle seine Einträge mit historischen Quellen, so dass nur Wörter aufgenommen wurden, die belegbar waren. Dies ergibt einen sehr zufälligen und teilweise speziellen Wortschatz. So kommt es vor, dass im Lexer, obwohl er weniger umfassend ist, Wörterbucheinträge auftauchen, die keine Entsprechung im Grimm haben. Auch gibt es Wörter die der Sprachentwicklung zum Opfer fielen, und somit im Neuhochdeutschen nicht mehr genutzt wurden.

Die scheinbar willkürliche Auswahl der Lemmata ist eins der Hauptprobleme der Alignierung zweier historischer Wörterbücher. Doch selbst wenn

man weiß, dass es eine neuhochdeutsche Entsprechung für eines der mittelhochdeutschen Wörter gibt, birgt die Aufgabe der Alignierung einige Schwierigkeiten.

Prof. Dr. Dietmar Seipel hat im Zuge des kooperativen Projekts *Wechselwirkungen zwischen linguistischen und bioinformatischen Verfahren, Methoden und Algorithmen: Modellierung und Abbildung von Varianz in Sprache und Genomen* das dieser Arbeit zu Grunde liegende Alignment-Tool entwickelt. Die Hauptaufgabe dieser Arbeit ist es, vorgegebene und eigene Verbesserungen zu realisieren.

Kapitel 2 stellt einige Grundlagen dieser Arbeit vor. Dies sind zum einen die Wörterbücher, die für das Tool genutzt werden, und zum anderen wird kurz das Verbundprojekt vorgestellt, im Zuge dessen das Alignmenttool entstanden ist.

Kapitel 3 stellt die grundsätzliche Arbeitsweise des Alignmenttools vor. In Kapitel 4 werden die Ideen zur Verbesserung des Tools vorgestellt und in Kapitel 5 folgt deren Umsetzung und die Auswertung der Testergebnisse. Danach folgen in den Kapiteln 6 und 7 weitere Überlegungen dazu, wie man das Tool verbessern kann.

Abschließen folgt der produzierte Code und das Literaturverzeichnis.

Die Arbeit wird zeigen, dass sich hauptsächlich durch die Weiterführung der sprachwissenschaftlichen Aspekte des Tools sowohl die Ergebnisanzahl von gefundenen Alignments, als auch deren Wahrscheinlichkeit erhöhen lässt. Weiterhin wird sich zeigen, dass sich die Ergebnisse hauptsächlich durch Anpassung und Erweiterung des originalen Quellcodes bewirken ließen. Aufgabe war es demnach nicht, etwas komplett neues zu programmieren, sondern mit dem Originalprogramm als Basis und Bezugspunkt weiterzuarbeiten. Wie dies umgesetzt wurde, soll im Folgenden vorgestellt werden.

Kapitel 2

Grundlagen

2.1 Wörterbücher

Der Datenbestand, mit dem das Alignment-Tool arbeitet, setzt sich aus den Lemmata zweier historischer Wörterbücher zusammen. Da sich diese Wörterbücher von heutigen Nachschlagewerken, wie z.B. dem Duden, unterscheiden, werden sie im Folgenden kurz vorgestellt.

2.1.1 „Das Mittelhochdeutsche Handwörterbuch“ von Matthias Lexer

„Das Mittelhochdeutsche Handwörterbuch“ [1], im Folgenden „der Lexer“ genannt, dient bei dieser Alignierungsaufgabe als Grundlage des mittelhochdeutschen Wortschatzes. Mit der Erstellung dieses Werkes begann Dr. Matthias Lexer im Jahr 1868 und arbeitete an verschiedenen Auflagen (das *Mittelhochdeutsche Handwörterbuch* und das *Mittelhochdeutsche Taschenwörterbuch*) bis 1885. Das Wörterbuch sollte einen alphabetisierten Index zum Wortfamilienwörterbuch *„Das Mittelhochdeutsche Wörterbuch“* („BMZ“ genannt) von G. F. Benecke, W. Müller und F. Zarncke (4 Bände. Leipzig 1854 – 66) darstellen und dieses um neue Einträge aus belegbaren Quellen erweitern. Auf der Seite des MWV (Mittelhochdeutsche Wörterbücher im Verbund)[1] der Uni Trier heißt es:

> „[...] Auf diese Weise verzeichnet der Lexer ca. 34.000 neue Stichwörter. Als Index bleibt das Handwörterbuch allerdings ganz eng auf den BMZ bezogen; Artikel zu solchen Stichwörtern, die auch im BMZ belegt sind, müssen stets mit den entsprechenden BMZ-Artikeln zusammen gelesen werden. “

[1] `http://germazope.uni-trier.de/Projekte/MWV/wbb`

Lexers Wörterbuch wurde mit seinen ca. 80.000 Einträgen zu einem der wichtigsten Hilfsmittel beim Studium des Mittelhochdeutschen, da es die einzelnen Einträge (sogenannte Lemmata) mit historischen Quellen belegt.

Zwischen 1881 und 1889 führte Matthias Lexer auch die Arbeit am Wörterbuch der Brüder Grimm fort.

2.1.2 „Das Deutsche Wörterbuch“ der Brüder Grimm

Das *Deutsche Wörterbuch* (DWB) [3] ist das größte und umfassendste Wörterbuch zur deutschen Sprache seit dem 16. Jahrhundert und dient bei dieser Aufgabe somit als Grundlage des neuhochdeutschen Wortschatzes. Begonnen haben es 1838 die Brüder Jacob und Wilhelm Grimm, weswegen es auch „der Grimm“ genannt wird.

Auch das DWB belegt die Herkunft und den Gebrauch der Einträge anhand von historischen Quellen. Der letzte Eintrag, den einer der Brüder Grimm hinzufügte, war der der 'Frucht'. Danach wurde die Arbeit von anderen Sprachwissenschaftlern fortgeführt.

Beendet wurde „Das Deutsche Wörterbuch“ 1961 mit insgesamt 32 erschienen Bänden und einem 1971 erschienen Quellenband. Es umfasst annähernd 300.000 Einträge, rund 4.000 Quellen wurden ausgewertet und viele Zitate verzeichnet.

Veranschaulichung der Wörterbücher

Die Abbildungen 2.1 und 2.2 sollen eine Vorstellung vermitteln, wie die beiden historischen Wörterbücher Lexer und Grimm (DWB) aussehen. Die Seite aus dem Lexer, 1. Auflage 1871, ist Eigentum der Bayrischen Staatsbibliothek[2] und kann dort eingesehen werden.

Die Seite aus dem Grimm ist ein Scan der im Literaturverzeichnis angegebenen dtv-Ausgabe von 1984, welche zum Bibliotheksbestand der Universität Würzburg gehört.

Abbildung 2.1 zeigt die Einträge *vlühtec-lîche, -en* bis *vluor*. Dieser Ausschnitt wurde gwählt, da man so den Eintrag *vluoch* mit seinem neuhochdeutschen Äquivalent *Fluch* im Grimm vergleichen kann.

Betrachten wir die Einträge im Lexer also am Beispiel 2.1.

Wie man sieht beginnt jeder Eintrag mit einem Lemma (*vluoch*). Danach folgt die Wortklasse (*stm.* - Substantiv, starke Deklination, männlich) und die Quellenangabe (meistens, wie auch in diesem Fall, ein Verweis auf das BMZ).

[2] `http://www.bsb-muenchen.de/`

vluoch - *stm.* (BMZ ib.) *verwünschung, -fluchung, fluch* DIEM. A.HEINR. WALTH. MS. (herre got, gip dem verschamten man wîbe vluoch *H.* 3, 81^{b}. do er wolte uns die argen flüeche vertrîben *ib.* 354^{b}. tôre, kum dîns fluoches abe MSF. 85,21. deist ein swinder vl. NEIDH. 100,8. mîne flüeche sint niht smal *ib.* 39, 37). BARL. ANEG. STRICK. (zeim vluoche werde im sîn gebet KARL 2909). da? ist ein vl. J.TIT. 5953. ein fluoch, ein slac den êren HADAM. 724. der fluoch stê ouf mir GEN. *D.* 49,6. der vl. muo? uber dich ergân *ib.* 19,5. nû ist der gotes vl. an uns ergân LIVL. *M.* 1140. der ir dise herzeleit tete mit flûche oder mit gebete HERB. 13436. des vertânen fl. SILV. 2845. er bôt vil manigen vl. geswinde, da? TROJ. 8891. der in zuo dem vl. spuon ROTH *dicht.* 4,95. sus begunde si dâ ir vluoch ander weide bannen GA. 3. 78,1316. dem künige wart dâ manic vl. gegeben ULR. *Wh.* 191^{a}. einem einen vl. tuon *ib.* 163^{c}. HELBL. 8,314. solten alle flüeche kleben, sô müesten lützel liute leben FREID. 130,12. — *mit gt.* flêkan *zu lat.* plango DWB.3,1829. *vgl.* CURT.3 261;

Beispiel 2.1: Der Wörterbucheintrag *vluoch* im Lexer

An diesem Punkt hört die Regelmäßigkeit in den Einträgen jedeoch auf. Nach der Quellenangabe des Lemmas folgen in diesem Fall neuhochdeutsche Übersetzungen, manchmal folgen aber auch Übersetzungen in andere Sprachen (z.B. Latein) oder auch keine Übersetzung. Danach folgen Zitate, in denen das Lemma auftaucht, mit Quellenangaben.
In manchen Fällen gibt es aber weder Übersetzung noch Zitate, sondern nur Verweise auf ein anderes Lemma. Dann ist man meist auf die Dialektform eines Hauptlemmas gestoßen.

Auch ein Wörterbucheintrag des Grimms, wie in Beispiel 2.2 das Lemma *Fluch*, soll näher betrachtet werden.

Man sieht, dass auch dieser Eintrag mit dem Lemma beginnt. Danach folgt bei Substantiven der Marker für 'masculinum' (m.) oder 'femininum' (f.). Im Anschluss daran folgt in diesem Beispiel eine lateinische Übersetzung sowie einige in andere mit dem Neuhochdeutschen verwandte Sprachen. Diese Übersetungen sind allerdings nicht in jedem Eintrag des Wörterbuchs vorhanden. Sind sie vorhanden, so sind sie gekennzeichnet, wie beispielsweise bei der Übersetzung *vluoch*, der ein 'mhd.' vorausgeht (Ausnahme: die lateinischen Übersetzungen). Danach folgen, wie auch schon im Lexer, Worterklärungen, Zitate und ähnliches mit Quellenangaben.

left=25mm, right=25mm, top=20mm, bottom=30mm

FLUCH, *m. exsecratio, imprecatio, maledictum, ahd.* fluoch, *mhd.* vluoch, *alts.* fluoc, *nd.* flok, *nnl.* vloek. *mangelt goth. ags. engl. altn. schw. dän., worüber mehr beim verbum. man sagt* ein schwerer, harter, bitterer, tiefer, herzlicher fluch *und dem* fluch, *der verwünschung steht der* wunsch, *der* segen, *dem* verwünschen, *devovere das* wünschen, *vovere gegenüber*, ein heiles wunsch, *wie bei* WALTHER 18, 25—28 *schlieszt dasselbe ein, was der* fluchende wunsch *im LS.* 2, 423 *ff. abspricht. merkwürdig ist* FRAUENLOBS *ausdruck* 'dës vluoches herter kisel' *MSH.* 2, 339^{b}. *Ettm. s. 5 und gleichsam anklingend an* fluoh *rupes, dessen* h *sich doch vom* ch *in* fluch *abkehrt. in einer andern stelle* FRAUENLOBS *Ettm. s.* 221 *heiszt es* 'von vluoches slihte'. flüche steigen auf in die höhe, fliegen (*sp.* 1784, 7), schweben und senken sich nieder (*mythol. s.* 1177), entschlüpfen, entwischen, folgen, verfolgen, sie kleben und ruhen auf einem. [...]

Beispiel 2.2: Der Wörterbucheintrag *Fluch* imd Grimm

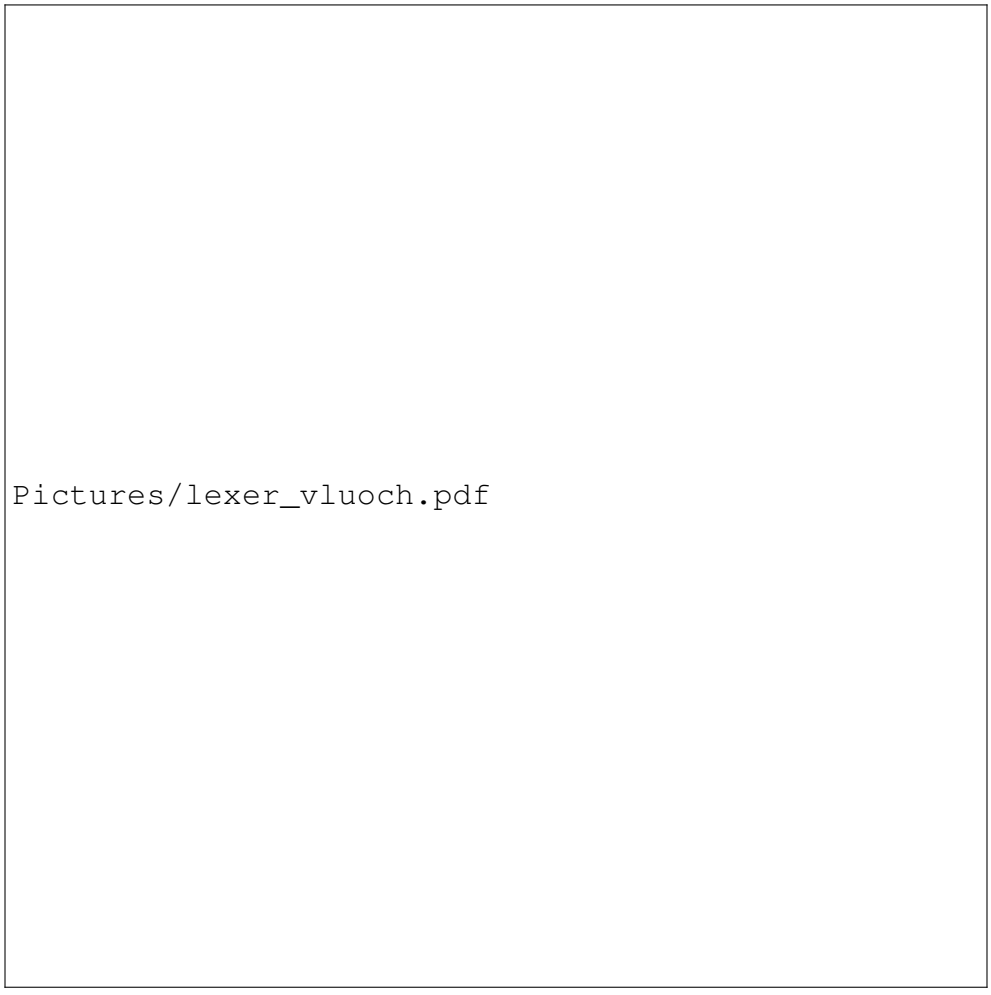


Abbildung 2.1: Der Eintrag *vluoch* im Lexer [2]

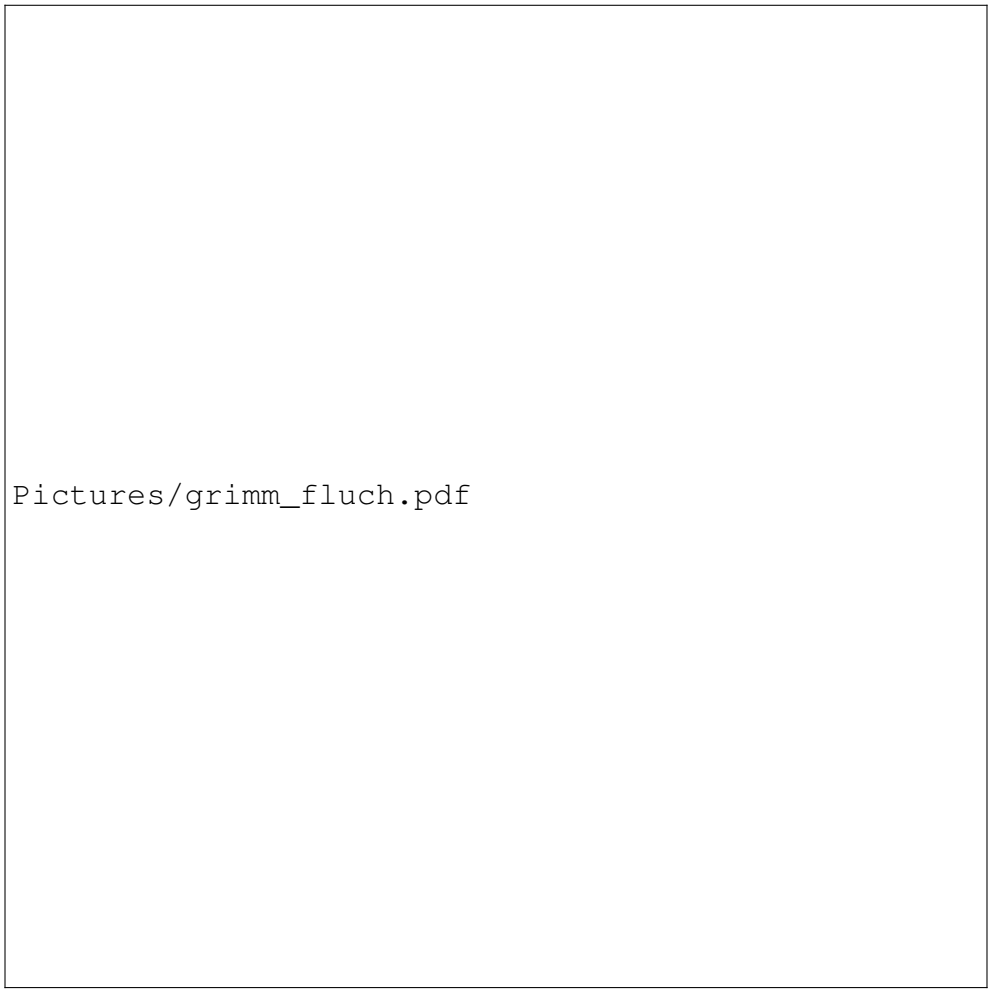


Abbildung 2.2: Der Eintrag *Fluch* im Grimm

2.2 Über das Projekt „Sprache und Genom“

Das Projekt *„Wechselwirkungen zwischen linguistischen und bioinformatischen Verfahren, Methoden und Algorithmen: Modellierung und Abbildung von Varianz in Sprache und Genomen“, oder kurz „Sprache und Genom“* [4], ist eine kooperative Arbeit zwischen dem Institut für Sprach- und Literaturwissenschaft (Technische Universität Darmstadt), dem Institut für deutsche Philologie, Computerphilologie (Universität Würzburg), dem Lehrstuhl für Informatik I „Effiziente Algorithmen und wissensbasierte Systeme“ (Universität Würzburg), dem Biozentrum (Universität Würzburg), dem Institut für Deutsche Sprache Mannheim (IDS), sowie bis Mai 2011 dem Kompetenzzentrum für elektronische Erschließungs- und Publikationsverfahren. Im Zeitraum von 2009 bis 2013 wurde dieses Verbundprojekt durch das Bundesministerium für Bildung und Forschung gefördert. Die Projektpartner bemühten sich im Zeitraum von vier Jahren die deutsche Sprache aus einem neuen Blickwinkel aus zu betrachten.

