Institut für Computerlinguistik
Universität Zürich

Automatisches Lernen und Anwenden von Syntaxbaumtransformationen mit Markow-Modellen

Lizenziatsarbeit der Philosophischen Fakultät der Universität Zürich
Referent: Prof. Dr. Michael Hess

Sonja Brodersen

sonja.brodersen@access.unizh.ch

Zürich, März 2003

Zusammenfassung

Es wird die automatische Umwandlung von Baumstrukturen, die auf unterschiedlichen kontextfreien Grammatiken beruhen, untersucht. Die Vielzahl von Unterschieden in den Grammatiktheorien bewirkt eine beträchtliche Uneinheitlichkeit von Syntaxanalysen, welche es zu verringern gilt. Es wird die Aufgabe gestellt, ein System zu entwickeln, implementieren und evaluieren, welches Syntaxbäume einer ersten Art automatisch Syntaxbäumen einer zweiten Art annähern kann.

Zur Entwicklung des Systems, welches die gestellte Aufgabe löst, finden Lösungen des Editierdistanzproblems für Baumstrukturen in Kombination mit supervisierten Lernverfahren mithilfe von Markow-Modellen Verwendung. Markow-Modelle werden bereits für die Zuweisung von Wortarten und auch für partielles Parsen eingesetzt.

Die Funktionsweise des Systems wird beschrieben. Es arbeitet in zwei Phasen: In der Trainingsphase des Systems wird eine Menge von Transformationen ermittelt, welche die Unterschiede zwischen den beiden verschiedenen Baumstrukturen repräsentieren. Aus den ermittelten Transformationen und lokalen Baumknotenkontexten wird ein Markow-Modell erstellt, welches die Strukturunterschiede festhält. Im Modell repräsentieren die Zustände die Transformationen und die Ausgaben die Knotenkontexte. Dieses Modell wird in der Anwendungsphase dazu verwendet, die wahrscheinlichsten Transformationen für einen beliebigen Baum der ersten Art zu berechnen. Schliesslich werden diese Transformationen auf den Baum angewendet, sodass ein der zweiten Art ähnlicherer Baum resultiert.

Zwei Teile gliedern diese Arbeit: Der Grundlagenteil präsentiert in kompakter Form das benötigte Hintergrundwissen. Elemente dieses Teils sind Syntaxbäume, Markow-Modelle, das Editierdistanzproblem und bekannte Anwendungen von Markow-Modellen. Im zweiten Teil werden die Architektur, Anwendung und Algorithmen des Systems eingehend beschrieben. Ausserdem wird die Leistung des Systems untersucht und bewertet.

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die maschinelle Syntaxanalyse von Sätzen nennt man Parsen. Dem Satz wird dabei eine syntaktische Struktur (Syntaxstruktur) zugeordnet. Diese Syntaxstruktur ist immer von einer Grammatik bestimmt, aufgrund derer der Parser die syntaktische Analyse durchführt. Der Nutzen einer solchen Syntaxstruktur liegt darin, dass sie bei der Bestimmung der Bedeutung eines Satzes hilfreich sein kann (Charniak, 1997).

Parser können sich täuschen. Das Hauptproblem beim Parsen ist die syntaktische Ambiguität, welche sehr häufig vorkommt. Fehler sind also nicht selten, insbesondere bei der Präpositionalphrasenanbindung, Subjekt- und Objekterkennung etc. Es ist sinnvoll, die Ausgabe eines Parsers noch weiter zu bearbeiten, indem Fehler lokalisiert und korrigiert werden.

Es gibt zahlreiche verschiedene Arten von Grammatiktheorien und dementsprechend viele Grammatiken. Eine Grammatik kann von einem Menschen manuell erstellt oder auch automatisch – dies geschieht meist mit statistischen Methoden – aus einer grossen Sammlung von Texten abgeleitet werden. Parser produzieren Syntaxstrukturen, denen eine von sehr vielen Grammatiken zugrundeliegt, was zur Folge hat, dass die Strukturen sich von Parser zu Parser, bzw. Grammatik zu Grammatik, mehr oder weniger stark unterscheiden.

Syntaxstrukturen kann man zwar gut vergleichen, aber die Interpretation des Vergleichs ist schwierig. Man sieht dies zum Beispiel beim Evaluieren von Parserausgaben. Es ist einfach, den Anteil der übereinstimmenden Teile von zwei Strukturen zu finden. Schwierig ist es aber, die Resultate linguistisch sinnvoll zu erklären und zu bewerten. Entscheidungen, welche Anteile der Strukturen wirklich falsch sind und welche nur andere Kodierungen für dasselbe Phänomen darstellen, stellen eine Schwierigkeit dar (Clematide, 2002). Wenn die Grammatik nicht direkt in Regelform sichtbar, sondern beispielsweise als probabilistisches Modell kodiert ist, stellt sich die Frage, worin sich die Grammatikmodelle unterscheiden und wo eine falsche Entscheidung getroffen wurde. Es wäre wünschenswert, systematische Abweichungen getrennt von einmaligen Abweichungen behandeln zu können.

Es kommt vor, dass Systeme für eine bestimmte Art von Syntaxstrukturen konzipiert worden sind, beispielsweise für ein spezielles annotiertes Korpus. Bei einer Anwendung dieser Systeme auf andere Syntaxstrukturen, müsste eine Anpassung des Systems an die neuen Strukturen erfolgen, d.h. man müsste eventuell sogar von Hand neue Regeln schreiben. Es wäre praktisch, wenn Syntaxbaumänderungen automatisch gelernt und durchge-

führt werden könnten.

Diese Schwierigkeiten könnten mit einer automatischen Annäherung von Syntaxbäumen an andere Syntaxbäume, die von einem anderen Parser generiert wurden, verbessert werden.

1.2 Aufgabestellung

Es wird die Aufgabe gestellt, ein System zu entwerfen, in PROLOG zu implementieren und zu evaluieren. Das System soll automatisch lernen, wie es Syntaxbäume einer ersten Art in Syntaxbäume einer zweiten Art umwandeln kann. Sodann soll das trainierte System fähig sein, einen Syntaxbaum der ersten Art einem Syntaxbaum der zweiten Art anzunähern, auch unter der Bedingung, dass der Syntaxbaum der zweiten Art unbekannt ist.

Zum Beispiel sollen Syntaxbäume, die der Lopar-Parser (siehe Schmid (2000)) ausgibt, so verändert werden, dass sie den NEGRA-Strukturen (siehe Skut u. a. (1997)) ähnlicher werden. Zur Illustration ist ein Lopar-Baum in Abbildung 1 auf der nächsten Seite zu sehen. Die Syntaxstruktur im NEGRA-Korpus für den gleichen Satz ist in Abbildung 2 (S. 4) dargestellt. Das zu entwickelnde System soll nach einer Trainingsphase in der Lage sein, den Lopar-Baum in einen neuen, NEGRA-ähnlichen Baum umzuwandeln, ohne die Zielstruktur zu kennen. Die Abbildung 3 (S. 4) zeigt den vom System transformierten Syntaxbaum, wobei der Zielbaum nicht bekannt war. Ein Vergleich zwischen dem NEGRA-Syntaxbaum und dem resultierenden Baum zeigt deutlich, dass eine starke Annäherung an die NEGRA-Struktur erreicht werden kann. Das System hat alle Knotenbeschriftungen angeglichen, den Knoten 5 zur Wurzel hinauf verschoben und alle überflüssigen Knoten, bis auf den Knoten 514, gelöscht.

Die Lösung der Aufgabe benötigt zwei Phasen: Zuerst soll in der Trainingsphase ein probabilistisches Modell erstellt werden, danach kommt in der Anwendungsphase das probabilistische Modell zur Anwendung.

In der Trainingsphase wird mittels einer grossen Anzahl von Trainingsstrukturen und einem annotierten Referenzkorpus eine Menge von Transformationen ermittelt. Daraus wird ein Markow-Modell erstellt, welches die Transformationen in Bezug auf lokale Baumstrukturen kodiert.

Die Anwendungsphase berechnet unter Verwendung des probabilistischen Modells die wahrscheinlichste Transformationsfolge für einen beliebigen Syntaxbaum. Die Anwendung der berechneten Transformationen ergibt schliesslich einen Syntaxbaum, der den Referenzstrukturen ähnlicher als der ursprüngliche Baum ist.