> Information und Wissen können durch Gruppierung von Einzelzeichen nach bestimmten Regeln kodiert werden, dabei gibt es prinzipielle strukturelle Gemeinsamkeiten zwischen Genomcode und sprachlichem Code sowie biologischer Entwicklung und Sprachentwicklung [...]. Ein zentrales Kennzeichen neben dieser „Entwicklungsfähigkeit“ und damit „Historizität“ ist die Vielfalt bzw. Varianz der Erscheinungen. Ein differenzierteres Verständnis der Mechanismen und Regeln von Entwicklung und Varianz ermöglicht neue und präzisere Verfahren der Informationsgewinnung sowie der Speicherung, Bearbeitung und Auswertung der dadurch gewonnenen Daten.
>
> (Von der Projekthomepage `http://www.sprache-und-genome.de/`)

Der Grundgedanke ist also, dass Sprache sich entwickelt - so ähnlich, wie es auch Genome in der Biologie tun, und eine Evolution durchläuft. Da in der Bioinformatik schon informatische Konzepte zur Analyse genutzt werden, wurden nun durch Prof. Seipel neue Methoden und Tools auf der Grundlage dieser Konzepte entwickelt, die in der Sprachwissenschaft zur Analyse angewendet werden können. So könnte man auch in der Linguistik Vermutungen und Untersuchungen zur Varianz und Entwicklung der Sprache anhand empirisch erhobener Daten belegen oder widerlegen. Da es das Ziel ist, die Daten automatisch generieren zu lassen, würden die Datenbestände wesentlich größer und auch, bei sorgfältiger Handhabung, wesentlich leichter auszuwerten, als es bei händisch gesammelten Daten der Fall sein kann.

Kapitel 3

Arbeitsweise des Alignment-Tools

Das Alignment-Tool wurde im Zuge des Projekts „Sprache und Genom“ von Prof. Dr. Dietmar Seipel entwickelt. Seine Arbeitsweise soll in diesem Kapitel vorgestellt werden.

3.1 Allgemeines

Die Alignierung eines Wortpaares zeigt an, wie ähnlich sich die beiden Wörter sind. Da sich das Neuhochdeutsche aus dem Mittelhochdeutschen entwickelt hat, berechnet die Alignierung also die Wahrscheinlichkeit, mit der sich das neuhochdeutsche Wort aus dem mittelhochdeutschen Wort entwickelt hat.

Diese Wahrscheinlichkeit wird folgendermaßen ermittelt: Die Grundlage bildet eine Lautverschiebungstabelle (die sogenannte Substitutionstabelle), in der die möglichen Entwicklungen, bzw. Verschiebungen, von Buchstaben und Buchstabengruppen mit zugehörigen abgeschätzten Wahrscheinlichkeiten vermerkt ist. Bei dem Vergleich eines mittelhochdeutschen mit einem neuhochdeutschen Wortes vergleicht man also die einzelnen Buchstaben miteinander. Multipliziert man dann die Wahrscheinlichkeiten der einzelnen Laut- bzw. Buchstabenverschiebungen, so erhält man die Wahrscheinlichkeit für das Alignment des ganzen Wortes.

Die Abbildung 3.1 stellt den groben Aufbau der einzelnen Komponenten dar, die für das Alignment nötig sind.

Der Basiskorpus ist ein wichtiger Grundbaustein des Alignment-Tools, welcher von Luise Borek (TU Darmstadt) erstellt wurde. Dieser besteht aus einer Liste von Wortpaaren, die sich jeweils aus einem mittelhochdeutschen Wort und seiner neuhochdeutschen Entwicklung zusammensetzten. Aus diesem Basiskorpus wird die Substitutionstabelle gebildet, die eine der vier Säu-

Abbildung 3.1: Grober Aufbau des Alignment-Tools

len des Alignment-Tools darstellt. Die anderen drei sind die Daten der Wörterbücher, also des Lexers und des DWBs, und natürlich das Alignmentprogramm an sich.
In der Substitutionstabelle sind die Übergangswahrscheinlichkeiten der einzelnen Zeichen (Buchstaben und Buchstabengruppen) gespeichert. Während das Alignmentprogramm also die Lemmata aus dem Lexer und dem DWB vergleicht um Alignments zu finden, konsultiert es für jeden einzelnen Buchstaben der zu vergleichenden Wörter die Substitutionstabelle, um die Wahrscheinlichkeiten zu ermitteln.
Ist ein Alignment gefunden, so wird es in die Alignmentdatenbank gespeichert.

Für die folgende Arbeit ist es wichtig, zwischen „Alignment-Tool“, welches alle oben genannten Komponenten enthält, und „Alignment-Programm“, welches die Implementierung des Alignierungsvorgangs zweier Lemmata ist

(also das Programm an sich), zu unterscheiden.

Der Basiskorpus, die Substitutionstabelle und die Alignmentmethoden sollen nun noch einmal genauer betrachtet werden.

3.2 Basiskorpus

Als Basiskorpus wird in diesem Fall eine Liste aus 1717 Wortpaaren bezeichnet. Jeweils ein mittelhochdeutsches Wort wurde mit seiner neuhochdeutschen Entsprechung gepaart, um so die Wahrscheinlichkeiten der Lautentwicklung erheben zu können.

```
so[ll]i[ch];sol%[ch]
sol[ch];sol[ch]
soli[ch];sol%[ch]
solh;sol[ch]
söl[ch];sol[ch]
sölh;sol[ch]
sp[uo]le;spule
sp[uo]l%;spule
brët;bre[tt]
zol;zo[ll]
vr[ei]sic;fr[ei]sig
vr[ei]sec;fr[ei]sig
blanc;blank
tœtigen;töt%%en
dürniz;durni[tz]
dürni[tz]e;durni[tz]%
dorni[tz]e;durni[tz]%
dürniz;turni[tz]
dürni[tz]e;turni[tz]%
dorni[tz]e;turni[tz]%
swindeln;[sch]windeln
l[ie]tlîn;l[ie]dl[ei]n
l[ie]delîn;l[ie]d%l[ei]n
ungel[ou]pli[ch];ung%l[au]bli[ch]
i[nn]ewendic;in%wendig
vogel;vogel
hunt;hund
sta[ch]el;sta[ch]el
gel[iu]te;gel[äu]te
...
```

Wichtig ist hierbei, dass sich das neuhochdeutsche Wort aus seinem mittelhochdeutschen Partner entwickelt haben muss. So wurde zum Beispiel mhd. *magentuon* mit nhd. *magdthum*, anstatt dem heute geläufigeren *jungfräulichkeit*, gepaart. An dieser Stelle sieht man sehr gut, dass eher die Entwicklung der Sprache automatisch untersucht werden soll, anstatt eine Übersetzung im klassischen Sinn zu finden. Weiterhin erkennt man, dass sich die Rechtschreibung der Grimm'schen Zeit teilweise von der heutigen unterscheidet (*thum* vs. *tum*), obwohl der Grimm ein neuhochdeutsches Wörterbuch ist.

Die Wortpaare wurden nun unter Beachtung linguistischer Erkenntnisse des Sprachwandels bearbeitet, damit sie gleich lang sind. Hierzu wurden teilweise Buchstaben zu einem Zeichen zusammengefasst (z.B. *[ou]*), wenn diese sich beispielsweise zu Einzelbuchstaben entwickelten. So soll sicher gestellt werden, dass das Programm relativ leicht erkennen kann, ob sich ein Zeichen (also ein Buchstabe oder eine Buchstabengruppe) ändert und wenn ja, in welches. Im oben genannten Wortpaar mhd. *magentuon* und nhd. *magdthum* erkennt man beispielsweise, dass mhd. *en* verschwindet und mhd. *uo* zu nhd. *u* wird. Letzteres ist ein Beispiel für die neuhochdeutsche Monophthongierung. Weitere Prozesse des Sprachwandels sind beispielsweise die neuhochdeutsche Diphthongisierung (z.B. mhd. *ü*, geschrieben als *iu*, wird zu nhd. *äu* oder *eu*), die Stärkung des Silbenrandes (frnhd. Konsonantenepethenese - mhd. *nieman* wird zu nhd. *niemand*), die Entwicklung der Fugenelemente (*Freund-**es**-kreis*) und andere [7]. Wichtig ist es vor allem, all jene Änderungen zu beachten, bei der Buchstaben im Wort verschwinden oder hinzu kommen.

Nach der Bearbeitung sieht ein Wortpaar dann beispielsweise so aus:
magent[uo]n ; mag%d[th]um. Das %-Zeichen im neuhochdeutschen Partner zeigt dabei an, dass hier ein Zeichen verschwunden ist.

Mit Hilfe des so bearbeiteten und vorbereiteten Basiskorpus kann nun die Substitutionstabelle erstellt werden.

3.3 Substitutionstabelle

Die Substitutionstabelle ist ein Bestandteil der Datenbank, mit der das Alignmentprogramm arbeitet.

Für jeden Eintrag stehen folgende vier Daten zu Verfügung: eine Identifikationsnummer ('NR'), das mittelhochdeutsche Zeichen ('SgnA'), das dazugehörige neuhochdeutsche Zeichen ('SgnB') und die relative Häufigkeit des Übergangs ('Freq'), bzw. die Wahrscheinlichkeit, mit der der Übergang zwischen den Lauten stattfindet. Lesen kann man diese Daten wie im Beispiel 3.1 für das mittelhochdeutsche Zeichen *d* gezeigt wird.

Wie schon angedeutet wird die Substitutionstabelle aus dem Basiskorpus erzeugt. Um dies zu erreichen, werden die Wortpaare in ihre einzelnen Zei-

NR	SgnA	SgnB	Freq
...			
1164	'd'	'%'	0.0088888888888888890
1165	'd'	'd'	0.8666666666666667000
1166	'd'	'dd'	0.0088888888888888890
1167	'd'	't'	0.1111111111111111000
1168	'd'	'tt'	0.0044444444444444444
1169	'dw'	'dw'	0.5000000000000000000
1170	'dw'	'qu'	0.5000000000000000000
1171	'e'	'%'	0.1952406706327744800
1171	'e'	'a'	0.0135208220659816100
1171	'e'	'ah'	0.0010816657652785290
1171	'e'	'e'	0.7301243915630070000
1171	'e'	'ee'	0.0032449972958355870
1171	'e'	'eh'	0.0037858301784748512
1171	'e'	'ei'	0.0005408328826392645
1171	'e'	'i'	0.0221741481882098450
1179	'e'	'ie'	0.0010816657652785290
1180	'e'	'o'	0.0010816657652785290
1181	'e'	'ä'	0.0227149810708491070
1182	'e'	'äh'	0.0027041644131963224
1183	'e'	'ö'	0.0021633315305570580
1184	'e'	'ü'	0.0005408328826392645
...			

Tabelle 3.1: Auszug aus der Substitutionstabelle

chen aufgespaltet und deren Übergangswahrscheinlichkeiten bestimmt. Anhand von dem Beispiel des mittelhochdeutschen Zeichens *d* soll das nun erläutert werden.

Zunächst wird gezählt, wie oft das Zeichen *d* in allen mittelhochdeutschen Wörtern des Basiskorpus auftaucht. Dies sind genau 234 Vorkommen. Da alle Wortpaare bearbeitet wurden, so dass sie gleich lang sind, weiß man nun, in welches Zeichen sich *d* gewandelt hat, da dieses Zeichen im neuhochdeutschen Wort an der selben Stelle im Wort stehen muss. So erfährt man dann zum Beispiel, dass *d* zweimal verschwindet und 196 mal unverändert bestehen bleibt. Mit diesem Wissen kann man dann sehr einfach berechnen, wie häufig sich das mittelhochdeutsche Zeichen in das Neuhochdeutsche entwickelt. Dies zeigt an, wie ähnlich sich diese Zeichen sind, bzw. wie wahrscheinlich dieser Übergang statt findet.

Die Wahrscheinlichkeiten werden für jedes einzelne Zeichen berechnet

Das mhd. Zeichen *d* wird im Nhd. zu:

1. *d* mit einer Wahrscheinlichkeit von 0.8667
2. *t* mit einer Wahrscheinlichkeit von 0.1111
3. *dd* mit einer Wahrscheinlichkeit von 0.0089
4. *tt* mit einer Wahrscheinlichkeit von 0.0044
5. % (verschwindet) mit einer Wahrscheinlichkeit von 0,0089

(Wahrscheinlichkeiten auf die vierte Nachkommastelle gerundet)

Beispiel 3.1: Übergänge für das mhd. Zeichen *d*

und in die Substitutionstabelle gespeichert. Diese bildet dann die Grundlage für die Alignmentmethoden, also das Alignmentprogramm. Die Alignmentmethoden arbeiten mit den Daten der Substitutionstabelle in Form von Transitionen (siehe Kapitel 4.2.1). Bei der Erstellung der Transitionen werden die Wahrscheinlichkeiten der Zeichenübergänge negativ logarithmiert, so dass die Transition mit dem kleinsten Wert die mit der höchsten Wahrscheinlichkeit ist.

3.4 Alignmentmethoden

Das Ziel der Alignierung ist es, für die mittelhochdeutschen Wörter des Lexers eine neuhochdeutsche Entsprechung aus dem Grimm zu finden. Hierfür nutzt man verschiedene Alignmentmethoden. Grundsätzlich werden alle Lemmata der Wörterbücher verglichen, wobei man davon ausgeht, dass die zwei Wörter, die sich am ähnlichsten sind, ein Paar bilden. Das heißt, dass das eine die Entsprechung des anderen in der jeweiligen Sprache ist.

Sind sich zwei Wörter sehr ähnlich, so ist die 'Distanz' dieser Wörter gering. Beispielsweise ist die Distanz zwischen mhd. *hunt* und nhd. *hund* gering, da sich nur ein Buchstabe unterscheidet und die unterschiedlichen Buchstaben sich ähnlich sind. Deswegen kann man davon ausgehen, dass diese beiden Lemmata ein Paar bilden. Eine geringe Distanz wird durch eine hohe Wahrscheinlichkeit der Wortpaarung angezeigt. Um diese Wahrscheinlichkeit zu berechnen, nutzt das Programm den Dijkstra-Algorithmus und Dynamic Programming.

3.4.1 Dijkstra-Algorithmus

Der Dijkstra-Algorithmus ist bekannt aus der Graphentheorie und wird genutzt, um den kürzesten Weg in einem kantengewichteten Graphen zu finden. Dies geschieht, indem die Kosten der einzelnen Kanten zwischen einem Anfangs- und Endpunkt addiert werden, und man dann den kostengünstigsten Weg auswählt.
Für die Alignierung von Wörtern muss der Algorithmus abgewandelt werden. Anstatt die Kosten, im Fall der Alignierung also die Übergangswahrscheinlichkeiten eines Zeichenübergangs, zu addieren, multipliziert man diese. Um trotzdem mit dem „Weg der geringsten Kosten“ arbeiten zu können, wurden die Wahrscheinlichkeiten der einzelnen Übergänge negativ logaritmiert. Eine Wahrscheinlickeit von 100% hat also den Wert -0,0 und je kleiner die Wahrscheinlichkeit wird, dest größer ist der Wert, den man durch den negativen Logarithmus erhält.
Im Fall der Alignierung von mittelhochdeutschen und neuhochdeutschen Wörterbucheinträgen kann man diesen Algorithmus nun gut anwenden.

Das zu vergleichende Wortpaar wird Buchstabe für Buchstabe betrachtet. Für jedes mittelhochdeutschen Zeichen ist aus der Substitutionstabelle bekannt, in welche neuhochdeutschen Zeichen es sich gewandelt haben kann. Diese Möglichkeiten werden nun durchlaufen und die Beste ausgewählt. Erreicht das Alignment eine bestimmten Grenzwert, die sogenannte Schranke, so bricht der Dijkstra-Algorithmus den Alignierungsversuch für das aktuelle Wortpaar ab.

Der Anfangswert für die Schranke liegt im Originalprogramm bei 1%. Je-

de erfolgreiche Alignierung erhöht diesen Wert um ihre eigene Wahrscheinlichkeit, so dass die folgenden Vergleiche schneller abgebrochen werden. Finden sich für ein mittelhochdeutsches Lemma in den verschiedenen Vergleichen mehrere mögliche neuhochdeutsche Alignments, ist also davon auszugehen, dass dasjenige mit der höchsten Wahrscheinlichkeit die korrekte Lösung ist. Diese wird dann in der Alignmentdatenbank gespeichert.

3.4.2 Dynamic Programming

Dynamische Programmierung ist eine aus der Mathematik bekannte Methode, die beispielsweise in der Bioinformatik genutzt wird, um DNS-Sequenzen zu alignieren. Diese werden als Folge von Buchstaben dargestellt und man versucht Beziehungen und Verwandtschaften zwischen den Sequenzen zu finden. Dementsprechend ist es naheliegend, diese Methode auf das vorliegende Problem der Sprach-Alignierung anzuwenden.

Der Algorithmus erstellt für das zu alignierende Wortpaar eine Matrix und berechnet über einen idealen Weg durch die Matrix die Wahrscheinlichkeit für das Alignment. Dynamic Programming bricht also auch beim Vergleich von zwei vollkommen unabhängigen Wörtern (z.B. mhd. *vluoch* und nhd. *bäckerei*) erst ab, wenn die Matrix komplett berechnet wurde. Eine Idee des Alignment-Tools war es, dies zu verhindern. Durch das setzten einer Schranke, welche anzeigt, dass die Alignierung zweier Wörter nicht weiter berechnet werden muss, sobald die Wahrscheinlichkeit unterhalb dieser Schranke liegt, wird das Alignment abgebrochen, da Alignments von zu geringer Ähnlichkeit uninteressant sind.

Kapitel 4

Ideen zur Verbesserung des Alignments

In diesem Kapitel werden die Ansätze vorgestellt, die das Alignment der Wörterbucheinträge verbessern sollen. Diese werden dann in den folgenden Unterkapiteln genauer beschrieben.

Wie schon erwähnt bedeutet Alignment, dass zu einem mittelhochdeutschen Lemma aus dem Lexer ein möglichst ähnliches neuhochdeutsches Lemma aus dem Grimm gefunden werden soll. Da der Lexer ca. 80.000, und der Grimm ca. 300.000 Einträge umfasst, müssen diese Ansätze auf Grund der Menge an anzustellenden Vergleichen effizient sein.

Es ist natürlich naheliegend, dass jedes Lemma im Lexer jeweils mit jedem Lemma im Grimm verglichen wird. Doch diese Vorgehensweise ist sehr zeit- und rechenintensiv. Veranschlagt man für das Vergleichen zweier Lemmata nur 0,001 Sekunden, so dauerte es gut neun Monate um alle Zuordnungspaare aus Lexer und Grimm zu finden. Deswegen wurden bei der Entwicklung des Alignment-Tools Methoden entwickelt, die die Suche einschränken und beschleunigen.

4.1 Linguistischer Ansatz

Der linguistische Ansatz zur Verbesserung des Alignments bezieht sich auf sprachwissenschaftliche Prinzipien, auf Grund derer die Datenbasis des Programms geändert und seine Arbeitsweise angepasst werden. Natürlich kann es sehr schwer sein, allgemeingültige Prinzipien zu finden, die sich dann technisch umsetzten lassen. Dies gründet sich schon darauf, dass Sprache kein einheitliches System bildet, welches sich homogen entwickelt, sondern von starken regionalen Faktoren, wie zum Beispiel Dialekten, beeinflusst wird.

Da die Datengrundlage des Alignments historische Wörterbücher sind, spiegeln sich auch in diesen diese Einflüsse wider. Wie schon eingangs erwähnt, belegen sowohl der Lexer als auch der Grimm ihre Einträge an historischen Quellen. Erst 1876 tagte in Berlin die „Erste orthographische Konferenz“ (auch „Konferenz zur Herstellung größerer Einigkeit in der deutschen Rechtschreibung“ genannt), mit der die Rechtschreibung im deutschen Sprachraum vereinheitlicht werden sollte. Auch die Veröffentlichung des ersten Wörterbuchs von Konrad Duden 1880 mit dem Titel „Vollständiges orthographisches Wörterbuch der deutschen Sprache“ (der erste Duden) trug maßgeblich zu dieser Einheit bei. Es wurde zu einem orthographischen Nachschlagewerk und seine Schreibweisen setzten sich nach und nach durch.

Im Gegensatz zu diesem fast pädagogisch angehauchtem Wörterbuch, welches die Regeln der Orthographie sammelte und somit bis zu einem gewissen Grad prägte, wollen der Grimm und der Lexer eher ihre jeweilige Sprache repräsentieren. Da der Lexer ausschließlich Einträge aus mittelhochdeutschen Quellen beinhaltet, wurde er zeitlich von keiner Orthographiereform berührt. Da auch der Grimm seine Einträge an Quellen belegt und die Arbeit an ihm 1838 begann, kann auch er zumindest in den ersten Jahrzehnten von keiner Rechtschreibreform betroffen gewesen sein. Beide Wörterbücher spiegeln also eher die Sprachsituation ihrer jeweiligen Zeit wider. Jeder Dialekt sprach anders als die anderen und jeder schrieb wie er sprach. Diejenigen, die schreiben konnten, taten dies auf den deutschen Sprachraum bezogen mit wenig Regelmäßigkeit.

Trotz allem kann man in der Sprachentwicklung Muster entdecken, die sich großflächig, wenn auch vielleicht nicht überall, durchzusetzten vermochten. Zu diesen gehört zum Beispiel die Verschmelzung von *nc*, welches im Mittelhochdeutschen noch oft am Wortende zu finden ist, zu dem *ŋ*-Laut (*ng*-Laut). Ein Beispiel dafür ist das Wortpaar mhd. *klanc* und nhd. *klang*.

Richard von Kienle [8] schreibt dazu, dass *c* die mittelhochdeutsche Schreibung für *g* im Auslaut eines Wortes war. Weiterhin schreibt er:

Kienle schreibt, dass sich mhd. *nc* zu nhd. *ŋ* entwickelt. Dies würde für das obige Beispiel mhd. *klanc* und nhd. *klang* bedeuten, dass sich *n* zu *ŋ (ng)* entwickelt und das *c* verschwindet. Für die Zwecke dieses Alignment-

> „In der Lautgruppe *ng* gilt noch mhd. der Lautwert *ŋ* + *g* (daher im Auslaut mhd. *-nc* = *ŋ* + *k*), der nhd. schriftsprachlich allgemein, mundartlich häufig zu *ŋ* assimiliert wird.“
>
> (Kienle, „Historische Laut- und Formenlehre des Deutschen“ [8], S. 122)

Tools ist diese rein linguistische Betrachtung aber nicht sinnvoll. Wichtig ist es festzustellen, dass sich der *ŋ*-Laut, der im Neuhochdeutschen dann durch *ng* schriftlich realisiert wird, in Verbindung von *n* + *c* im Auslaut entwickelt.

Dieses Beispiel zeigt ein Muster, welches großflächig auf den deutschen Sprachraum Gültigkeit hat: Laute entwickeln sich in Abhängigkeit von ihren Nachbarlauten. Weitere Faktoren sind die Betonung der Silbe (unbetonte Silben und somit ihre Laute werden gern 'verschluckt'), ob ein Laut Anfangs- oder Endlaut ist, und so weiter.

Im Folgenden sollen einige Vorschläge vorgestellt werden, wie man diese Faktoren der Lautentwicklung technisch umsetzten kann, um somit das Alignment und langfristig die Linguistik zu unterstützen.

4.1.1 Basiskorpus

Zunächst wird der Basiskorpus betrachtet. Dieser bildet die Grundlage zur Erstellung der Substitutionstabelle und somit des Alignments.

Wie schon in Kapitel 3.2 beschrieben, wurden im Korpus teilweise mehrere Buchstaben zu Zeichen zusammengefasst, womit die Wahrscheinlichkeiten in der Substitutionstabelle beeinflusst werden. Ein Beispiel hierfür ist das Wortpaar *holzdinc ; holzdi[ng]%*. In der Zählung für die Wahrscheinlichkeiten wird auf Grund dieser Notation gezählt, dass *n* zu *ng* wird und dass *c* verschwindet. Richtiger wäre jedoch $n \rightarrow n$ und $c \rightarrow g$, oder auch *[nc]* $\rightarrow$ *[ng]*. Die Gründe dafür wurden in der Einleitung zu Kapitel 4.1 beschrieben.

Auch beim Wortpaar *überhoeric ; überhörig* könnte es sinnvoll sein, die Buchstaben *ic* und *ig* zu einem Zeichen zu gruppieren. Dadurch würden dieses Vorkommen aus den Wahrscheinlichkeitsberechnungen der jeweiligen Einzelbuchstaben entfallen.

Zwar sind im originalen Basiskorpus schon einige Buchstabengruppen mittels eckiger Klammerung als solche ausgezeichnet, doch dies beschränkt sich häufig auf Fälle, in denen sich aus Buchstabengruppen ein Einzelbuchstabe entwickelt oder umgekehrt. Dies kann man also noch erweitern.

Das Ziel all dieser Maßnahmen ist es, die Wahrscheinlichkeiten der Lautverschiebungen, bzw. Lautentwicklungen, anzupassen und zu präzisieren. So werden folglich auch die Wahrscheinlichkeiten für ein gefundenes Alignment beeinflusst und eventuell werden dadurch Alignment erzielt, die vorher nicht gefunden wurden.

4.1.2 Substitutionstabelle

Weiterhin kann man die erstellte Substitutionstabelle betrachten. Wie in Kapitel 3.3 beschrieben, durchläuft das Programm bisher die Wörter des Basiskorpus zeichenweise und nimmt die Wahrscheinlichkeiten der einzelnen Zeichen samt ihrer Wandlung in die Substitutionstabelle auf. In Wörtern verändern sich einzelne Zeichen allerdings nicht unabhängig von ihrer Umgebung.

Um diese Tatsache nicht unbeachtet zu lassen, wurde die Idee verfolgt das Speichern von Buchstabengruppen unabhängig von der Klammerung im Basiskorpus zu erweitern. Das heißt, dass auch Buchstabengruppen aufgenommen werden, die nur aus Einzelbuchstaben bestehen und nicht durch Klammern als Gruppe gekennzeichnet wurden. Somit wird nicht nur die Umgebung der einzelnen Buchstaben mit betrachtet, sondern es gelangen auch prägnante Buchstabengruppen wie Prä- und Suffixe (Affixe) in die Substitutionstabelle.

Ein Datenbankbestand von übersetzten Affixen kann bei der Alignierung helfen, da man sich somit auf den mittelhochdeutschen Wortstamm konzentrieren kann. Weiterhin verringert sich durch die Substitution von Lautgruppen die Zahl der aufmultiplizierten Wahrscheinlichkeiten.

Stellt man sich beispielsweise ein Wort mit neun Buchstaben vor. Jeder der Buchstaben kann mittels der Substitutionstabelle mit einer Wahrscheinlichkeit von 0.9 in seinen Zielbuchstaben substituiert werden. Dann wäre die Gesamtwahrscheinlichkeit für das Alignment dieses Wortes $0.9^9 = 0.387$. Obwohl die einzelnen Übergangswahrscheinlichkeiten der Buchstaben also sehr hoch sind, ist die Endwahrscheinlichkeit relativ gering. Kann man nun aber einzelne Buchstaben zu Affixen verbinden, erhöht sich die Gesamtwahrscheinlichkeit. Nimmt man an, dass das neun-buchstabige Wort einen jeweils drei Buchstaben langen Präfix und Suffix mit jeweils einer Übergangswahrscheinlichkeit von 0.8 hat, so ergibt sich eine Gesamtwahrscheinlichkeit von $0.8 * 0.9^3 * 0.8 = 0.467$. Die Gesamtwahrscheinlichkeit steigt also, obwohl die Wahrscheinlichkeit der Affixe geringer ist, als die der einzelnen Buchstabenübergänge.

Da der Alignierung durch die Aufnahme von Affixen und Lautgruppen nun nicht nur hauptsächlich Einzelbuchstaben zur Verfügung stehen, verringert sich die Zahl der Alignierungsschritte und somit erhöht sich die Gesamtwahrscheinlichkeit des gefundenen Alignments. Der Nachteil ist hingegen, das größere Buchstabengruppen auch mehr Rechenleistung und somit mehr Zeit erfordern.

Der wichtigste Aspekt dieser Änderung ist allerdings, dass die Buchstaben und Laute nicht mehr nur als unabhängige Einheiten betrachtet werden. Ihr Umfeld wird, wenn auch in technisch eingeschränkter Form, mit in Betracht gezogen.

4.1.3 Wortstruktur - Wortzerlegung

Bei langen Wörtern erreicht man oft nur ein schlechtes Alignment, da sich die Wahrscheinlichkeiten für die einzelnen Lautverschiebungen im Wort multiplizieren und somit die Endwahrscheinlichkeit der gefundenen Alignierung sehr gering ist.

Um auch bei langen Wörtern ein Ergebnis zu erreichen, kann man versuchen, Wörter zu zerlegen und somit kürzere Teilwörter zu betrachten. Für diese kann dann eventuell ein Alignment gefunden werden. Dies kann auf Grund der Alphabetisierung der Wörterbücher über die Abspaltung gemeinsamer Präfixe geschehen (beispielsweise kämen nach dem Lemma *haus* zunächst alle Lemmata, die mit *haus* beginnen), oder man kann das Analyse-Tool von Prof. Seipel und Luise Borek, das Morphem-Annotations-Tool [11], nutzen, welches Wortbildungsstrukturen erkennen kann.

Im Folgenden werden zwei Wortbildungen, das Kompositum und das Derivat, vorgestellt um zu verdeutlichen, was Wortzerlegung bedeutet.

Das Kompositum

Ein Kompositum (pl. Komposita) ist das Ergebnis einer Komposition, einer Form der Wortbildung, bei der mindestens zwei selbstständig vorkommende Wörter ein Neues bilden.

Stuhlbein $\rightarrow$ Stuhl + Bein

Beispiel 4.1: Das Kompositum *Stuhlbein*

Da Wörterbücher alphabetisch geordnet sind, kann man davon ausgehen, dass bei einem Kompositum das erste Teilwort schon als Lemma aufgetaucht ist und somit schon mit seiner Alignierung in der Datenbank gespeichert wurde.

Ist das der Fall, so kann man das Kompositum in seine Teilwörter zerlegen und muss nun nur noch das zweite noch unbekannte Teilwort alignieren, wie Beispiel 4.2 zeigt.

1. vluoch - *stm.* [...]
2. vluoch-salm - *stm.* [...]

Beispiel 4.2: Zerlegung des mhd. Wortes *vluochsalm*

Das Lemma (mhd.) *vluoch* wurde schon mit (nhd.) *fluch* aligniert, bevor man auf das Lemma (mhd.) *vluoch-salm* (nhd. *fluchpsalm*) trifft. Um dieses zu alignieren, muss man nun nur noch das zweite Teilwort (mhd.) *salm* alignieren.

Aus einem langen Wort sind so zwei kurze geworden. Die Wahrscheinlichkeiten der zwei Alignierungen müssen nun noch multipliziert werden und man erhält die Gesamtwahrscheinlichkeit für das Wort *vluoch-salm*.

Das Derivat

Ein Derivat ist das Ergebnis einer Derivation, einer Form der Wortbildung, bei der ein Wort durch die Kombination eines Grundwortes und eines Affixes entsteht.

Menschheit → Mensch + heit

Beispiel 4.3: Das Derivat *Menschheit*

Auch bei Derivaten kann die Zerlegung des zur alignierenden Wortes zu guten Ergebnissen führen, da der Lexer und der Grimm oft überraschen und zum Beispiel Suffixe wie *heit* und *keit* teilweise als Lemmata führen. Wäre dies nicht der Fall, müsste man diese häufigen Affixe aufnehmen, damit die Wortzerlegung von Derivaten erfolgreich ist.

4.2 Technische Optimierung

Im Bereich der technischen Optimierung wurde schon im Prozess der Ideenfindung zu Beginn dieser Arbeit deutlich, dass es hier wenige Ansatzpunkte gibt, an denen man große Verbesserungen erzielen kann. Für die Diplomarbeit wurde dann die Aufgabe gestellt, die Transitionen zu modifizieren. Dies wird im folgenden Kapitel beschrieben. Weitere Ideen werden im Kapitel 7, dem Ausblickskapitel, kurz vorgestellt.

In Tabelle 4.1 sieht man einen Stichprobentest zu den Transitionen. Man sieht darin das mittelhochdeutsche Lemma (MHD), das entsprechende neuhochdeutsche Lemma (NHD), die Zeit, die die bisherige Transition ('X-Transition') benötigt, die Zeit, die die neue Transition ('XY-Transition') benötigt, und die Differenz der beiden. Sie zeigt, dass sich durch die Einführung der XY-Transition pro Berechnung des Alignments nur in einigen Fällen eine Zeitersparnis erzielen lies. Da 80.000 Wörter im Lexer aligniert werden sollen, könnte dies trotzdem zu einer hilfreichen Verbesserung der Rechendauer führen.