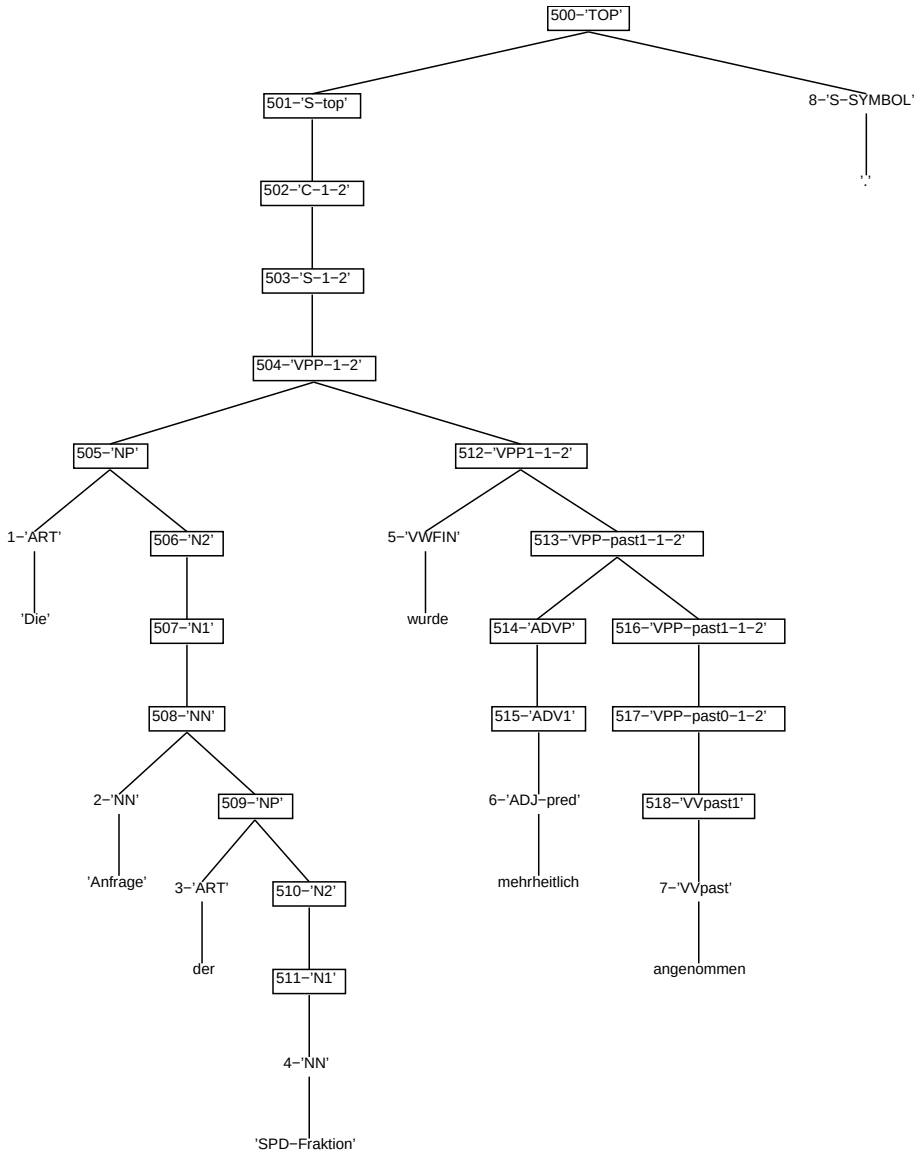


Abbildung 1: Eingabe des Systems: Original-Lopar-Baum

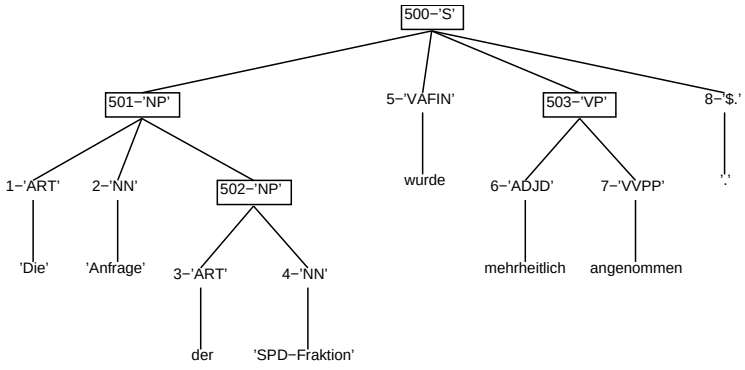


Abbildung 2: Unbekannte Zielstruktur: NEGRA-Baum

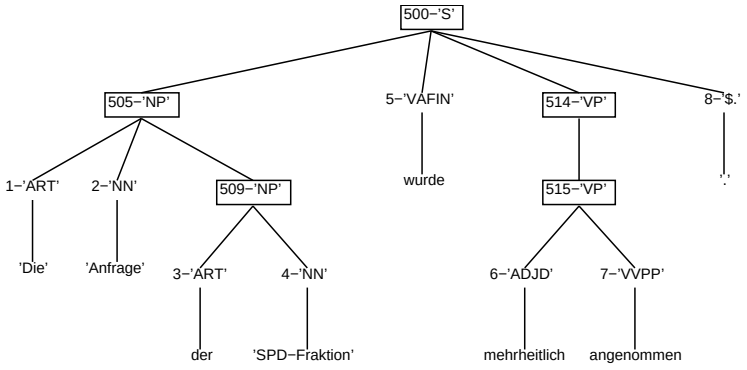


Abbildung 3: Ausgabe des Systems: NEGRA-ähnlicher, transformierter Lopar-Baum

Diese Lösung der Aufgabe setzt voraus, dass die Trainingsstrukturen mehrheitlich systematisch vom Referenzkorpus abweichen. Es wird vermutet, dass solche systematischen Abweichungen in einem geeigneten probabilistischen Modell relativ gut abgebildet werden können.

1.3 Aufbau der Arbeit

Im Wesentlichen bilden zwei Teile den Inhalt dieser Arbeit. Die erste Hälfte stellt die benötigten Grundlagen kompakt dar. Im Kapitel 2 (S. 8) liegt der Schwerpunkt auf Syntaxbäumen und ihren Zusammenhang mit Grammatiken. Die theoretischen Grundlagen von Markow-Modellen finden sich im Kapitel 3 (S. 16). Darauf folgt eine Diskussion zum transformationsbasierten Parsen (siehe Kapitel 4 (S. 26)), zur Editierdistanz von Baumstrukturen (siehe Kapitel 5 (S. 30)) und zur Anwendung von Markow-Modellen im Tagging- und Parsingkontext (siehe Kapitel 6 (S. 35)).

Die Funktionsweise des implementierten Systems, welches gemäss Aufgabestellung arbeitet, erläutert der zweite Teil der Arbeit. Im Kapitel 7 (S. 44) wird hauptsächlich eine Übersicht zur Architektur des Systems präsentiert und das verwendete Markow-Modell beschrieben. Die Definitionen und Algorithmen sowohl der Trainingsphase als auch der Anwendungsphase sind Gegenstand der nächsten beiden Kapitel 8 (S. 53) und 9 (S. 67). Gegen Ende des Systemteils steht eine Leistungsbewertung im Kapitel 10 (S. 80). Abschliessende Bemerkungen folgen im Kapitel 11 (S. 90).

Schliesslich finden sich im Anhang einige sekundäre Daten, wie die NEGRA-Kategorien, eine Modulübersicht der Implementierung und die wichtigsten Evaluierungsdaten. Es wird darauf verzichtet, den Quellkode des Systems innerhalb der Arbeit zu präsentieren. Bei Interesse sei für die Besichtigung des Quellkodes auf die URL `http://www.cl.unizh.ch/broder/liz/code` verwiesen.

Teil I

Grundlagen

2 Grammatiken und Syntaxbäume

Syntaxbäume sind die grundlegenden Objekte, die das System bearbeiten muss. Ausserdem ist der Zusammenhang zwischen Syntaxbäumen und ihrer zugrundeliegenden Grammatik für das Verständnis der Aufgabe wesentlich. Deshalb wird hier zu Beginn der Arbeit Platz für Grammatiken und Syntaxbäume eingeräumt. In der Evaluierung des Systems wird das NEGRA-Korpus als Referenz verwendet. Es ist deshalb sinnvoll, einige Grundsätze von NEGRA zu betrachten.

Dieses Kapitel gibt eine knappe Darstellung der Definition, Terminologie und Zusammenhänge von Grammatiken und Syntaxbäumen. Anschliessend werden kurz die Grundsätze von NEGRA und die im System verwendete Baumdarstellung in PROLOG diskutiert.

2.1 Stochastische kontextfreie Grammatiken (PCFG)

Eine kontextfreie Grammatik G ist ein Quadrupel $\langle V_{NT}, V_T, S, R\rangle$, wobei

- V_{NT} die endliche Menge von Nichtterminalsymbolen
- V_T die endliche Menge von Terminalsymbolen ($V = V_N \cup V_T$ die Menge von Symbolen)
- $S \in V_N$ das spezifizierte Startsymbol
- R die endliche Menge von Umschreiberegeln (engl. *rewrite rules*, auch Phrasenstrukturen genannt) $X \rightarrow \beta$, wobei gilt: $X \in V_N$ und $\beta \in V^*$,

bezeichnet (Brants, 1999b, S. 21).