MHD	**NHD**	**Zeit X**	**Zeit XY**	Differenz
aptei	abtei	5.00 s	4.00 s	1.00 s
blat	blatt	5.00 s	4.00 s	1.00 s
tac	tag	5.00 s	4.00 s	1.00 s
vluoch	fluch	5.00 s	5.00 s	-
smërze	schmerz	7.00 s	7.00 s	-
keiser	kaiser	8.00 s	7.00 s	1.00 s
ouge	auge	4.00 s	4.00 s	-
künec	könig	5.00 s	4.00 s	1.00 s
spil	spiel	11.00 s	9.00 s	2.00 s
hërze	herz	5.00 s	5.00 s	-
vart	fahrt	6.00 s	5.00 s	1.00 s

Tabelle 4.1: X-Transition vs. XY-Transition

4.2.1 Transitionen

Wie schon beschrieben, nutzt das Alignmentprogramm die Übergangswahrscheinlichkeiten, die in der Substitutionstabelle in der Datenbank gespeichert werden um Wörter zu vergleichen. Um mit diesen schnell und effizient arbeiten zu können, werden diese Übergänge in Form von Transitionen in den Zwischenspeicher von Prolog geladen. Im bisherigen Programm wurden die Transitionen nur unter Beachtung des mittelhochdeutschen Zeichens erstellt. Beispiel 4.4 zeigt, wie diese aussehen.

```
alignment_transitions(w, [
   ([w], [w], 0.14339448627969412),
   ([w], [], 2.477302050402781),
   ([w], [b], 3.1704492309627264),
   ([w], [h], 5.568344503761097),
   ([w], [u], 5.568344503761097) ]).
```

Beispiel 4.4: Die X-Transition für den Buchstaben *w*

Das heißt, dass das Programm obige Transition heraussucht, sobald es im mittelhochdeutschen Wort auf das Zeichen *w* trifft. Danach überprüft es den Wahrscheinlichkeiten nach, welche der fünf Möglichkeiten genutzt wird. Anzumerken sei hierbei, dass die Wahrscheinlichkeiten negativ logarithmiert sind. Dementsprechend ist die $w \rightarrow w$ - Transition die Wahrscheinlichste.

Gerade in einem Fall, bei dem sich nicht nur ein Buchstabe in einen anderen entwickelt, sondern in dem Buchstabengruppen betroffen sind, ist ersichtlich, dass hier viele Fehlentscheidungen getroffen werden können. Das Beispiel 4.5 soll dies verdeutlichen. Wenn eine falsche Transition benutzt wurde und es noch Alternativen gibt, dann führt dies natürlich dazu, dass eine dieser Alternativen ausprobiert wird. Stehen also wie in Beispiel 4.5 17 Transitionen zur Auswahl, führt das zu einem erhöhten Rechenaufwand.

Trifft das Alignmentprogramm auf den Buchstaben *c*, so weiß es zu diesem Zeitpunkt noch nicht, was danach kommt. Deswegen werden zunächst alle Transitionen herausgesucht, die mit *c* beginnen. Wie Beispiel 4.5 zeigt, sind dies sechs Transitionen mit insgesamt 17 Übergängen. Erst wenn der nächste Buchstabe im zu alignierenden Wort bekannt ist, kann diese Auswahl eingegrenzt werden.

Die Idee um dies zu verbessern ist nun, das auch der aktuelle Buchstabe des neuhochdeutschen Wortes, mit dem verglichen wird, mit in Betracht gezogen wird. Auf diese Weise könnte die Auswahl der auszuprobierenden Transitionen stark eingegrenzt werden und teilweise kann dadurch sogar direkt die richtige Transition gefunden werden.

```
alignment_transitions(c, [
   ([c], [g], 0.5663954749208014),
   ([c], [], 1.8191584434161694),
   ([c], [k], 2.10684051586795),
   ([c], [ch], 2.7999876964278956),
   ([c], [ck], 2.7999876964278956),
   ([c], [z], 4.304065093204169),
   ([c], [f], 4.997212273764115),
   ([c], [sch], 4.997212273764115) ]).
alignment_transitions(ch, [
   ([ch], [ch], 0.07913732055872386),
   ([ch], [h], 3.6054978451748854),
   ([ch], [k], 3.828641396489095),
   ([ch], [g], 4.116323468940876),
   ([ch], [ck], 4.5217885770490405) ]).
alignment_transitions(chs, [
   ([chs], [chs], -0.0) ]).
alignment_transitions(ck, [
   ([ck], [ck], -0.0) ]).
alignment_transitions(cks, [
   ([cks], [cks], -0.0) ]).
alignment_transitions(cz, [
   ([cz], [cz], -0.0) ]).
```

Beispiel 4.5: Die X-Transition für den Buchstaben *c*

Kapitel 5

Umsetzung und Testergebnisse

Im folgenden Kapitel soll die Umsetzung der Ideen aus Kapitel 4 erläutert werden. Weiterhin befasst sich dieses Kapitel mit den Testergebnissen und deren Auswertung.

Um Vergleichswerte zu erhalten, wurden drei Durchläufe mit dem unveränderten Original-Tool durchgeführt. Bei diesen Tests wurde die Anzahl der gefundenen Alignments, die Dauer des Testdurchlaufs, sowie die Durchschnittswahrscheinlichkeit der gefundenen Alignments erfasst. Alle Test wurden mit einer leeren Alignmentdatenbank durchgeführt, so dass tatsächlich jedes Wort aligniert wird und man die hierfür benötigte Zeit messen kann. Um ein möglichst vielseitiges Ergebnis zu bekommen, wurden für die drei Testläufe die Lemmata des Lexers ausgewählt, die mit *j*, *o* und mit *v* beginnen. Für den Anfangsbuchstaben *j* findet man im mittelhochdeutschen Wörterbuch 449 Einträge, für *o* findet man 705 Einträge und für *v* findet man 7.743 Einträge.

Diese Anfangsbuchstaben wurden jeweils aus einem speziellen Grund ausgewählt. Mit dem Buchstaben *j* beginnen nur relativ wenige Lemmata des Mittelhochdeutschen Wörterbuchs. So steht dieser Testlauf für einen relativ kleinen Datenbestand kürzerer Vergleichsdauer. Der Buchstabe *o* wurde gewählt, da die Durchschnittswahrscheinlichkeit der Ergebnisse mit über 23% relativ hoch ist und man somit Einflüsse auf die Wahrscheinlichkeitswerte besser beurteilen kann. Der Anfangsbuchstabe *v* wurde vor allem gewählt, um Veränderungen an der Zeit feststellen zu können.

Ein mittelhochdeutsches *v* entwickelt sich im Neuhochdeutschen bevorzugt in ein *f* (mit 58,75%) oder es bleibt ein *v* (mit 41,25%). Diese beiden Entwicklungen sind die Wahrscheinlichkeit betreffend so nah beieinander, dass keine von beiden realistisch betrachtet unwahrscheinlicher ist als die andere. Somit kann sich jedes mittelhochdeutsche Lemma, welches mit *v* beginnt, folglich in ein neuhochdeutsches Lemma, welches mit *v* oder *f* beginnt, entwickeln. Im Grimm gibt es unter diesen beiden Buchstaben insgesamt fast 31.000 Einträge mit denen das mittelhochdeutsche Lemma im schlimmsten Fall verglichen werden müsste. Somit ist *v* also der offensichtliche Kandidat,

wenn man einen Testlauf anstrebt, bei dem viele Vergleiche angestellt werden müssen und sich eine hohe Testdauer ergibt.

Tabelle 5.1 zeigt die Ergebnisse der Ermittlung der Vergleichswerte - also der Werte, die sich bei Benutzung des originalen Alignment-Tools ergeben haben.

Präfix	**DS**	**Ergebnisse**	**Zeit** (hh:mm:ss)	**Wahrscheinlichkeit**
j	449	204	00:50:49	13,12%
o	705	355	00:59:25	23,08%
v	7743	3868	13:49:24	13,03%
Gesamt	8897	4427	15:39:38	13,84%

Tabelle 5.1: Ergebnisse der Basisdatenermittlung

Diese Form der Ergebnistabelle wird noch häufiger auftauchen. Sie setzt sich aus folgenden fünf Werten zusammen: dem Anfangsbuchstaben der Lemmata (Präfix); der Anzahl der bearbeiteten Lemmata (Datensätze DS), welche in den weiteren Tabellen weggelassen werden, da sich dieser Wert nicht ändert; der Anzahl der gefunden Alignments (Ergebnisse); der Dauer des Testlaufs (Zeit) und der Durchschnittswahrscheinlichkeit der gefundenen Ergebnisse.

5.1 Basiskorpus

Die Änderungen an dem Basiskorpus kann man in drei Hauptgruppen einteilen: Die Buchstabengruppe *ig*, die Buchstabengruppe *ng* und Sonstiges. Insgesamt wurden an 80 Wortpaaren, also 4,659% des Korpusbestandes, Änderung vorgenommen.

bisher	**überarbeitet**
vr[ei]s**ic**;fr[ei]s**ig**	vr[ei]s**[ic]**;fr[ei]s**[ig]**
i[nn]ewend**ic**;in%wend**ig**	i[nn]ewend**[ic]**;in%wend**[ig]**
i[rr]**ec**;i[rr]**ig**	i[rr]**[ec]**;i[rr]**[ig]**
n[iu]nz**ic**;n[eu]nz**ig**	n[iu]nz**[ic]**;n[eu]nz**[ig]**
bild**ec**lî[ch]e;bild**%%**li[ch]%	bild**[ec]**lî[ch]e;bild**%**li[ch]%
riht**ec**h[ei]t;ri[ch]t**ig**h[ei]t	riht**[ec]**h[ei]t;ri[ch]t**[ig]**h[ei]t
...	

Beispiel 5.1: Änderungen betreffend die Buchstabengruppe *ig*

bisher	**überarbeitet**
kla**nc**;kla**ng**	kla**[nc]**;kla**[ng]**
si[ch]%li**nc**;si[ch]eli**ng**	si[ch]%li**[nc]**;si[ch]eli**[ng]**
ga**nc**haft;ga**ng**haft	ga**[nc]**haft;ga**[ng]**haft
antla**nc**;antla**ng**	antla**[nc]**;antla**[ng]**
besa**nc**t;bese**ng**t	besa**[nc]**t;bese**[ng]**t
sa**nc**hûs;sa**ng**h[au]s	sa**[nc]**hûs;sa**[ng]**h[au]s
holzdi**nc**;holzdi**ng**	holzdi**[nc]**;holzdi**[ng]**
...	

Beispiel 5.2: Änderungen betreffend die Buchstabengruppe *ng*

	bisher	**überarbeitet**
	tœt**ig**en;töt**%%**en	tœt**[ig]**en;töt**%**en
	en**%[ph]**inden;em**pf**inden	en**[ph]**inden;em**[pf]**inden
	ungev[uo]c**%%**;ungefüg**ig**	ungev[uo]c**%**;ungefüg**[ig]**
	m[ei]st**ec**;m[ei]st**%%**	m[ei]st**[ec]**;m[ei]st**%**
	...	
a)	hant**twehele**;handtu%[ch]%%%	hant**t[uo][ch]**;handtu[ch]
b)	ungel[ou]pli[ch];ung%l[au]bli[ch]	
c)		m[uo]ter;mu[tt]er

Beispiel 5.3: Sonstige Änderungen

Die Beispiele 5.1 und 5.2 zeigen, dass oft darauf geachtet wurde insbesondere die genannten Buchstabengruppe mit Klammern zu versehen. Folgender Gedanke steckt dahinter: Durch die Klammerung entfernt man dieses Vorkommen der Einzelbuchstaben bei der Erstellung der Substitutionstabelle aus deren Wahrscheinlichkeitsberechnung. Wird also der Basiskorpus durchlaufen und die Übergangswahrscheinlichkeiten für *c* berechnet werden, wird das Vorkommen von *c* innerhalb der Klammerung nicht beachtet. Dafür werden die Buchstabengruppen *ec* und *nc* in die Substitutionstabelle aufgenommen. Dadurch nimmt man offensichtlich Einfluss darauf, mit welcher Wahrscheinlichkeit ein *c* zu einem *g* wird. Weiterhin gibt man somit indirekt zu verstehen, dass ein *c* in Verbindung mit *e* oder *n* sehr wahrscheinlich zu einem *g* wird. Indem man so Einfluss auf die berechneten Wahrscheinlichkeiten nimmt, verbessert man die Wahrscheinlichkeiten der gefunden Alignments, falsche Alignments werden besser vermieden und eventuell können sogar mehr Alignmentergebnisse erzielt werden.

Beispiel 5.3 zeigt die sonstigen Änderungen, die vorgenommen wurden. Erklärungen sind für die mit a), b) und c) gekennzeichneten Wortpaare angebracht.

Wortpaar a), *hanttwehele ; handtu%[ch]%%%*, wurde verändert, da eine ethmyologische Überprüfung des neuhochdeutschen Wortes *tuch* ergab, dass es sich aus dem mittelhochdeutschen *tuoch* entwickelte [10]. Das mittelhochdeutsche *twehele* entwickelte sich aus *que[h]le* [9] und bedeutet ebenfalls *Tuch*. Während das mittelhochdeutsche *tuoch* im Grimm'schen Wörterbuch nicht mehr als Lemma geführt wird, tauchen sowohl *twe(h)le* als auch *[hand]qehle* als Lemmata im Grimm auf. Alle drei mittelhochdeutschen Formen für das Wort *Tuch* findet man im Lexer.

Wortpaar b), *ungel[ou]pli[ch] ; ung%l[au]bli[ch]*, wurde einmal aus dem Basiskorpus entfernt, da es doppelt auftauchte. Damit sich die Anzahl der Wortpaare insgesamt nicht änderte, wurde dafür das Wortpaar c), *m[uo]ter ; mu[tt]er*, eingeführt.

5.1.1 Testergebnisse

Die Testergebnisse bestätigen die zu Beginn dieses Unterkapitels aufgestellten Vermutungen.

Präfix	**Ergebnisse**	**Zeit** (hh:mm:ss)	**Wahrscheinlichkeit**
j	204 (±0,00%)	00:50:49 (±0,00%)	13,12% (±0,00%)
o	354 (-0,28%)	00:59:20 (-0,14%)	24,52% (+1,44%)
v	3946 (+2,02%)	13:47:30 (-0,23%)	13,97% (+0,93%)
Gesamt	**+1,74%**	**-0,21%**	**+0,91%**

Tabelle 5.2: Ergebnisse der Änderungen am Basiskorpus

Man sieht im Vergleich zu den Basiswerten, dass bei dem Präfix *v* leichte Verbesserungen bei der Anzahl der Ergebnisse erzielt wurden. Bei *o* sank die Anzahl der Ergebnisse geringfügig. Dies muss näher betrachtet werden, da dies nicht vorher gesehehn wurde.

Die Auswertung der Ergebnisse zeigte, dass für *o* vier Alignments nicht mehr gefunden wurden (für die mhd. Lemmata *obenc*, *oberwërc* und zweimal *oli*), welche aber im Original-Tool auch nur teilweise richtig aligniert wurden (mit *oben*, *oberberg* und *oi*). Es ist also eine Verbesserung, dass falsche Alignments vermindert wurden. Dafür wurden drei neue Alignments gefunden (zu den mhd. Lemmata *oberecheit*, *ortpic* und *ovenkrucke*). Die begründet die Verminderung der Ergebniszahl um ein Ergebnis (also 0,28%).

Insgesamt stieg die Anzahl der gefundenen Alignments um 1,74%. Die Dauer des Alignments hat sich mit 0,21% nur unwesentlich verkürzt und die Durchschnittswahrscheinlichkeit ist um knapp 1% gestiegen. Beim Präfix *j* zeigten sich interessanterweise keine prozentualen Veränderungen.

Was man bei diesen auf den ersten Blick recht unerheblichen Verbesserungen nicht vergessen darf, ist, dass nur an 4,7% des Basiskorpus Veränderungen vorgenommen wurden und diese auch lediglich im Bezug auf spezielle Eigenschaften in den Wörtern. Sollte der Datenbestand des Basiskorpus erweitert werden und man die Daten mit tiefgehenden linguistischen Kenntnissen über den jetztigen Grad hinaus analysiert und bearbeitet, so kann das Alignment der Wörterbücher davon durchaus profitieren.

5.2 Substitutionstabelle

Zur Umsetzung der in Kapitel 4.1.2 beschriebenen Idee war es nötig, mehrere Prädikate (Methoden) des ursprünglichen Alignment-Tools zu verändern. Die Umsetzung dieser Änderungen kann man am Code in Kapitel Code: alignment_sub_create_table am Ende des Dokuments nachprüfen.

5.2.1 Implementierung

Ein sehr wichtiges neues Prädikat ist `codes_to_substrings/3`. Ihm wird die maximale Länge der gesuchten Buchstabengruppen und ein Lemma in Form einer Liste der einzelnen Zeichen übergeben. Zurück gibt es die Buchstabengruppen als Strings.
Für die Testläufe wurde die Länge dieser Buchstabengruppen auf drei Zeichen festgelegt. Somit ist ein Großteil der im Neuhochdeutschen genutzten Affixe in der neuen Substitutionstabelle enthalten.

```
/* codes_to_substrings(Length, Codes, Strings) <-
      */

codes_to_substrings(Length, Codes, Strings) :-
   findall( String,
      ( middle_sequence(Codes, Xs),
        length(Xs, L),
        between(1, Length, L),
        atomic_list_concat(Xs, String_0),
        delete_percents(String_0, String) ),
      Strings ).

delete_percents(Name_1, Name_2) :-
   name_exchange_sublist([["%",""]], Name_1, Name),
   ( Name = '' ->
     Name_2 = '%'
   ; Name_2 = Name ).
```

Untersuchen wir die Arbeitsweise der Methode anhand eines Beispiels. Das Lemma, welches in Buchstabengruppen zerlegt werden soll, sei aus dem Wortpaar `toetigen ; töt%%en` (Teil des Basiskorpus), der neuhochdeutsche Partner `töt%%en`. Da das Lemma keine eckigen Klammern zur Buchstabengruppierung aufweist, ist die Liste, die an `codes_to_substrings/3` übergeben wird
`Codes = [ t, ö, t, %, %, e, n ].`

Nun werden mittels des Meta-Prädikats `findall/3` alle Stings in einer Liste gesammelt, die durch folgende Suche gefunden werden:
Gesucht sind alle Teillisten von `Codes`, deren Länge zwischen 1 und `Length` (also hier: 3) liegt.
Diese Teillisten werden zu Strings verkettet und die Prozentzeichen entfernt oder teilweise doppelte Prozentzeichen durch ein Einzelnes ersetzt.

```
Codes = [ t, ö, t, %, %, e, n ]

Middle Sequences:
   [ [t], [t, ö], [t, ö, t], [t, ö, t, %], ...,
     ..., [%, %, e, n], [%, e, n], [e, n], [n] ]

Filter nach Länge:
   [ [t], [t, ö], [t, ö, t], [ö], [ö, t],
     [ö, t, %], [t, %], [t, %, %], [%, %], [%], ... ]

Zu Strings verketten:
   [ t, tö, töt, ö, öt, öt%, t%, t%%, %%, %, ... ]

Prozentzeichen entfernen:
  [ t, tö, töt, ö, öt, öt, t, t, %, %, ... ]
```

Die Entfernung der Prozentzeichen ist ein wichtiger Schritt. Wie schon angesprochen, stehen Prozentzeichen für Buchstaben, die in der Wortentwicklung verschwunden sind. Für die Substitutionstabelle ist nun wichtig, dass man in der Erstellung dieser Buchstabengruppen darauf achtet, dass `t` und `t%` oder `t%%` das Gleiche sind. Einzeln stehende Prozente, also `%` oder `%%` dürfen allerdings nicht komplett verschwinden, sondern müssen als `%` in die Liste der Buchstabengruppen aufgenommen werden.

Später wird die so über alle Lemmas gesammelte Liste von Doppelvorkommen bereinigt und alphabetisch geordnet.

Nun wird in einen zweiten Lauf über alle Lemmata geprüft, in welche Buchstabengruppe sich die gesammelten Buchstabengruppen jeweils entwickelt haben und wie oft dies geschehen ist. So erfährt man die Häufigkeit (Frequenz) der Übergänge und somit ihre Wahrscheinlichkeit. Weiterhin wird

auch das Vorkommen jeder mittelhochdeutschen Buchstabengruppe gespeichert, so dass man die Möglichkeit hat, nur häufige Übergänge in die Substitutionstabelle aufzunehmen. Die Möglichkeit Übergänge nach bestimmten Kriterien zu filtern und nur die Ausgewähten zu speichern, bieten die Prädikate in der Datei `alignment_find_frequency.pl` in Kapitel Code: alignment_find_frequency am Ende des Dokuments.

5.2.2 Testergebnisse

Die Testergebnisse bestätigen die vorhergesagten positiven Auswirkungen auf die Anzahl der gefunden Alignments und deren Wahrscheinlichkeiten. Tabelle 5.3 stellt diese Ergebnisse vor.

Präfix	**Ergebnisse**	**Zeit** (hh:mm:ss)	**Wahrscheinlichkeit**
j	275 (+34,80%)	02:51:29 (+237,45%)	21,70% (+8,58%)
o	433 (+21,97%)	02:27:32 (+148,30%)	39,54% (+16,46%)
v	5773 (+49,25%)	57:15:25 (+314,21%)	37,14% (+24,11%)
Gesamt	**+46,40%**	**+299,56%**	**+22,80%**

Tabelle 5.3: Ergebnisse der Änderungen der Substitutionstabelle

Wie man sehen kann, gab es in jedem der drei Testläufe einen erheblichen Anstieg der gefunden Alignments. Für den Präfix *j* wurden beispielsweise Alignments zu *jagethunt (Jagdhund), jâmerstimme (Jammerstimme), jegermeister (Jägermeister)* und *juncvrouwelîcheit (Jungfräulichkeit)* neu gefunden. Wie man sieht, sind das alles recht lange Worte, welche das originale Alignmentprogramm nur schwer alignieren kann.

Die ursprüngliche Substitutionstabelle beinhaltet nur wenige Buchstabengruppen, welche selten länger als zwei Buchstaben sind. Die Durchschnittslänge der Einträge in der Tabelle ist 1,5 Buchstaben. Aligniert man ein Wort wie *jâmerstimme*, welches aus elf Buchstaben besteht, so macht man durchschnittlich 7,3 Alignierungsschritte (Buchstabenübergänge), deren Wahrscheinlichkeiten sich aufmultiplizieren. Mit den neuen Buchstabengruppen in der Substitutionstabelle, deren Durchschnittslänge bei 2,5 Buchstaben liegt, reduzieren sich die Alignierungsschritte bei einem elf Buchstaben langen Wort auf durchschnittlich 4,4.

Sowohl die originale als auch die veränderte Substitutionstabelle weisen für ihre Buchstabenübergänge eine Durchschnittswahrscheinlichkeit von rund 0,5 auf. Bei einem elfbuchstabigen Wort erreicht man mit der originalen Substitutionstabelle also durchschnittlich eine Alignmentwahrscheinlichkeit von 0,63%. Diese liegt unterhalb der gesetzten Schranke (der unteren Grenze für die Wahrscheinlichkeit interessanter Alignments), welche bei

1% liegt. Mit der veränderten Substitutionstabelle erreicht man eine Durchschnittswahrscheinlichkeit von 4,73% und liegt somit oberhalb der Schranke.

Allein im Testlauf für *v* wurden knapp 2.000 Alignments mehr gefunden, also ein Plus von fast 50%. Insgesamt stieg die Anzahl der Ergebnisse um 46,40%. Auch die Durchschnittswahrscheinlichkeiten der gefunden Alignments stieg erheblich. Im Fall des Buchstaben *v* um über 24%. Insgesamt wurde ein Plus von 22,80% erziehlt.

Die negativen Vorhersagen zur Rechenzeit wurden ebenfalls bestätigt. Wie schon beschrieben, erfordert es wesentlich mehr Rechenleistung beständig mehrere Buchstaben im zu alignierenden Wort zu betrachten. Außerdem hat sich die Substitutionstabelle durch die Aufnahme weiterer Buchstabengruppen stark vergrößert. Die Dauer der Berechnungen hat sich insgesamt mit einem Anstieg von rund 300% fast vervierfacht.

5.3 Transitionen

Um den Rechenaufwand während des Vergleichs zweier Lemmata noch etwas zu verringern, wurde das Programm angepasst, so dass es auch das neuhochdeutsche Zeichen im zu vergleichenden Wort beachtet. Die vorgenommen Änderungen kann man in Code: XY-Transition nachsehen. Beispiel 5.4 zeigt die Form der neuen XY-Transition.

```
alignment_transition(w, w, [
   ([w], [w], 0.14339448627969412)]).
alignment_transition(w, '', [
   ([w], [], 2.477302050402781)]).
alignment_transition(w, b, [
   ([w], [b], 3.1704492309627264)]).
alignment_transition(w, h, [
   ([w], [h], 5.568344503761097)]).
alignment_transition(w, u, [
   ([w], [u], 5.568344503761097)]).
```

Beispiel 5.4: Die XY-Transition für den Buchstaben *w*

Das heißt nun, dass das Programm, wenn es im mittelhochdeutschen Wort auf das Zeichen *w* trifft, nicht einfach alle fünf *w*-Transitionen heraus sucht. Es zieht auch in Betracht, welches Zeichen im neuhochdeutschen Wort ansteht und sucht nun also genau die Transition heraus, die passt. Selbst falls noch immer mehrere Transitionen zur Auswahl stehen (siehe Beispiel 5.5 für

den Übergang $c \rightarrow c$: acht Übergänge statt vorher 17), so hat sich deren Anzahl doch soweit verringert, dass die Rechenzeit positiv beeinflusst wird.

```
alignment_transition(c, ch [
   ([c], [ch], 2.7999876964278956)])   .
alignment_transition(c, ck [
   ([c], [ck], 2.7999876964278956)])   .
alignment_transitions(ch, ch [
   ([ch], [ch], 0.07913732055872386)
alignment_transitions(ch, ck [
   ([ch], [ck], 4.5217885770490405)])   .
alignment_transitions(chs, chs [
   ([chs], [chs], -0.0) ])   .
alignment_transitions(ck, ck  [
   ([ck], [ck], -0.0) ])   .
alignment_transitions(cks, cks [
   ([cks], [cks], -0.0) ])   .
alignment_transitions(cz, cz [
   ([cz], [cz], -0.0) ])   .
```

Beispiel 5.5: Die XY-Transitionen für den Übergang $c \rightarrow c$

5.3.1 Testergebnisse

Die ausführlichen Testreihen konnten die Ergebnisse der Stichproben aus Tabelle 4.1 bestätigen.

Präfix	**Ergebnisse**	**Zeit** (hh:mm:ss)	**Wahrscheinlichkeit**
j	204 (±0,00%)	00:50:49 (±0,00%)	13,12% (±0,00%)
o	355 (±0,00%)	00:54:04 (-9,00%)	23,08% (±0,00%)
v	3868 (±0,00%)	12:22:11 (-10,52%)	13,03% (±0,00%)
Gesamt	**±0,00%**	**-9,85%**	**±0,00%**

Tabelle 5.4: Ergebnisse der Einführung der XY-Transition

Tabelle 5.4 zeigt, dass sich an der Anzahl gefundener Alignments und den Durchschnittswahrscheinlichkeiten im Vergleich zu den Basiswerten nichts geändert hat. Dies begründet sich damit, dass sich an dem Algorithmus zu Berechnung der Alignments und den zu Grunde liegenden Daten nichts geändert hat.

Die XY-Transition hat lediglich Auswirkungen auf die Zeit, die zur Berechnung der Alignments benötigt wird. Hier wurde eine Verbesserung um fast 10% erziehlt. Dies mag zunächst nach einem relativ niedrigen Wert klingen, doch wie schon beschrieben, wird der Nutzen dieser Modifizierung vor allem in Verbindung mit Buchstabengruppen relevant. Da diese im originalen Tool nur begrenzt vorhanden sind, war vorauszusehen, dass die positive Wirkung der XY-Transition gering ausfallen wird.

5.4 Wortzerlegung

Um die Wortzerlegung umzusetzten wurde von der Tatsache Gebrauch gemacht, dass die Grundlage der Daten alphabetisierte Wörterbücher darstellen. Das mittelhochdeutsche Lemma *hûs* (nhd. *Haus*) wird also vor *hûsbrôt* (nhd. *Hausbrot* - qualitativ schlechteres, für den Hausbedarf zubereitetes Brot) bearbeitet und in die Alignmenttabelle gespeichert.

Die Idee ist nun von einem zu zerlegenden Lemma alle Präfixe abzuspalten. Für *hûsbrôt* wären das beispielsweise *hûsbrô*, *hûsbr*, *hûsb* und so weiter. Nun prüft man für jeden dieser Präfixe, ob er als selbstständiges, schon aligniertes Lemma in der Datenbank gespeichert ist. Da die Wörterbuchdaten alphabetisch sortiert sind, ist das in unserem Beispiel für den Präfix *hûs* der Fall.

Danach wird das Restwort des Lemmas untersucht. In diesem Fall also *brôt*, welches ebenso ein eigenständiges Lemmat im Lexer ist. Dieses wird nun wiederum aligniert bzw. wird überprüft, ob es schon aligniert ist. Ist dies geschehen, muss man nur die Einzelwahrscheinlichkeiten von *hûs* und *brôt* multiplizieren um die Gesamtwahrscheinlichkeit des Alignments für *hûsbrôt* zu ermitteln.

Dieses Beispiel zeigt, dass die Wortzerlegung sehr auf das Alignment von Komposita abzielt.

5.4.1 Implementierung

Die Implementierung der Wortzerlegung kann man im Detail im Kapitel Code: alignment_word_partition am Ende dieser Arbeit nachvollziehen. Hier sollen nun grob die Artbeitsweise der Wortzerlegung und einige wichtige Hauptprädikate genauer betrachtet werden.

Dem Prädikat `word_partition/1` wird der mittelhochdeutsche Präfix übergeben, mit dem gearbeitet werden soll. Dies kann ein Anfangsbuchstabe sein, wenn alle Einträge dieses Buchstabens bearbeitet werden sollen, eine Vorsilbe wie *ver-* oder auch ein ganzen Wort. Die Wortzerlegung bearbeitet dann alle Lemmata des Lexer, die mit dieser Buchstabenfolge beginnen.

Zunächst wird eine Liste all dieser Lemmata erstellt und an das Prädikat `lemma_alignment/1` übergeben. Hier wird nun für jedes Lemma erst geprüft, ob es schon aligniert ist, oder, falls nicht, ob es sich alignieren lässt. Ist das nicht der Fall, wird die Wortzerlegung bekommen.

Das Prädikat `maximum_prefix_alignment/1` erstellt für das ihm übergebene Lemma eine Liste aller Präfixe und prüft für jeden, ob er als Eintrag in der Alignmentdatenbank vorhanden ist. Ist ein Präfix gefunden, für den das der Fall ist, werden Lemma und Präfix an das Prädikat `find_maximum_prefix_alignment/2` übergeben.

```
/* find_maximum_prefix_alignment(Prefix, Lemma)) <-
      */

find_maximum_prefix_alignment(Prefix, Lemma) :-
   lemma_exists_in_alignments(
      Prefix, PrePartner, PreSim),
   concat(Prefix, RestWord, Lemma),
   ( lemma_exists_in_alignments(
        RestWord, RestPartner, RestSim) ->
     ( Pairs = [RestSim:RestPartner] ,
       alignment_found(
          Lemma, PrePartner, PreSim, Pairs) )
     ; ( word_to_similarities(
            [step:4], RestWord, Pairs),
         alignment_found(
            Lemma, PrePartner, PreSim, Pairs) ) ).
```

`find_maximum_prefix_alignment/2` sucht nun zunächst für den Präfix das Alignment aus der Datenbank und speichert das Ergebnis `PrePartner` und die Wahrscheinlichkeit ab. Danach wird geprüft, ob der Rest des Lemmas schon aligniert ist. Ist das der Fall, so ist das neue Alignment schon fast gefunden. Gibt es das Restwort noch nicht in der Alignmentdatenbank, so versucht man das Restwort zu alignieren.

Nun übergibt man die gefunden Teilergebnisse an `alignment_found/4`.

```
/* alignment_found(
      Lemma, Prefix_Partner, Prefix_Sim, Rest_Pairs) <-
      */

alignment_found(
   Lemma, Prefix_Partner, Prefix_Sim, Rest_Pairs) :-
   ( Rest_Pairs \= [] ->
     ( last(Rest_Pairs, Pair),
       Pair = Rest_Sim:Rest_Partner,
       concat(Prefix_Partner, Rest_Partner, Partner_0),
       ( lemma_exists_in_dictionary(Partner_0, dwb) ->
```

```
        Partner = Partner_0
      ; concat('[*] ', Partner_0, Partner) ),
      Similarity is (Prefix_Sim * Rest_Sim),
      Pairs = [Similarity:Partner] )
    ; Pairs = '-' ),
  lemma_to_lexer_tuple(Lemma, Tuple),
  alignment_tuple_insert_maximum(Tuple, Pairs).
```

`alignment_found/4` prüft nun, ob für das Restwort des Lemmas ein Alignment gefunden wurde. Ist dies nicht der Fall, wäre die Variable `Rest_Pairs` eine leere Liste und das Lemma würde mit `NULL`-Werten für Alignmentergebniss und Wahrscheinlichkeit in die Datenbank gespeichert.

Wurde für das Restwort ein Alignment gefunden, so wird nun das Alignmentergebniss für das Restwort an das Ergebnis des Präfixes gehängt. Außerdem wird überprüft, ob das Gesamtergebnis ein Lemma im DWB (also dem Grimm) ist und falls nicht, wird es mit einem `[*]` gekennzeichnet. Die Similarities (also die Wahrscheinlichkeiten) der Alignments für Präfix und Restwort werden multipliziert um die Gesamtwahrscheinlichkeit zu erhalten. Das so gefundene Ergebnis wird dann in die Alignmentdatenbank gespeichert.

An zwei Beispielen soll der Weg der Wortzerlegung deutlich gemacht werden:

```
Lemma = tûsentblat
direkt alignierbar: nein

längster schon alignierter Präfix:
   tûsent -> tausend (4,73%)

Restwort: blat
Alignierung:
   blat -> blatt (4,98%)

Verknüpfung: tausend + blatt -> tausendblatt
   0,0473 (4,73%) * 0,0498 (4,98%) -> 0,0024 (0,24%)

tausendblatt Lemma im Grimm: ja

-> speichern

     ********************************************

Lemma = tûsentvalten
direkt alignierbar: nein
```

```
längster schon alignierter Präfix:
   tûsent -> tausend (4,73%)

Restwort: valten
Alignierung:
   valten -> falten (25,93%)

Verknüpfung: tausend + falten -> tausendfalten
   0,0473 (4,73%) * 0,2593 (25,93%) -> 0,0123 (1,23%)

tausendblatt Lemma im Grimm: nein
   -> markieren mit [*]

-> speichern
```

5.4.2 Testergebnisse

Tests haben gezeigt, dass die Zahl der gefundenen Alignments durch diese Methode stark erhöht werden konnte. Tabelle 5.5 soll einen Eindruck davon vermitteln.