Die Symbolfolge $\alpha X \gamma \in V^*$ kann genau dann in einem Schritt in $\alpha\beta\gamma \in V^*$ umgeschrieben werden, wenn eine Umschreiberegel $X \rightarrow \beta$ in R enthalten ist. Dies wird als $\alpha X \gamma \Rightarrow \alpha\beta\gamma$ notiert. Wenn eine Symbolfolge $\Phi \in V^*$ in die Symbolfolge ψ in einer endlichen Anzahl Schritte umgeschrieben werden kann, dann wird das als $\Phi \Rightarrow^* \psi$ notiert. Die Sprache $L(G)$ ist die Menge aller Symbolfolgen $W \in V_T^*$, welche von G generiert werden können. $L(G)$ wird definiert als $\{W \in V_T^* : S \Rightarrow^* W\}$.

Die Ableitung einer Terminalsymbolfolge $W \in V_T^*$ ist eine Folge der Umschreibungen von W zum Startsymbol: $S = \alpha_0 \Rightarrow \alpha_1 \Rightarrow \cdots \Rightarrow \alpha_k = W$. Eine Linksableitung von W ist diejenige Ableitung, welche in jedem Schritt das erste (= linke) Nichtterminalsymbol umschreibt.

Eine stochastische kontextfreie Grammatik (PCFG, engl. *probabilistic context-free grammar*) ist ein Quintupel $\langle V_{NT}, V_T, S, R, P \rangle$, wobei

- V_{NT} die endliche Menge von Nichtterminalsymbolen
- V_T die endliche Menge von Terminalsymbolen ($V = V_N \cup V_T$ die Menge von Symbolen)
- $S \in V_N$ das spezifizierte Startsymbol
- R die endliche Menge von Umschreiberegeln $X \rightarrow \beta$, wobei gilt: $X \in V_N$ und $\beta \in V^*$,
- P eine Wahrscheinlichkeitsfunktion von R auf $[0, 1]$, sodass $\forall X \in V_N : \sum_{\beta \in V^*} P(X \rightarrow \beta) = 1$,

bezeichnet (Brants, 1999b, S. 22).

Die Wahrscheinlichkeit einer Ableitung $P(S \Rightarrow^* W)$ ist das Produkt aller Regeln, die in der Ableitung zum Einsatz kommen. Die Wahrscheinlichkeit einer Zeichenkette ist die Summe der Wahrscheinlichkeiten aller Linksableitungen, welche die Zeichenkette ergeben.

2.2 Syntaxbäume

Ein Syntaxbaum T ist ein gerichteter, geordneter und beschrifteter Baum. Man kann einen Syntaxbaum durch ein Sextupel $\langle N_{NT}, N_T, L, l, R, D \rangle$ beschreiben, wobei

- N_{NT} die endliche Menge von Nichtterminalknoten (kurz: Nichtterminalen)
- N_T die endliche Menge von Terminalknoten (kurz: Terminalen) ($N = N_{NT} \cup N_T$ die Menge von Knoten)
- L die endliche Menge von Beschriftungen
- $l : N \rightarrow L$ die surjektive Beschriftungsfunktion
- $R \in N$ den spezifizierten Wurzelknoten
- D die endliche Menge von Mutter-Kinder-Beziehungen $M \rightarrow K$, wobei gilt:
  - $M \in N_{NT}$ und M hat die Bedeutung: Mutter von jedem $k_i \in K$

– $K \in N^*$ und K hat die Bedeutung: Eine geordnete Folge von Kindern $K = k_1, \ldots, k_n$

bezeichnet.

Die Elemente $M \rightarrow K \in D$ drücken zwei elementare Beziehungen gleichzeitig aus: Die unmittelbare Dominanz und die lineare Präzedenz unter den Kindern.

Der Aufbau von Bäumen kann anschaulich durch die folgende rekursive Vorschrift beschrieben werden: Ein einzelner Knoten ist ein Baum. Ein oder mehrere Bäume $T_1, \ldots, T_n$ ($n \geq 1$) können je mit einer Kante an einen Knoten x gehängt werden, sodass aus dieser Verbindung wiederum ein Baum B entsteht. Die Bäume $T_1, \ldots, T_n$ heissen dann Teilbäume von B. Jeder Knoten jedes angehängten Teilbaums wird von x dominiert.

Ein gerichteter Baum hat eine Wurzel R, die von keinem anderen Knoten dominiert wird. Jeder Knoten eines Baumes kann die Wurzel für einen Teilbaum bilden. Gibt es einen Knoten m, der einen Knoten k unmittelbar dominiert (d.h. k hängt direkt mit einer Kante an m), heisst m Mutter von k und k Kind von m. Gibt es einen weiteren Knoten s, der von m unmittelbar dominiert wird, dann nennt man s und k Schwestern. Knoten, die keine Kinder besitzen, heissen Terminale.

Das Niveau $n(k)$ eines Knotens k ist gleich der Länge, d.h. der Anzahl Knoten, des Pfades zur Wurzel R des Baumes. Die Wurzel selbst besitzt das Niveau 0. Die Tiefe des Baumes ist das grösste auftretende Niveau.

Da der Zugriff auf die einzelnen Knoten eines Baumes nur indirekt über die Wurzel erfolgen kann, werden Algorithmen benötigt, die beim Durchlaufen eines Baumes jeden Knoten genau einmal besuchen. Später in dieser Arbeit werden zwei verschiedene Baumdurchläufe verwendet. Ihre rekursive Definition lautet:

Preorder-Durchlauf: Beginne bei der Wurzel.

1. Besuche den Knoten.
2. Falls der Knoten Kinder hat, durchlaufe in Preorder die Teilbäume seiner Kinder der Reihe nach von links nach rechts.

Postorder-Durchlauf: Beginne bei der Wurzel.

1. Falls der Knoten Kinder hat, durchlaufe in Postorder die Teilbäume seiner Kinder der Reihe nach von links nach rechts.
2. Besuche den Knoten.

Ein Syntaxbaum der Terminalsymbolfolge W entspricht der Baumrepräsentation einer Linksableitung von W. D.h. die Wurzel ist mit S beschriftet, die Terminale sind mit Elementen von V_T beschriftet, und die Spannweite des Baumes ist W. Alle internen Knoten sind mit Elementen aus V_{NT} beschriftet, sodass sie die einzelnen Umschreibungsschritte der Linksableitung abbilden.

Das Abbildungsverhältnis zwischen Syntaxbaum und grammatischer Struktur ermöglicht es, aus Syntaxbäumen die zugrundeliegende Phrasenstrukturgrammatik abzulesen. Syntaxbäume von verschiedenen Parsern enthalten zwar meistens eine vergleichbare Information, unterscheiden sich aber oft wesentlich in grammatischen Konzepten und in der Notation. Notationsformate können im Allgemeinen mit einfachen Skripts maschinell umgewandelt werden, wohingegen grammatische Konzepte nicht so einfach ineinander übersetzt werden können.

2.3 NEGRA-Korpus

Das NEGRA-Korpus ist eine Sammlung von deutschen Sätzen, die linguistisch interpretiert sind. Solche Sammlungen werden auch Baumbanken (engl. *treebanks*) genannt. Die Sätze stammen aus der Frankfurter Rundschau und sind manuell bzw. semi-automatisch mit Syntaxstrukturen versehen. Die Syntaxstrukturen sind typischerweise Bäume und möglichst neutral, d.h. von verschiedenen Syntaxtheorien unabhängig und deswegen vielseitig verwendbar.

Das Annotieren in NEGRA besteht einerseits im Konstruieren von hierarchischen Repräsentationen für Sätze und andererseits im Hinzufügen von kategorieller und funktionaler Information. Die Repräsentationen werden anhand einer vordefinierten Grammatik erstellt. Um den Annotationsaufwand zu minimieren, wurde das Korpus semi-automatisch erstellt. D.h. eine Annotation wurde von einem Annotationstool vorgeschlagen und von zwei unabhängigen Personen geprüft und gegebenenfalls verändert. Anschliessend wurden die Annotationen abgeglichen.

NEGRA unterscheidet sich von anderen Baumbanken, wie z.B. von der Penn Treebank[1], in einigen Punkten. Die Penn Treebank weist kontextfreie Konstituentenstrukturen auf, in welchen nicht lokale Abhängigkeiten mit Spuren (engl. *traces*) behandelt werden. Da NEGRA deutsche Sätze enthält, muss es möglichst übersichtlich mit der relativ freien Wortstellung des Deutschen umgehen können. Im Deutschen kommen häufig nichtkonti-

[1]Ein Projekt der University of Pennsylvania siehe URL `http://www.cis.upenn.edu/~treebank/home.html`

nuierliche Konstituenten vor, beispielsweise in Topikalisierungen, Scrambling, Extrapositionen etc. Dies führte zu einer übermässigen Verwendung von Spuren, was in unübersichtlichen Strukturen resultieren würde.

Skut u. a. (1997) begründen deshalb die Verwendung von Argumentenstrukturen als Alternative. Die Argumentenstruktur kann in Bäumen dargestellt werden, welche überkreuzende Kanten aufweisen. Sie entscheiden sich für einfache, d.h. flache Strukturen, um das Ambiguitätspotenzial zu reduzieren. Diese Einschränkung auf flache Strukturen machen sie mit funktionalen Kantenbeschriftungen wieder wett.

Die Kodierung von verschiedenen Typen von Informationen im Korpus umfasst:

Argumentenstruktur: In Bäumen mit überkreuzenden Kanten wird die Argumentenstruktur dargestellt.

Morphologische Analyse: Die ersten 60.000 Tokens sind mit morphologischen Informationen versehen.

Wortartinformation: Jedes Terminal erhält ein Tag. Es wird das Stuttgart-Tübingen-Tagset (STTS) verwendet (siehe Anhang A.1 (S. 94)).

Syntaktische Kategorien: Die Kategorien von nichtterminalen Knoten beschriften den Knoten (siehe Anhang A.2 (S. 96)).

Grammatische Funktion: Die grammatische Funktion (z.B. SB für Subjekt, HD für Kopf etc.) einer Konstituente in der unmittelbar dominierenden Phrase beschriftet die betreffende Kante. Da Kantenbeschriftungen oft nicht Teil der Syntaxbaumdefinition sind, werden sie zum Mutterknoten, wo die Kante herkommt, hinaufprojiziert und zur zusätzlichen Knotenbeschriftung gemacht.

Da NEGRA alle Äusserungen annotiert, ist die Existenz von einzelnen Nominalphrasen, die in keinen Satz eingebettet sind, nicht auszuschliessen.

Bei Verwendung von probabilistischen Methoden in der Verarbeitung natürlicher Sprache ist es üblich, einen Teil einer Baumbank als Referenz für das Training und einen anderen Teil für die Evaluierung zu benutzen. Das Vorkommen von Konstellationen in der Baumbank kann beispielsweise zum Erstellen eines probabilistischen Modells benutzt werden. Die manuell annotierten Syntaxbäume der Baumbank werden als Wahrheit (engl. *gold standard, truth*) betrachtet. Die Tatsache, dass immer noch Fehler in Baumbanken vorkommen, muss leider hingenommen werden.