MHD	**Original**	**Wortzerlegung**	**Wahrscheinlichkeit**
tûsent	*tausend*	*tausend*	4,73%
tûsentblat	NULL	*tausendblatt*	0,24%
tûsentkunstiger	NULL	*tausendkünstiger*	00,07%
tûsentleie	NULL	*[*]tausendleie*	03,07%
tûsentlisteler	NULL	NULL	NULL
tûsentlistic	*tausendlistig*	*tausendlistig*	01,17%
tûsentste	*tausendste*	*tausendste*	02,31%
tûsentvalt	*tausendfalt*	*tausendfalt*	01,82%
tûsentvalten	NULL	*[*]tausendfalten*	01,23%
tûsentvaltic	NULL	*tausendfaltig*	00,85%
tûsentveltic	NULL	*[*]tausendfeldig*	00,12%
tûsentvalticlîche	NULL	*[*]tausendfaltigleiche*	00,37%
tûsentvalticlîchen	NULL	*[*]tausendfaltigleichen*	00,34%
tûsentwarbe	NULL	*[*]tausendwarbe*	02,42%
tûsentwarp	NULL	*[*]tausendwarp*	02,22%
tûsentzal	NULL	*tausendzahl*	00,17%

Tabelle 5.5: Vergleich der Ergebnisse für den mhd. Präfix *tûsent*

Die Tabelle zeigt den Unterschied in der Ergebnisanzahl für den mittelhochdeutschen Präfix *tûsent* (nhd. *tausend*). Mit diesem Präfix verzeichnet

der Lexer 16 Einträge, die in der Tabelle alle aufgeführt sind. Das Originalprogramm konnte für diese 16 Einträge nur vier Ergebnisse erzielen. Mit Hilfe der Wortzerlegung konnte diese Zahl auf 15 Ergebnisse erhöht werden.

Wie man in der Tabelle in der Spalte für die Wahrscheinlichkeiten sieht, wurden einige Alignments gefunden, deren Wahrscheinlichkeit unter 1% liegt. Dies ist bemerkenswert, da die feste Schranke des Alignmentprogramms bei 1% liegt und jedes Alignment, welches diesen Wert unterschreitet, abgebrochen wird. Diese Ergebnisse haben nur auf Grund ihrer Wortlänge eine so niedrige Wahrscheinlichkeit, denn wie schon mehrfach beschrieben multiplizieren sich die Übergangswahrscheilichkeiten der einzelnen Zeichen zur Gesamtwahrscheinlichkeit des Alignments auf. Die Wortzerlegung ist also zusätzlich eine Methode, die feste Schranke indirekt „wortlängensensibel“ zu machen, so dass auch lange Wörter eine Chance haben aligniert zu werden. Dies könnte man allerdings auch direkter erreichen, indem man die Schranke in Abhängigkeit zur Wortlänge festlegt.

Die mit einem *[*]* gekennzeichneten Ergebnisse sind im Grimm nicht als Lemmata aufgeführt. Im Gegensatz zum originalen Programm werden hier keine Lemmata sondern ihre Teilwörter mit den Lemmata des Grimms verglichen. Durch das Zusammenfügen der gefundenen Teil-Alignments kann so also ein Ergebnis entstehen, welches kein Lemma des Grimms ist. Deshalb prüft das Programm, ob das gefundene Alignment Lemma im Grimm ist. Wenn nicht, wird dem Ergebnis *[*]* voran gesetzt.

Weiterhin ist festzustellen, dass für die drei Wörter *tûsentveltic*, *tûsentvalticlîche* und *tûsentvalticlîchen* falsche Ergebnisse gefunden wurden. Dies ist einfach zu erklären: Bei *tûsentveltic* ist der größte lexikalische Präfix zu seinem Vorgänger *tûsent*. Somit aligniert das Programm das Restwort *veltic* mit dem Grimm'schen Lemmata *feldig*. Bei *tûsentvalticlîche* und *tûsentvalticlîchen* bleibt nach Abspaltung des größten lexikalischen Präfixes *lîche*, bzw. *lîchen* stehen, welches richtigerweise auf die Lemmata *Leiche* und *Leichen* aligniert wird.

Ein weiterer Testlauf über den Präfix *abe-* (nhd. *ab-*), mit dem im Lexer 327 Lemmata beginnen, zeigt die Wirksamkeit der Wortzerlegung noch deutlicher. Das Originalprogramm konnte 143 der mit *abe* beginnenden mittelhochdeutschen Lemmata alignieren. Mit Hilfe der Wortzerlegung stieg diese Zahl auf 323. Somit blieben nur 4 Lemmata mit diesem Präfix ohne Alignment.

Diese Zahlen zeigen, dass es sich sehr lohnt die Wortzerlegung nach Präfix und Restwort in das Alignment-Tool aufzunehmen. Wie eingangs erwähnt, zielt die Implementierung der Wortzerlegung eher auf das Alignment von Komposita ab. Deswegen sind die Ergebnisse der *abe*-Testreihe so erstaunlich, da sich kaum Komposita mit diesem Präfix finden lassen.

In ihrer jetzigen Form ist die Wortzerlegung eher als Zusatz zum Alignment-Tool zu sehen. Wie schon beschrieben, wird für die gefundenen Teilwörter geprüft, ob sie schon aligniert sind und in der Datenbank stehen. Natürlich werden nicht vorhandene Teilwörter aligniert, aber schnell und effizient arbeitet die Wortzerlegung nur, wenn schon ein Datenbestand an alignierten Wörtern zur Verfügung steht. Alle Tests wurden deshalb auch mit den schon bestehenden Alignments durchgeführt.

Abschließend lässt sich sagen, dass die Wortzerlegung noch weiterer Verbesserungen bedarf um die Anzahl fehlerhafter Ergebnisse zu verringern, doch schon jetzt ist die Zahl der zusätzlich gefunden Alignments bemerkenswert.

5.5 Kombination der Änderungen: Testergebnisse

Im folgenden Kapitel werden die Ergebnisse der kombinierten Wirkung der Änderungen des Alignment-Tools vorgestellt. Die Wortzerlegung wurde hierbei jedoch außen vor gelassen, da sie in der jetzigen Form eher als Zusatz zum Alignment-Tool zu sehen ist. Die im Weiteren vorgestellten Ergebnisse zeigen also die Kombination der Änderungen an dem Basiskorpus, der Substitutionstabelle und den Transitionen.

Tabelle 5.6 zeigt die Daten, die die Testläufe ergeben haben.

Präfix	**Ergebnisse**	**Zeit** (hh:mm:ss)	**Wahrscheinlichkeit**
j	274 (+34,31%)	00:59:57 (+17,97%)	21,71% (+8,59%)
o	430 (+21,13%)	01:05:34 (+10,35%)	39,97% (+16,90%)
v	5770 (+49,17%)	22:40:52 (+64,08%)	37,20% (+24,17%)
Gesamt	**+46,24%**	**+58,19%**	**+22,89%**

Tabelle 5.6: Ergebnisse der kombinierten Neuerungen

Wie man sehen kann, arbeitet alle drei Änderungen des Tools gut Hand in Hand. Sowohl die Überarbeitung des Basiskorpus als auch die Aufnahme von zusätzlichen Buchstabengruppen in die Substitutionstabelle verbessern die Anzahl der gefundenen Alignments und deren Durchschnittswahrscheinlichkeit. Die Testergebnisse haben ebenfalls gezeigt, dass die Einführung von zusätzlichen Buchstabengruppen in die Substitutionstabelle diesbezüglich wesentlich größeren Nutzen zeigt, als die Überarbeitung des Basiskorpus. Die XY-Transition vermindert erheblich die negativen Auswirkungen auf die Laufzeit, die durch die Änderung an der Substitutionstabelle entstanden sind.

Insgesamt wurde die Anzahl der gefunden Alignments um über 2.000 Ergebnisse, also 46,24%, erhöht. Die Durchschnittswahrscheinlichkeit der Ergebnisse erhöhte sich um 22,89%. Die Laufzeit erhöhte sich um 58,19%. Daran sieht man deutlich, wie nützlich die XY-Transition in Verbindung mit den Lautgruppen ist. Sie konnte eine Laufzeiterhöhung um den Faktor 4 auf eine Erhöhung um den Faktor 1,6 reduzieren.

Die Abbildung 5.1 stellt diese Ergebnisse noch einmal übersichtlich in einem Säulendiagramm dar. Auf der y-Achse wurde für eine bessere Darstellung ein Schnitt bei 60% gemacht und das Diagramm an dieser Stelle zusammengestaucht.

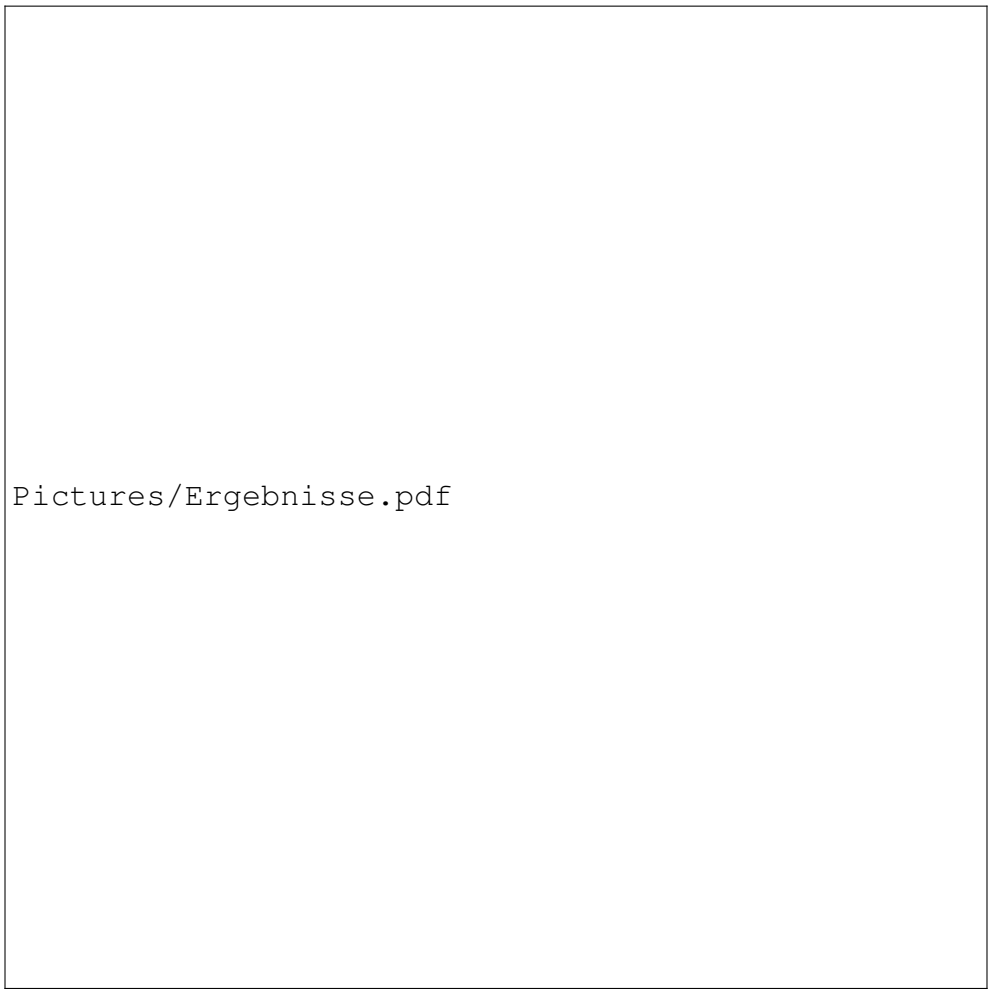


Abbildung 5.1: Grafische Darstellung der Ergebnisse

5.6 Zusammenfassung und Auswertung

Es hat sich gezeigt, dass auf technischer Seite nur wenig Optimierungsbedarf besteht. Da das Alignment-Tool jedoch im Zuge eines interdisziplinären Projekts entstanden ist und linguistische Prozesse analysieren, bzw. dokumentieren soll, lassen sich auf linguistischer Seite Wege finden, das Alignment zu verbessern. Diese linguistisch begründeten Änderung können dann wiederum auf technischer Seite zu Optimierungsbedarf führen.

Ein gutes Beispiel hierfür ist die Aufnahme der Lautgruppen. Wie die Abbildung 5.1 zeigt, wäre sie allein betrachtet keine sinnvolle Erweiterung des Alignment-Tools, da die Erhöhung der Rechendauer schlicht viel zu gravierend ist. Erst in Verbindung mit der XY-Transition ergibt diese Änderung Sinn. Die Transitionen wiederum haben für sich allein gesehen keine große Auswirkung auf das Tool. Ein an sich schon sehr effizientes Programm wird um einen geringen Faktor verbessert. Doch sie verringern die Erhöhung der Rechendauer, die durch die Lautgruppen ausgelöst wurde von Faktor 4 auf Faktor 1,6.

Die Änderungen am Basiskorpus hingegen zeigen kaum eine Auswirkung auf das Alignment-Tool. Da die Überarbeitung des Korpus eine Arbeit ist, die schwer zu automatisieren ist, lohnt der Arbeitsaufwand für die geringe Besserung kaum. Allein den aktuellen Basiskorpus in beschriebener Art zu überarbeiten, dauerte weit über fünf Stunden. Sollte sich allerdings eine Möglichkeit finden, diesen Prozess beispielsweise über reguläre Ausdrücke zu automatisieren, so würde es das Alignment-Tool weiter abrunden.

Auch die Tests mit der Wortzerlegung haben weit bessere Resultate erzielt, als erwartet wurde. Auch wenn einige falsche Alignments gefunden wurden, beeinträchtigt dies in keinster Weise die positiven Ergebnisse der Testläufe. Auch das originalen Alignment-Programm hat eine Fehlerquote und liefert einen kleinen Anteil falscher Ergebnisse. Dies lässt sich nicht vermeiden, wenn man versucht Sprache automatisch zu untersuchen. Wie schon früher erwähnt, ist Sprache kein sich homogen entwickelndes System. Es gibt immer wieder Ausnahmen und Besonderheiten, die sich keiner Regelmäßigkeit unterwerfen lassen. Dementsprechend war schon für das Original-Programm, und somit auch für die im Zuge dieser Arbeit entstandenen Modifikationen, klar, dass es immer nötig sein wird, die Daten zu kontrollieren und fehlerhafte Ergebnisse zu verbessern.

Mit diesem Wissen im Hinterkopf kann man feststellen, dass die Wortzerlegung und die Speicherung weiterer Lautgruppen in die Substitutionstabelle die wirkungsvollsten Änderung am Programm sind. Durch sie lässt sich die Ergebnisanzahl stark erhöhen und dies ist schließlich das Ziel des Alignment-Programms.

Kapitel 6

Übersetzungen und Verweise

Es ist sinnvoll, aus dem vorhanden Datenbestand alle Informationen auszulesen, die für das Projekt hilfreich sein können. So stehen in beiden zu Grunde liegenden Wörterbüchern teilweise Übersetzungen. Diese auszulesen kann den Basiskorpus und auch das Alignment verbessern, in dem man sie in den Datenbestand aufnimmt. Ebenso kann es sehr hilfreich sein Verweise zu finden und auszulesen. Teilweise stehen in den Wörterbüchern Dialektformen eines Wortes. Findet man diese, so reicht es eins der Worte zu alignieren, da alle nur Formen des selben Wortes sind.

Auch wenn von mir an der Realisierung dieser Idee nur mitgeholfen wurde, soll sie der Vollständigkeit halber kurz erwähnt werden, da sie Teil der Diplomarbeit ist.

6.1 Direkte Übersetzung aus den Wörterbüchern

Bei einigen Lemmata, sowohl im Lexer als auch im Grimm, kann man eine direkte Übersetzung mdh $\rightarrow$ nhd, bzw. nhd $\rightarrow$ mdh, finden. Also sucht man nach Eigenschaften, die es ermöglichen, die Übersetzungen (falls vorhanden) zu erkennen und automatisiert auszulesen und zu speichern.

Lexer $\rightarrow$ Neuhochdeutsch

Im Lexer sind teilweise direkte Übersetzungen vom Mittelhochdeutschen ins Neuhochdeutsche vorhanden. Es ist allerdings schwierig diese automatisiert auszulesen, da es keine klare, einheitliche Struktur gibt. Dies wird im Beispiel 6.1 deutlich.

Man sieht in dem Beispiel, dass es keinen Marker oder eine Art Schlagwort gibt, die anzeigen, dass eine neuhochdeutsche Übersetzung folgt. Ohne einen hohen Rechenaufwand durch anzustellende Vergleiche kann man

1. vluoch - *stm.* (BMZ ib.) *verwünschung, -fluchung, fluch* DIEM. A.HEINR. WALTH. MS. [...]

2. vluor - *stm.* (BMZ III. 356a) *stf.* in statvluor; *md.* vlûr (flûre, flôre MICH. M. HOF 43) —: *flur, feldflur, saatfeld*

3. adel-ar - *swm.* (BMZ I. 49a) *syncop.* adlar. adler *(auch stm.) edler ar, adler*

Beispiel 6.1: Übersetzungen im Lexer

kaum herausfinden, an welcher Stelle des Wörterbucheintrags eine Übersetzung steht.

Dieser Weg Übersetzungen auszulesen wurde dann auch nicht verfolgt, da nicht in jedem Eintrag Übersetzungen vorhanden sind. Die zu erzielenden Ergebnisse wären also den Aufwand nur bedingt wert.

DWB → Mittelhochdeutsch

Im Grimm ist es möglich, direkte Übersetzungen vom Neuhochdeutschen ins Mittelhochdeutsche auszulesen, da die mittelhochdeutschen Übersetzungen oft mit einem vorgestellten *'mhd.'* gekennzeichnet sind.
Allerdings treten auch hier Unregelmäßigkeiten auf, wie man im Beispiel 6.2 sehen kann.

1. fluch, *m. exsecratio, imprecatio, maledictum, ahd.* fluoch, *mhd.* vluoch, [...]

2. fluchen, *maledicere, imprecari, ein wort, dessen form noch nicht gehörig auseinander gesetzt ist. zwar herscht schon ahd. die schwache vor* fluochôn fluochôta, *und mhd. nhd. gilt keine andere*, vluochen, vluochete, fluchen fluchte.