2.4 Formate

2.5 Das zeilenbasierte Exportformat

Das zeilenbasierte Exportformat enthält die folgenden Einheiten pro Zeile: Token bzw. Knotennummer, Beschriftung des Knotens mit der Wortart bzw. der syntaktischen Kategorie, morphologische Information zu einzelnen Tokens, Beschriftung der Kante zur Mutter mit der grammatischen Funktion der Konstituente und der Mutterknoten.

```
#BOS 2 2 899973978 1
Sie                     PPER    3.Pl.*.Nom      SB      504
gehen                   VVFIN   3.Pl.Pres.Ind   HD      504
gewagte                 ADJA    Pos.*.Akk.Pl.St NK      500
Verbindungen            NN      Fem.Akk.Pl.*    NK      500
und                     KON     --              CD      502
Risiken                 NN      Neut.Akk.Pl.*   CJ      502
ein                     PTKVZ   --              SVP     504
,                       $,      --              --      0
versuchen               VVFIN   3.Pl.Pres.Ind   HD      505
ihre                    PPOSAT  *.Akk.Pl        NK      501
Möglichkeiten           NN      Fem.Akk.Pl.*    NK      501
auszureizen             VVIZU   --              HD      503
.                       $.      --              --      0
#500                    NP      --              CJ      502
#501                    NP      --              OA      503
#502                    CNP     --              OA      504
#503                    VP      --              OC      505
#504                    S       --              CJ      506
#505                    S       --              CJ      506
#506                    CS      --              --      0
#EOS 2
```

2.6 Pennformat

Das Pennformat stellt die Baumstruktur im Format analog zur Penn-Treebank mit einer Klammerstruktur dar. Die Kantenbeschriftung wird zum Mutterknoten projiziert.

```
(
```

```
        (CS
                (S-CJ
                        (PPER-SB Sie)
                        (VVFIN-HD gehen)
                        (CNP-OA
                                (NP-CJ
                                        (ADJA-NK gewagte)
                                        (NN-NK Verbindungen)
                                )
                                (KON-CD und)
                                (NN-CJ Risiken)
                        )
                        (PTKVZ-SVP ein)
                )
                ($, ,)
                (S-CJ
                        (VVFIN-HD versuchen)
                        (VP-OC
                                (NP-OA
                                        (PPOSAT-NK ihre)
                                        (NN-NK Möglichkeiten)
                                )
                                (VVIZU-HD auszureizen)
                        )
                )
        )
        ($. .)
)
```

2.7 Formatanpassungen im Anwendungsfall des implementierten Systems

Die Lösung der Aufgabe, Syntaxbäume verschiedener Art zu vergleichen und einander anzunähern, verlangt ein einheitliches Format, um diese Bäume zu repräsentieren. Der Einfachheit halber gilt die Einschränkung auf Bäume, die keine überkreuzenden Kanten enthalten. Zudem sollen nur Sätze behandelt werden, keine alleinstehenden Nominalphrasen

oder Ähnliches. Die Bäume sollen mit PROLOG verarbeitet werden. Diese Bedingungen begründen ein vorheriges Erstellen einer Teilmenge von NEGRA, in einem bestimmten Format.

Das Ausgangs- und Zielformat der Syntaxbäume in PROLOG basiert auf drei verschiedenen Prädikaten. Natürlich wären unzählige Arten von Baumrepräsentationsformen für jedes spezifische Problem ideal gewesen. Da aber ständige Formatänderungen verhindert werden sollten, wurde ein möglichst flexibles Format bevorzugt (**ID/LP-Format**).

Die drei PROLOG-Prädikate sind:

`id(Satz, Knoten, Kind).` Im Satz mit der Nummer `Satz` dominiert der Knoten mit der Nummer `Knoten` unmittelbar den Knoten mit der Nummer `Kind`. Man nennt diese Beziehung unmittelbare Dominanz.

`lp(Satz, Knoten, RechteSchwester).` Im Satz mit der Nummer `Satz` sind die beiden Knoten `Knoten` und `RechteSchwester` Schwestern und `Knoten` steht links von `RechteSchwester`. Diese Beziehung heisst unmittelbare lineare Präzedenz.

`feature(Satz, Knoten, Merkmal).` Mit diesem Prädikat werden Eigenschaften von Knoten festgehalten. `Merkmal` kann drei verschiedene Strukturen enthalten:

`Merkmal = hd=Wort` In diesem Fall ist der `Knoten` ein Terminal. Es hat das `Wort` als Zusatzinformation bei sich. Wörter werden nicht in `id/3`-Fakten repräsentiert, sondern zu den Tags hinaufgeschoben. Dies spart Speicherplatz und Einlese- und Rechenzeit, weil dadurch die Anzahl `id/3`-Fakten reduziert wird und für die Wörter keine Knotennummern vergeben werden müssen.

`Merkmal = cat=Kategorie` Jeder Knoten hat eine `Kategorie`. Dies ist bei Terminalen die Wortart des entsprechenden Wortes (z.B.: NN, VVFIN) oder in allen anderen Fällen die Phrasenkategorie (z.B.: S, NP, VP).

`Merkmal = fun=Funktion` Eine Phrase kann eine grammatische Funktion haben, wie z.B. Subjekt, Genitivobjekt etc. Wie man sieht, wurde hier die grammatische Funktion zum Mutterknoten hinaufprojiziert.

Jeder Knoten ausser der Wurzel kommt genau einmal in einer `id/3`-Klausel als `Kind` vor.

3 Markow-Modelle

Das probabilistische Modell ist das Resultat des Lernprozesses, welchen das System durchläuft, und somit das Herzstück des Systems. Man kann das entwickelte System nicht verstehen, ohne Markow-Modelle zu kennen.

Es werden zunächst Markow-Prozesse kurz erläutert und erweitert zu Markow-Modellen. Dann folgt die Besprechung eines Algorithmus, wie ein Modell zu Berechnungen verwendet werden kann. Das Erstellen von Markow-Modellen bildet den Schluss dieses Kapitels.