Beispiel 6.2: Übersetzungen im DWB (Grimm)

Luise Borek von der TU Darmstadt, die ebenfalls an dem Projekt „Sprache und Genom“ [4] mitgewirkt hat, und Prof. Dr. Dietmar Seipel haben diesen Weg verfolgt um Übersetzungen aus dem Grimm zu ziehen und somit den Korpus gefundener Wortpaare zu erweitern.

6.2 Verweise im Wörterbuchbestand finden

Im Mittelhochdeutschen, wie auch im Neuhochdeutschen, gab es verschiedene Dialekte. Hat der Lexer mehrere Dialektformen für ein Wort aufgenom-

men, so steht bei der Dialektform oft ein Verweis auf das Standardwort (siehe Beispiel 6.3).

1. brucke brücke brügge - *stswf.* (BMZ I. 266a) *brücke, allgem. s. noch* brucke und stege [...]
2. brugge - *f.s.*LEXER *brücke*.
3. brüge - *f.s.*LEXER *brücke*.

Beispiel 6.3: Verweise im Lexer

Findet man diese Verweise, kann man das Alignment verbessern und beschleunigen, da das Standardlemma eventuell schon aligniert ist, bzw. leichter zu alignieren ist. Weiterhin könnte man auch überlegen die Alignmentdatenbank so anzupassen, dass zu jedem Wort ein Verweis gespeichert werden kann. Somit könnte man, sollte in dem Datenbestand ein Verweis gefunden werden, diesen direkt in die Datenbank speichern und das Dialektwort nicht alignieren. In der Datenbank würde also das Dialektwort auf das Standardwort verweisen, welches wiederum aligniert ist und mit einem neuhochdeutschen Ergebnispartner in die Datenbank gespeichert wurde.

Kapitel 7

Ausblick

Bisher konnten durch die in Kapitel 4 vorgestellten Änderungen von Basiskorpus, Substitutionstabelle, Wortzerlegung und der XY-Transition einige positive Auswirkungen auf das Alignment-Tool erzielt werden. Die Ergebnisse in Kapitel 5 belegen dies.

Im Zuge dieser Diplomarbeit sind noch weitere Ideen zur Verbesserung des Alignments und zur Beschleunigung des Alignmentprozesses entstanden, die zukünftig umgesetzt werden können. Zum Zeitpunkt der Fertigstellung dieser Diplomarbeit waren die dazu erforderlichen Daten aber noch nicht vorhanden. Diese Ideen sollen im folgenden Kapitel vorgestellt werden. Auch diese folgenden Ansätze kann man in einen linguistischen Ansatz (Wortklassen), und einen technischen Ansatz (Schrittweite und Schranke) aufteilen.

7.1 Wortklassen

Die Datenbanken, die dem Alignmentprogramm die Lemmata des Grimm'schen Wörterbuchs und des Lexers zur Verfügung stellen, beinhalten diese Daten ohne weitere Eigenschaften zu den Lemmata. In beiden Wörterbüchern sind aber die Wortklassen, also Substantiv, Verb, Adjektiv, usw., der einzelnen Einträge zu finden. Für eine weitere Beschleunigung des Alignments könnte man diese auslesen, speichern und verwenden.

Kennt man die Wortklasse eines Lemmas, kann man die Vergleiche, die für dessen Alignierung durchgeführt werden müssen, weiter einschränken. Bisher wird diese Einschränkung lediglich auf Grund der Substitutionstabelle und somit der Buchstaben und Laute betrieben. Ein Substantiv muss aber nicht mit ein Verb oder Adjektiv verglichen werden, da das korrekte Alignment wieder ein Substantiv sein wird. Können also beim digitalen Auslesen der Wörterbücher die Marker für die Wortklasse ebenfalls gespeichert werden, kann man die Vergleiche auch unter dem Aspekt der Wortklassen einschränken.

7.2 Schrittweite

Die Schrittweite bezeichnet die Anzahl von Buchstaben, die beim Alignieren eines Wortes betrachtet werden können. Dies ist nützlich, da man so schneller zum Ergebnis, also Erfolg oder Abbruch der Alignierung, gelangt. Vor allem im Hinblick auf die vorhanden Buchstabengruppen in der Substitutionstabelle ist dies sinnvoll (z.B. *ou, ck, sch, ei, ...*). Es birgt aber auch seine Nachteile. Durch das Betrachten mehrerer Buchstaben, wird die erforderliche Rechenleistung erhöht.

Bisher ist es schon möglich die Schrittweite beliebig zu wählen. Da aber noch nicht untersucht wurde, welcher Gewinn daraus zu ziehen ist, wurde der Wert der Schrittweite standardmäßig auf `[step:4]` gesetzt. Zu testen ist nun, ob eine variable Schrittweite die Effizienz den Programms erhöht.

Gestartet werden sollte dabei möglichst mit einer kleinen Schrittweite von eins oder zwei, da diese weniger Berechnungen zum Durchlauf eines Wortes benötigen und somit schneller sind. Findet sich ein Ergebnis, so wird der Ausgangswert der Schranke auf dessen Wahrscheinlichkeit erhöht. Falls nicht, bleibt die Schranke unverändert. Danach wird die Schrittweite auf einen Maximalwert erhöht, welcher sich an der Länge der Buchstabengruppen in der Substitutionstabelle messen sollte. Man sollte auch nachprüfen, ob eine schrittweise Erhöhung der Schrittweite sinvoll ist.

Somit müssen die Vergelichs-Lemmata zwar öfter durchlaufen werden, doch da man die Schranke im Idealfall sehr schnell weit erhöht hat, werden uninteressante Berechnungen auch weit schneller abgebrochen.

7.3 Schranke

Die Schranke ist die Abbruchbedingung des Alignmentprogramms. Unterschreitet eine Alignmentwahrscheinlichkeit den Grenzwert, den die Schranke setzt, so wird das Alignment abgebrochen. Bisher ist die Schranke mit einem Wert von 0,01, also 1% fest im Programm implementiert.

Hat man für ein Wort eine passende Alignierung gefunden, die oberhalb der Schranke liegt, so gibt die Wahrscheinlichkeit dieser Alignierung den neuen Grenzwert. Auf diese Art und Weise können weitere Berechnungen noch schneller abgebrochen werden. Nur bessere Ergebnisse können berechnet werden, was jedes Mal wiederum den Grenzwert erhöht.

Grenzen	Anzahl	Anteil
0,0 - 0,1	9.745	0,22
0,1 - 0,2	2.587	0,06
0,2 - 0,3	2.867	0,06
0,3 - 0,4	3.049	0,07
0,4 - 0,5	2.888	0,06
0,5 - 0,6	22.830	0,50
0,6 - 0,7	680	0,02
0,7 - 0,8	403	0,01
0,8 - 0,9	157	0,00
0,9 - 1,0	27	0,00
Gesamt:	45.233	1,00

Tabelle 7.1: Wahrscheinlichkeitsverteilung der bisher alignierten Wörter

Die neue Idee ist nun, nicht mit einer niedrigen Schranke zu starten und diese zu erhöhen, sondern sich anzuschauen, in welchem Wahrscheinlichkeitsbereich die meisten bisher alignierten Wortpaare liegen und diesen Wert als Startschranke zu nehmen. Sollte man keine Ergebnisse finden, so kann die Schranke gesenkt werden.

Diese Methode soll, ähnlich wie bei der Schrittweite, Rechenzeit ersparen, da unwahrscheinliche Alignierungsversuche schnell abgebrochen werden und man somit schnell zu einem wahrscheinlichen Ergebnis kommt, sollte es eins geben.

Zu diesem Zweck wurde in Tabelle 7.1 eine Wahrscheinlichkeitsverteilung aller bisher alignierten Wörter aufgestellt. Diese zeigt, dass sich ca. 50% der Wörter innerhalb der Wahrscheinlichkeitsgrenzen von 50% und 60% befinden. Wird diese Idee also umgesetzt, so wäre ein Wert von 0,5 ein guter Anfangswert für die Schranke.

Code

In diesem Kapitel wird der Code gezeigt, der die Änderungen aus Kapitel 4 umsetzt. Prädikate, die aufgerufen werden, aber im Folgenden nicht beschrieben sind, gehören zum originalen Code von Prof. Dr. Dietmar Seipel und wurden nicht geändert.

alignment_sub_create_table

Der Code zum Kapitel 5.2.1:

Mit dem Hauptprädikat `alignment_sub_create_table/1` wird eine .csv-Datei eingelesen, die den Basiskorpus (siehe 3.2) beinhaltet. Danach erstellt man die Liste der Zeichen, wobei die dynamische Variable der Zeichenlänge (`signlength`) beachtet wird. Über diese kann man festlegen, dass die auszulesenden Zeichen eine Länge von 1 bis `signlength` haben sollen. Sind diese Zeichen gesammelt, werden ihre Frequenzen (also die Übergangswahrscheinlichkeiten), berechnet.

Das zweite Hauptprädikat `alignment_sub_save_table/1` speichert alle gefundenen Zeichenübergänge in eine .csv-Datei zur weiteren Verarbeitung. Über das `save_all/0`-Prädikat aus der Datei `alignment_find_frequency.pl` können alle in einer .csv-Datei gespeicherten Zeichenübergänge ungefiltert in eine SQL-Datenbank eingespeist werden.

```
/*****************************************************/
/***                                               ***/
/***           Alignment:  Create Subtable         ***/
/***                                               ***/
/*****************************************************/



/*** tests *******************************************/
```

```
test(alignment, subtable) :-
   dislog_variable_set(signlength, 3),
   dislog_variable_get(home,
      '/sources/projects/linguistics/
       haubner/test.csv', Input),
   dislog_variable_set(input, Input),
   alignment_sub_create_table(Input),
   dislog_variable_get(home,
      '/sources/projects/linguistics/
       haubner/test_out.csv', Output),
   dislog_variable_set(output, Output),
   alignment_sub_save_table(Output).


/*** interface ****************************************/


alignment_sub_create_table(Input) :-
   send(@align_submatrix, clear),
   csv_read_file(Input, Rows, [separator(59)]),
   maplist(alignpro_read_alignment, Rows, Lems),
   alignment_sub_get_signs_from_lemmas(
      Lems, Sgns_A, Sgns_B),
   alignment_sub_calculate_frequencies(
      Sgns_A, Sgns_B, Lems).

alignment_sub_save_table(Output) :-
   alignsub_get_signs_from_table(Sgns_A, Sgns_B0),
   delete(Sgns_B0, [], Sgns_B),
   findall( row(Sgn_A, Sgn_B, Freq, Count_A),
      ( member(Sgn_A, Sgns_A),
        member(Sgn_B, Sgns_B),
        alignlib_get_frequency(Sgn_A, Sgn_B, Freq),
        Freq > 0,
        alignment_sub_get_count(Sgn_A, Count_A),
        Count_A > 1 ),
      Rows ),
   csv_write_file(Output, Rows, [separator(59)]).


/*** implementation ***********************************/
```

```
/* alignment_sub_get_signs_from_lemmas(
      Lems, Sgns_A1, Sgns_B1) <-
      */

alignment_sub_get_signs_from_lemmas(
      Lems, Sgns_A1, Sgns_B1) :-
   alignment_sub_collect_signs_from_lemmas(
      Lems, Sgns_A0, Sgns_B0),
   sort(Sgns_A0, Sgns_A1),
   sort(Sgns_B0, Sgns_B1).

alignment_sub_collect_signs_from_lemmas(
      [Lem|Lems], Sgns_A1, Sgns_B1) :-
   !,
   dislog_variable_get(signlength, Len),
   Lem = [Lem_A, Lem_B],
   codes_to_substrings(Len, Lem_A, Sgns_A),
   codes_to_substrings(Len, Lem_B, Sgns_B),
   alignment_sub_collect_signs_from_lemmas(
      Lems, Sgns_A0, Sgns_B0),
   append(Sgns_A, Sgns_A0, Sgns_A1),
   append(Sgns_B, Sgns_B0, Sgns_B1).
alignment_sub_collect_signs_from_lemmas([], [], []).


/* codes_to_substrings(Length, Codes, Strings) <-
      */

codes_to_substrings(Length, Codes, Strings) :-
   findall( String,
      ( middle_sequence(Codes, Xs),
        length(Xs, L),
        between(1, Length, L),
        atomic_list_concat(Xs, String_0),
        delete_percents(String_0, String) ),
      Strings ).

delete_percents(Name_1, Name_2) :-
   name_exchange_sublist([["%",""]], Name_1, Name),
   ( Name = '' ->
     Name_2 = '%'
   ; Name_2 = Name ).


/* alignment_sub_calculate_frequencies(
```

```
      Sgns_A, Sgns_B, Lems_0) <-
      */

alignment_sub_calculate_frequencies(
      Sgns_A, Sgns_B, Lems_0) :-
   send(@align_counter, clear),
   include(alignlib_test_length, Lems_0, Lems_1),
   maplist(alignment_sub_count_in_lemma, Lems_1),
   forall( member(Sgn_A, Sgns_A),
      ( alignlib_count(point(Sgn_A, null), 0, NSgnA_),
        alignment_sub_set_count(Sgn_A, NSgnA_),
        forall( member(Sgn_B, Sgns_B),
           ( alignlib_count(
                point(Sgn_A, Sgn_B), 0, NSgnASgnB),
             ( NSgnASgnB == 0 -> true
             ; Freq is NSgnASgnB/NSgnA_,
               alignlib_set_frequency(
                  Sgn_A, Sgn_B, Freq)  ) ) ) ) ),
   send(@align_counter, clear).

alignment_sub_count_in_lemma(
      [[Lem_A0|Lems_A], [Lem_B0|Lems_B]]) :-
   dislog_variable_get(signlength, Len),
   codes_to_substrings(Len, [Lem_A0|Lems_A], Prefixes_A),
   codes_to_substrings(Len, [Lem_B0|Lems_B], Prefixes_B),
   forall( nth1(N, Prefixes_A, Sgn_A),
      ( nth1(N, Prefixes_B, Sgn_B),
        alignlib_count(point(Sgn_A, null), 1, _),
        alignlib_count(point(Sgn_A, Sgn_B), 1, _) ) ).

alignment_sub_set_count(Sgn, Count) :-
   atom_number(Atm_Count, Count),
   alignlib_set(
      @align_submatrix, point(Sgn, []), Atm_Count).

alignment_sub_get_count(Sgn, Count) :-
   ( alignlib_get(
        @align_submatrix, point(Sgn, []), Atm_Count) ->
     atom_number(Atm_Count, Count)
   ; Count = 0 ).



/*****************************************************/
```

alignment_find_frequency

Die durch die Prädikate der `alignment_create_subtable.pl` erstellten .csv-Datei können mittels der Prädikate in dieser Datei gefiltert werden. Die gefilterten Ergebnisse können mittels der Pradikate `save_all/0` und `alignment_save_frequencies/3` gespeichert werden.

Die Datei `alignment_find_frequency.pl` bietet folgende Filtermöglichkeiten (Anmerkung: Das Wort 'Affix' steht hier für eine Buchstabengruppe aus einem oder mehreren Buchstaben):