3.1 Markow-Prozesse

Markow-Prozesse (auch: markowsche Ketten) sind stochastische Prozesse. Sie behandeln also Folgen von Zufallsvariablen, die voneinander abhängig sind. Die Zufallsvariablen $X = X_1, \ldots, X_t$ können Werte aus einer endlichen Menge von Zuständen $S = \{s_1, \ldots, s_n\}$ annehmen. Die Art der Abhängigkeit ist durch die markowsche Eigenschaft definiert, welche bedeutet, dass die Wahrscheinlichkeit einer Zufallsvariable X_{t+1} in einer Folge nur von einer limitierten Anzahl m seiner Vorgängervariablen abhängt. Für den Fall, dass $m = 1$ gilt, betrachtet man zwei Zustände, die unmittelbar aufeinanderfolgen, sogenannte Bigramme von Zuständen. In diesem Fall spricht man von einem Markow-Prozess erster Ordnung.

$$P(X_{t+1} = s_i \mid X_1 = s_1, \ldots, X_t = s_k) = P(X_{t+1} = s_i \mid X_t = s_k) \tag{3.1}$$

Ein Markow-Prozess kann als ein System aus einer Menge von Zuständen, welche miteinander durch Übergänge verbunden sind, veranschaulicht werden. Die Übergänge sind mit Übergangswahrscheinlichkeiten beschriftet. Die Folge entsteht, indem das System zu jedem Zeitpunkt t von einem Zustand dem Übergang entlang in den nächsten Zustand übergeht.[2]

Eine kompakte Beschreibung eines Markow-Prozesses bietet eine Übergangsmatrix A:

$$a_{ij} = P(X_{t+1} = s_j \mid X_t = s_i) \tag{3.2}$$

[2] Diese Vorstellung beruht auf der Ähnlichkeit zu endlichen Automaten.

wobei $\sum_{j=1}^{N} a_{ij} = 1$ gilt, d.h. dass die Summe aller Übergänge, die von einem Zustand ausgehen, 1 ist. Zusätzlich nehmen wir einen Startzustand s_0 an, welcher Übergänge zu jedem $X_1 = s_i$ hat.

Im Markow-Prozess erster Ordnung ist die Wahrscheinlichkeit einer Folge von Zuständen das Produkt der betreffenden Übergangswahrscheinlichkeiten und einfach zu berechnen:

$$\begin{aligned} P(X_1, \ldots, X_T) &= P(X_1|s_0)P(X_2|X_1)P(X_3|X_1, X_2) \cdots P(X_T|X_1, \ldots, X_{T-1}) \\ &= P(X_1|s_o)P(X_2|X_1)P(X_3|X_2) \cdots P(X_T|X_{T-1}) \\ &= \prod_{t=0}^{T-1} a_{X_t X_{t+1}} \end{aligned} \tag{3.3}$$

3.2 Markow-Modelle

Ein Markow-Modell unterscheidet sich von einem Markow-Prozess darin, dass es eine zusätzliche Eigenschaft aufweist: Die Zustände senden Signale aus.[3] Jeder Zustand kann mehrere Signale aussenden und ein Signal kann von verschiedenen Zuständen ausgegeben werden. Die Definition der Ausgabewahrscheinlichkeiten lautet:

$$P(O_t = \sigma_k \mid X_t = s_i) = b_{ik} \tag{3.4}$$

Ein Markow-Modell ist ein Quadrupel $\langle S, \Sigma, A, B \rangle$, wobei

- $S = \{s_0, \ldots, s_n\}$ die Menge aller Zustände
- $\Sigma = \{\sigma_1, \ldots, \sigma_m\}$ das Ausgabealphabet
- $A = \{a_{ij}\}, i, j \in S$ die Menge aller Wahrscheinlichkeiten, dass unmittelbar nach dem Zustand i der Zustand j eintritt,
- $B = \{b_{ik}\}, s_i \in S, \sigma_k \in \Sigma$ die Menge aller Wahrscheinlichkeiten, dass der Zustand i das Signal k aussendet,

bezeichnen.

- Die Zustandsfolge ist $X = X_1, \ldots, X_T \quad X_t : S \mapsto \{0, \ldots, n\}$.
- Die Ausgabefolge ist $O = O_1, \ldots, O_T \quad O_t \in \Sigma$.

[3] Dies ist ein zustandsemittierendes Modell. Oft werden auch übergangsemittierende Modelle verwendet, in welchen die Ausgabe vom aktuellen und vom vorherigen Zustand abhängt.

3.2.1 Beispiel: Der verrückte Getränkeautomat

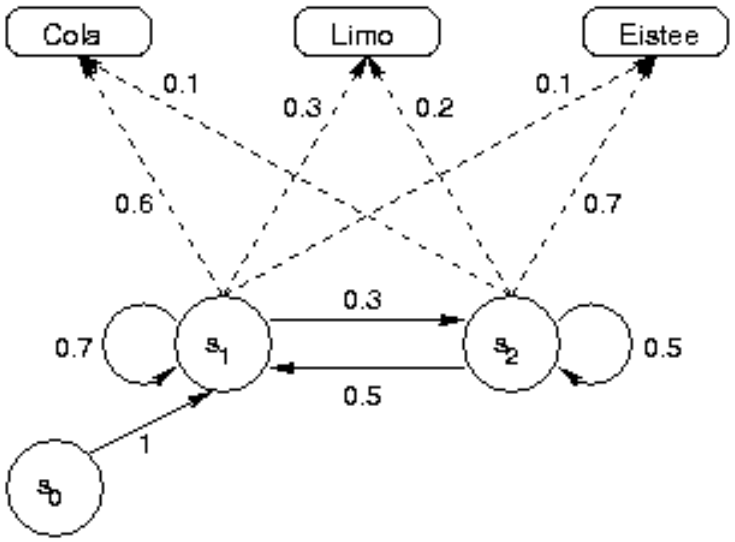


Abbildung 4: Der verrückte Getränkeautomat

Die Abbildung 4 zeigt den verrückten Getränkeautomaten (vgl. *the crazy softdrink machine* in Manning und Schütze (1999, S. 321)). Der Automat hat drei Zustände: Den Startzustand s_0, s_1 und s_2. s_1 steht für 'Cola bevorzugt' und s_2 steht für 'Eistee bevorzugt'. Zwischen s_1 und s_2 wechselt der Automat nach jedem Getränkebezug zufällig hin und her.