- alle Übergänge für eine mittelhochdeutsche Affix-Liste
- alle Übergänge für eine neuhochdeutsche Affix-Liste
- den Übergang, bzw. die Frequenz für ein bestimmtes Affix-Paar finden
- alle Übergänge für einen mittelhochdeutschen Affix finden
- alle Übergänge für einen neuhochdeutschen Affix finden
- nur häufige Übergänge finden (Übergabe einer Mindesthäufigkeit)
- nur hochfrequente Übergänge finden (Übergabe einer Mindestfrequenz)

```
/*******************************************************/
/***                                                 ***/
/***          Alignment:  Find Frequencies           ***/
/***                                                 ***/
/*******************************************************/



:- dislog_variable_get(home,
      '/sources/projects/linguistics/haubner/
       alignment_subtable.csv', Output),
   dislog_variable_set(output, Output).


/*** implementation save_all ***************************/


save_all :-
   send(@align_submatrix, clear),
```

```
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   forall( (member(row(Sgn_A, Sgn_B, Freq, _), Rows),
            Freq > 0),
       alignment_save_frequencies(Sgn_A, Sgn_B, Freq) ).

alignment_save_frequencies(Sgn_A, Sgn_B, Freq) :-
   atom_number(Atm_Freq, Freq),
   atomic_list_concat( [ 'INSERT INTO ',
     'Subtable(SgnA, SgnB, Freq) VALUES (\'',
     Sgn_A, '\', \'', Sgn_B, '\', ', Atm_Freq, ');' ],
     Save_Stm ),
   odbc_query(mysql, 'USE Alignment'),
   odbc_query(mysql, Save_Stm).


/*** implementations filter ***************************/


/* alignment_find_frequencies_mhdlist(
      Mhd_Lists, Freqs) <-
      */

alignment_find_frequencies_mhdlist(Mhd_Lists, Freqs) :-
   alignment_find_frequencies_mhdlist(
      Mhd_Lists, Freqs, 0, 0).

alignment_find_frequencies_mhdlist(
      [Sgn_A|Mhd_Lists], Freqs, Min_Count, Min_Freq) :-
   send(@align_submatrix, clear),
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   findall( [Sgn_A, Sgn_B, Freq],
      ( alignment_find_frequencies_mhd(Sgn_A, F),
        member([Sgn_A, Sgn_B, Freq], F),
        member(row(Sgn_A, Sgn_B, Freq, Count), Rows),
        Min_Count =< Count,
        Min_Freq =< Freq ),
      Freqs_1),
   alignment_find_frequencies_mhdlist(
      Mhd_Lists, Freqs_0, Min_Count, Min_Freq),
   append(Freqs_1, Freqs_0, Freqs).
alignment_find_frequencies_mhdlist([], [], _, _).
```

```
/* alignment_find_frequencies_nhdlist(
      Nhd_Lists, Freqs) <-
      */

alignment_find_frequencies_nhdlist(Nhd_Lists, Freqs) :-
   alignment_find_frequencies_nhdlist(
      Nhd_Lists, Freqs, 0, 0).

alignment_find_frequencies_nhdlist(
      [Sgn_B|Nhd_Lists], Freqs, Min_Count, Min_Freq) :-
   send(@align_submatrix, clear),
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   findall( [Sgn_A, Sgn_B, Freq],
      ( alignment_find_frequencies_nhd(Sgn_B, F),
        member([Sgn_A, Sgn_B, Freq], F),
        member(row(Sgn_A, Sgn_B, Freq, Count), Rows),
        Min_Count =< Count,
        Min_Freq =< Freq ),
      Freqs_1),
   alignment_find_frequencies_nhdlist(
      Nhd_Lists, Freqs_0, Min_Count, Min_Freq),
   append(Freqs_1, Freqs_0, Freqs).
alignment_find_frequencies_nhdlist([], [], _, _).


/* find specific frequencies:
      alignment_find_frequency(Sgn_A, Sgn_B, Freq) <-
      alignment_find_frequencies_mhd(Sgn_A, Freq) <-
      alignment_find_frequencies_nhd(Sgn_B, Freq) <-
      */

alignment_find_frequency(Sgn_A, Sgn_B, Freq):-
   send(@align_submatrix, clear),
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   forall( member(row(Sgn_A, Sgn_B, Freq_1, _), Rows),
      alignlib_set_frequency(Sgn_A, Sgn_B, Freq_1) ),
   alignlib_get_frequency(Sgn_A, Sgn_B, Freq).

alignment_find_frequencies_mhd(Sgn_A, Freq) :-
   send(@align_submatrix, clear),
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   findall( Sgn_B,
```

```
      member(row(Sgn_A, Sgn_B, _, _), Rows),
     Sgns_B ),
   findall( [Sgn_A, Sgn_B, Freq],
      ( member(Sgn_B, Sgns_B),
        member(row(Sgn_A, Sgn_B, Freq, _), Rows),
        Freq > 0 ),
      Freq ).

alignment_find_frequencies_nhd(Sgn_B, Freq) :-
   send(@align_submatrix, clear),
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   findall( Sgn_A,
      member(row(Sgn_A, Sgn_B, _, _), Rows),
     Sgns_A ),
   findall( [Sgn_A, Sgn_B, Freq],
      ( member(Sgn_A, Sgns_A),
        member(row(Sgn_A, Sgn_B, Freq, _), Rows),
        Freq > 0 ),
      Freq ).


/* filter accortding to commonness and frequentness:
      alignment_find_common_transition(
         Min_Count, Transitions <-
      alignment_find_frequent_transition(
         Min_Freq, Transitions) <-
      */

alignment_find_common_transition(
      Min_Count, Transitions):-
   send(@align_submatrix, clear),
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   findall( [Sgn_A, Sgn_B, Freq],
        ( member(row(Sgn_A, Sgn_B, Freq, Count), Rows),
          Min_Count =< Count ),
        Transitions ).

alignment_find_frequent_transition(
      Min_Freq, Transitions) :-
   send(@align_submatrix, clear),
   dislog_variable_get(output, Output),
   csv_read_file(Output, Rows, [separator(59)]),
   findall( [Sgn_A, Sgn_B, Freq],
```

```
        ( member(row(Sgn_A, Sgn_B, Freq, _), Rows),
          Min_Freq =< Freq ),
        Transitions ).



/*****************************************************/
```

XY-Transition

Die Code-Änderungen für Kapitel 5.3:

Um die bisher im Programm aktive Transition durch die in Kapitel 4.2.1 beschriebene XY-Transition zu ersetzten, müssen die folgenden drei Stellen im Originalcode geändert werden.

Hierbei geht es darum, dass das Programm, wenn das Alignment initialisiert und die Transitionen geladen werden, die Informationen anders formatiert. Anstatt der alten Form (Beispiel 7.1) soll eine neue Form (Beispiel 7.2) geschaffen und genutzt werden.

```
alignment_transitions( MHD, [
   ([MHD], [NHD_1], Wahrscheinlichkeit_1),
   ([MHD], [NHD_2], Wahrscheinlichkeit_2),
   ([MHD], [NHD_3], Wahrscheinlichkeit_3),
   ... ] ).
```

Beispiel 7.1: Allgemeine Form der X-Transition

```
alignment_transitions( MHD, NHD[
   ([MHD], [NHD], Wahrscheinlichkeit_1) ] ).
...
```

Beispiel 7.2: Allgemeine Form der X-Transition

```
/*****************************************************/
/***                                               ***/
/***        Alignment:  XY-Transition              ***/
/***                                               ***/
/*****************************************************/


/*** Ais der Datei alignment_tables.pl ***************/

:- dynamic
   alignment_point_value/2,
   alignment_symbol_difference/3,
   alignment_transitions/2,
   alignment_transitions/3.


alignment_symbol_differences_to_transitions :-
   retractall(alignment_transitions(_, _, _)),
   ddbase_aggregate( [C, D, list(Z)],
      ( alignment_symbol_difference(X, Y, V),
        atom_chars(X, [C|Cs]),
        ( atom_chars(Y, [D|Ds]) ->
          Z = ([C|Cs], [D|Ds], V)
        ; ( D = '',
            Z = ([C|Cs], [], V) ) ) ),
      Pairs ),
   ( foreach([C, D, Transitions], Pairs) do
        alignment_transitions_sort_by_distance(
           Transitions, Transitions_0),
        assert(alignment_transitions(
           C, D, Transitions_0)) ),
   findall( Z,
      ( alignment_symbol_difference('', Y, V),
        atom_chars(Y, [D|Ds]),
        Z = ([], [D|Ds], V) ),
      TransitionsY ),
   alignment_transitions_sort_by_distance(
      TransitionsY, Transitions_1),
   assert(alignment_transitions('', D, Transitions_1)),
   listing(alignment_transitions/3).
```

```
/*** Aus der Datei alignment_chars_to_similarities.pl */


chars_could_be_aligned_to_char(
      Chars, Char, Min, Distance) :-
   Chars = [C1|_],
   alignment_transitions(C1, Char, Transitions),
   ddbase_aggregate( [min(Dist)],
      ( member((Chars_1, _, Dist), Transitions),
        prefix(Chars_1, Chars),
        Dist < Min ),
      Pairs ),
   member([Distance], Pairs).

dijkstra_edge(
      [Xs_1, Ys_1], [Xs_2, Ys_2], Distance) :-
   Xs_1 = [X|_],
   ( Ys_1 = [Y|_]
     ; Y = '' ),
   alignment_transitions(X, Y, Transitions),
   member((As, Bs, Distance), Transitions),
   alignment_distance_is_admissible(Distance),
   append(As, Xs_2, Xs_1),
   append(Bs, Ys_2, Ys_1).


/*****************************************************/
```

alignment_word_partition

Der Code zum Kapitel 5.4.1:

Die Wortzerlegung wird genutzt, um ein Wort in einen Präfix und ein Restwort aufzuspalten. Zunächst wird eine Liste aller Lemmata des Lexers herausgesucht, die mit dem eingegebenen Präfix beginnen. Dieser Präfix kann ein einzelner Buchstabe sein, eine Buchstabengruppe oder auch ein ganzes Wort.

Nun wird für jedes Lemma geprüft ob es alignierbar ist. Ist das nicht der Fall, wird das Präfix-Alignment, bzw. die Wortzerlegung angewendet. Hierbei wird untersucht, ob ein Wort einen schon alignierten Präfix hat. Ist das der Fall, wird an diesem das Lemma getrennt, so dass nur noch das Restwort aligniert werden muss.

```
/*******************************************************/
/***                                                 ***/
/***         Alignment:  Word Partition              ***/
/***                                                 ***/
/*******************************************************/



/*** interface ******************************************/


word_partition(Prefix) :-
   dislog_variable_set(alignment_mode, dijkstra),
   ( alignment_is_initialized -> true
   ; alignment_initialize ),
   lemma_list_from_prefix(Prefix, Lemmas),
   lemma_alignment(Lemmas).


/*** implementation *************************************/


/* lemma_list_from_prefix(Prefix, Lemmas) <-
      */

lemma_list_from_prefix(Prefix, Lemmas) :-
```

```
   odbc_query(mysql, 'USE Alignment'),
   atomic_list_concat( [ 'SELECT * FROM Lexer ',
      'WHERE Lemma LIKE \'',
      Prefix, '%\';' ], SelectStm ),
   findall( Lemma,
      odbc_query( mysql,
         SelectStm, row(_, _, Lemma) ),
      Lemmas ).

lemma_alignment([Lemma|Lemmas]) :-
   nl,
   lemma_is_alignable(Lemma),
   maximum_prefix_alignment(Lemma),
   lemma_alignment(Lemmas).
lemma_alignment([]).


/* lemma_is_alignable(Lemma) <-
      */

lemma_is_alignable(Lemma) :-
   ( lemma_exists_in_alignments(Lemma, _, _) ->
     true
   ; word_to_similarities_save(Lemma) ),
   !.

word_to_similarities_save(Lemma) :-
   Options = [step:4],
   word_to_similarities_save(Options, Lemma).
word_to_similarities_save(Options, Lemma) :-
   dictionary_to_chars_cached('DWB', Lists),
   lemma_to_lexer_tuple(Lemma, Tuple),
   atom_chars(Lemma, Chars),
   chars_to_similarities_single_measure(
      Options, Chars, Lists, Pairs),
   alignment_tuple_insert_maximum(Tuple, Pairs).


/* maximum_prefix_alignment(Lemma) <-
      */

maximum_prefix_alignment(Lemma) :-
   ( lemma_exists_in_alignments(Lemma, _, _) ->
     true
   ; ( atom_chars(Lemma, Lemma_Atm),
```

```
       findall(Prefix_0,
          ( prefix(Prefix_0_Atm, Lemma_Atm),
            atom_chars(Prefix_0, Prefix_0_Atm) ),
          Prefixes_0),
       reverse(Prefixes_0,Prefixes),
       forall( member(Prefix, Prefixes),
          ( lemma_exists_in_alignments(Prefix, _, _) ->
            find_maximum_prefix_alignment(
               Prefix, Lemma)
          ; alignment_found(
               Lemma, '', 1, []) ) ) ),
   !.

find_maximum_prefix_alignment(Prefix, Lemma) :-
   lemma_exists_in_alignments(
      Prefix, PrePartner, PreSim),
   concat(Prefix, RestWord, Lemma),
   ( lemma_exists_in_alignments(
        RestWord, RestPartner, RestSim) ->
     ( Pairs = [RestSim:RestPartner] ,
       alignment_found(
          Lemma, PrePartner, PreSim, Pairs) )
     ; ( word_to_similarities(
            [step:4], RestWord, Pairs),
         alignment_found(
            Lemma, PrePartner, PreSim, Pairs) ) ).

alignment_found(
   Lemma, Prefix_Partner, Prefix_Sim, Rest_Pairs) :-
   ( Rest_Pairs \= [] ->
     ( last(Rest_Pairs, Pair),
       Pair = Rest_Sim:Rest_Partner,
       concat(Prefix_Partner, Rest_Partner, Partner_0),
       ( lemma_exists_in_dictionary(Partner_0, dwb) ->
         Partner = Partner_0
       ; concat('[*] ', Partner_0, Partner) ),
       Similarity is (Prefix_Sim * Rest_Sim),
       Pairs = [Similarity:Partner] )
     ; Pairs = '-' ),
   lemma_to_lexer_tuple(Lemma, Tuple),
   alignment_tuple_insert_maximum(Tuple, Pairs).


/* lemma_exists_in_alignments(
      Lemma, Partner, Similarity) <-
```

```
      */

lemma_exists_in_alignments(
      Lemma, Partner, Similarity) :-
   odbc_query(mysql, 'USE Alignment'),
   odbc_query( mysql,
        'SELECT * FROM Alignment',
        row(_, _, Lemma, Partner, Similarity0) ),
   ( Similarity0 = '$null$' ->
     false
   ; atom_number(Similarity0, Similarity) ).


lemma_exists_in_dictionary(Lemma, Dictionary) :-
   odbc_query(mysql, 'USE Alignment'),
   atomic_list_concat( [ 'SELECT * FROM ', Dictionary,
      ' WHERE Lemma = \'',Lemma, '\';' ], SelectStm ),
   odbc_query( mysql, SelectStm, row(_, _, Lemma) ).


lemma_to_lexer_tuple(Lemma, Tuple) :-
   odbc_query(mysql, 'USE Alignment', _, []),
   odbc_query(
      mysql, 'SELECT * FROM Lexer',
      row(N, Id, Lemma), []),
   Tuple = [N, Id, Lemma].

/*****************************************************/
```

alignment_probability_distribution

Die folgenden zwei Prädikate erzeugen eine Wahrscheinlichkeitsverteilung der bisher gefunden Alignments mit einer 10%-Unterteilung. Diese Verteilung hat zu den Überlegungen bezüglich der Schranke in Kapitel 7.3 geführt.

```
/*******************************************************/
/***                                                 ***/
/***    Alignment:  Probability Distribution         ***/
/***                                                 ***/
/*******************************************************/



/*** implementation ************************************/


alignment_probability_distribution(Distributions) :-
   odbc_query(mysql, 'USE Alignment'),
   findall(Similarity,
      odbc_query(mysql, 'SELECT Similarity
         FROM Alignment WHERE NR > 1',
         row(Similarity) ),
      Similarities),
   list_remove_elements(
      ['$null$'], Similarities, Similarities_1),
   maplist(
      atom_to_number, Similarities_1, Similarities_2),
   msort(Similarities_2, SimSorts),
   Ranges = [0.1, 0.2, 0.3, 0.4, 0.5,
             0.6, 0.7, 0.8, 0.9, 1],
   find_distribution(SimSorts, Ranges, Distributions).

find_distribution(SimSorts, Ranges, Distribution) :-
   ddbase_aggregate( [Range, length(X), list(X)],
      ( member(Range, Ranges),
        member(X, SimSorts), X =< Range ),
   Distribution ).


/*******************************************************/
```

Literaturverzeichnis

[1] Matthias Lexer: *„Mittelhochdeutsches Handwörterbuch“*; 3 Bände; Leipzig 1872–1878; auf Wörterbuchnetz.de

[2] Bayerische Staatsbibliothek München, Signatur: L.germ. 157-3

[3] Jacob & Wilhelm Grimm; Versch.: *„Das Deutsche Wörterbuch“ (DWB)*; 1. Band 1854, 32. Band 1961; Taschenbuchausgabe: Deutscher Taschenbuch Verlag München 1984; auf Wörterbuchnetz.de

[4] Projekt „Sprache und Genome“

[5] Prof. Dr. Dietmar Seipel: Skript zur Vorlesung *„Deduktive Datenbanken“* Kapitel 2 und Kapitel 4; Universität Würzburg; 2012

[6] William F. Clocksin, Christopher S. Mellish: *„Programming in Prolog“*; Springer Berlin Heidelberg; 5. Auflage 2003

[7] Damaris Nübling: *„Historische Sprachwissenschaft des Deutschen - Einführung in die Prinzipien des Sprachwandels“*; Verlag Narr Dr. Gunter; Auflage: 2., überarbeitete Auflage 2007; S.32 ff

[8] Richard von Kienle: *„Historische Laut- und Formenlehre des Deutschen“*; Max Niemeyer Verlag Tübingen; 2. Auflage 1969

[9] Dr. Albert Hoefer: *„Beiträge zur Etymologie und vergleichenden Grammatik der Hauptsprachen des Indogermanischen Stammes, Erster Band“*; Berlin; 1. Auflage 1839; S.280

[10] Wolfgang Pfeifer: *„Etymologische Wörterbuch des Deutschen“*; Deutscher Taschenbuch Verlag München; 8. Auflage 2005; auf dwds.de

[11] D. Seipel, L. Borek: *„A Tool for Collaborative Rule-Based Morpheme Annotation“*; The International Symposium on Grids and Clouds and the Open Grid Forum (ISGC 2012); Academia Sinica, Taipei, Taiwan 2012

Genutzte Software:

[12] Jan Wielemaker u.a.: SWI-Prolog

[13] Prof. Dr. Dietmar Seipel: „The DisLog Developers' Toolkit“ (DDK)

[14] Notepad++

[15] TeXnicCenter

[16] MikTeX

[17] XAMPP

[18] Office 2010 auf Microsoft.com

Hardware und Software

Die Arbeiten für diese Diplomarbeit wurden auf einem Lenovo ideapad U310 durchgeführt. Dieser verfügt über einen Intel(R) Core(TM) i3-3217U CPU @ 1.80GHz DualCore. Weiterhin arbeitet der Laptop mit 4,00GB RAM. Das installierte Betriebssystem ist Windows 8.

Das Programm zur Alignierung der Wörterbücher wurde in der Programmiersprache *Prolog* [5] [6] geschrieben und auch weiterentwickelt. Als Programmierumgebung dient hierzu Notepad++ [14]. Genutzt wurde Prolog in der *SWI-Prolog*-Variante von Jan Wielemaker [12], erweitert durch das DDK (The DisLog Developers' Toolkit) [13], welches von Prof. Dr. Dietmar Seipel beständig weiterentwickelt wird. Zur Handhabung der SQL-Datenbanken wurde XAMPP [17] genutzt.

Die Arbeit wurde in LaTeX in der Umgebung TeXnicCenter [15] in Verbindung mit MiKTeX [16] geschrieben. Die Auswertungen wurden mit Microsoft Excel 2010 [18] erstellt.