Die Übergangswahrscheinlichkeiten und die Ausgabewahrscheinlichkeiten der Abbildung 4 sind nochmals in den Tabellen 1 und 2 auf der nächsten Seite zusammengestellt.

	nach		
von	s_0	s_1	s_2
s_0	0	1	0
s_1	0	0.7	0.3
s_2	0	0.5	0.5

Tabelle 1: Übergangswahrscheinlichkeiten

Die Betrachtung der Ausgabewahrscheinlichkeiten zeigt, dass der Zustand s_1 tatsächlich die Ausgabe von Cola bevorzugt. Analog dazu bevorzugt s_2 die Ausgabe von Eistee. Doch die Ausgaben aller Getränke ist in beiden Zuständen möglich.

	Ausgabe		
Zustand	Cola	Eistee	Limo
s_1	0.6	0.1	0.3
s_2	0.1	0.7	0.2

Tabelle 2: Ausgabewahrscheinlichkeiten

Im Kapitel 3.3.1 (S. 22) wird der Getränkeautomat nochmals zur Veranschaulichung als Beispiel verwendet.

3.3 Hidden-Markow-Modelle

Ein Hidden-Markow-Modell bezeichnet ein Markow-Modell, welches beispielsweise dazu verwendet wird, von einer beobachteten Ausgabefolge O Rückschlüsse auf die wahrscheinlich zugrundeliegende Zustandsfolge X zu ziehen. Die Zustände sind dann nicht direkt beobachtbar.

Die Notation $\max_X f(X)$ bedeutet im Folgenden den maximalen Wert von f, wenn X variiert. Wenn der Wert von X gesucht ist, der f maximiert, wird das mit $\underset{X}{\text{argmax}} f(X)$ notiert.[4]

Die beste Chance, die zugrundeliegende Zustandsfolge für die Ausgabe O zu sein, hat die wahrscheinlichste Zustandsfolge. Formal ausgedrückt:

$$\underset{X}{\text{argmax}} P(X \mid O) \tag{3.5}$$

Manchmal (z.B. in der Modellparameter-Berechnung) ist der Ausdruck $P(X \mid O)$ nicht ideal für eine Berechnung. Dann wird er unter Zuhilfenahme der Umkehrformel von Bayes umgeformt:

$$\underset{X}{\text{argmax}} P(X \mid O) = \underset{X}{\text{argmax}} \frac{P(O \mid X)P(X)}{P(O)} \tag{3.6}$$

Der Nenner $P(O)$ ist für beliebige X konstant. Er kann also bei der Berechnung weggelassen werden. So maximiert auch

$$\underset{X}{\text{argmax}} P(X \mid O) = \underset{X}{\text{argmax}} P(O \mid X)P(X) \tag{3.7}$$

[4]Falls mehrere Werte von X f maximieren, wird zufällig ein Wert genommen.

die wahrscheinlichste Zustandsfolge.

Der Vorteil dieser zweiten Form liegt darin, dass sie das Modell in die zwei Bereiche Ausgaben $P(O \mid X)$ und Zustandsübergänge $P(X)$ teilt.

$$P(X) = \prod_{t=0}^{T-1} a_{X_t X_{t+1}} \tag{3.8}$$

$$P(O \mid X) = \prod_{t=1}^{T} b_{X_t O_t} \tag{3.9}$$

3.3.1 Die wahrscheinlichste Zustandsfolge für eine bestimmte Ausgabe berechnen

Eine häufige Fragestellung im Zusammenhang mit Hidden-Markow-Modellen lautet: Wir haben eine Beobachtung gemacht, wie können wir nun mit möglichst kleinem Rechenaufwand bestimmen, welches die wahrscheinlichste Zustandsfolge durch das Modell war, um diese Beobachtung zu erklären? Gesucht wird also dasjenige X, welches die Wahrscheinlichkeit maximiert:

$$\operatorname*{argmax}_X P(X \mid O) \tag{3.10}$$

Diese Fragestellung ist im Allgemeinen für Klassifikationsfragen interessant und wird auch für probabilistisches Tagging verwendet (siehe dazu Abschnitt 6.1 (S. 35)).

In der Abbildung 5 auf der nächsten Seite ist ein Graph dargestellt, der einen Knoten für jedes Zustand-Zeitpunkt-Paar besitzt, wobei jeder Knoten der Zeit t genau mit jedem Knoten der Zeit $t-1$ und $t+1$ verbunden ist. Im Folgenden wird ein solcher Graph Gitteranordnung oder gitterartige Anordnung (engl. *trellis*) genannt. Ein Knoten an der Stelle (s,t) bedeutet, dass sich das Markow-Modell zur Zeit t im Zustand s befindet, er kann Informationen zu den bisherigen Zustandssequenzen speichern bis zu $X_t = s$. Eine solche Gitteranordnung wird bevorzugt für Algorithmen verwendet, die dynamische Programmierung, d.h. Akkumulatoren, benutzen.

Ein effizienter Algorithmus für die Berechnung der wahrscheinlichsten Zustandsfolge bei gegebener Ausgabe ist der Viterbi-Algorithmus. Die Menge der Variablen $\delta_t(i)$ wird definiert:

$$\delta_t(i) = \max_{X_1,\ldots,X_{t-1}} P(X_1,\ldots,X_{t-1}, X_t = s_i, O_1,\ldots,O_t) \tag{3.11}$$

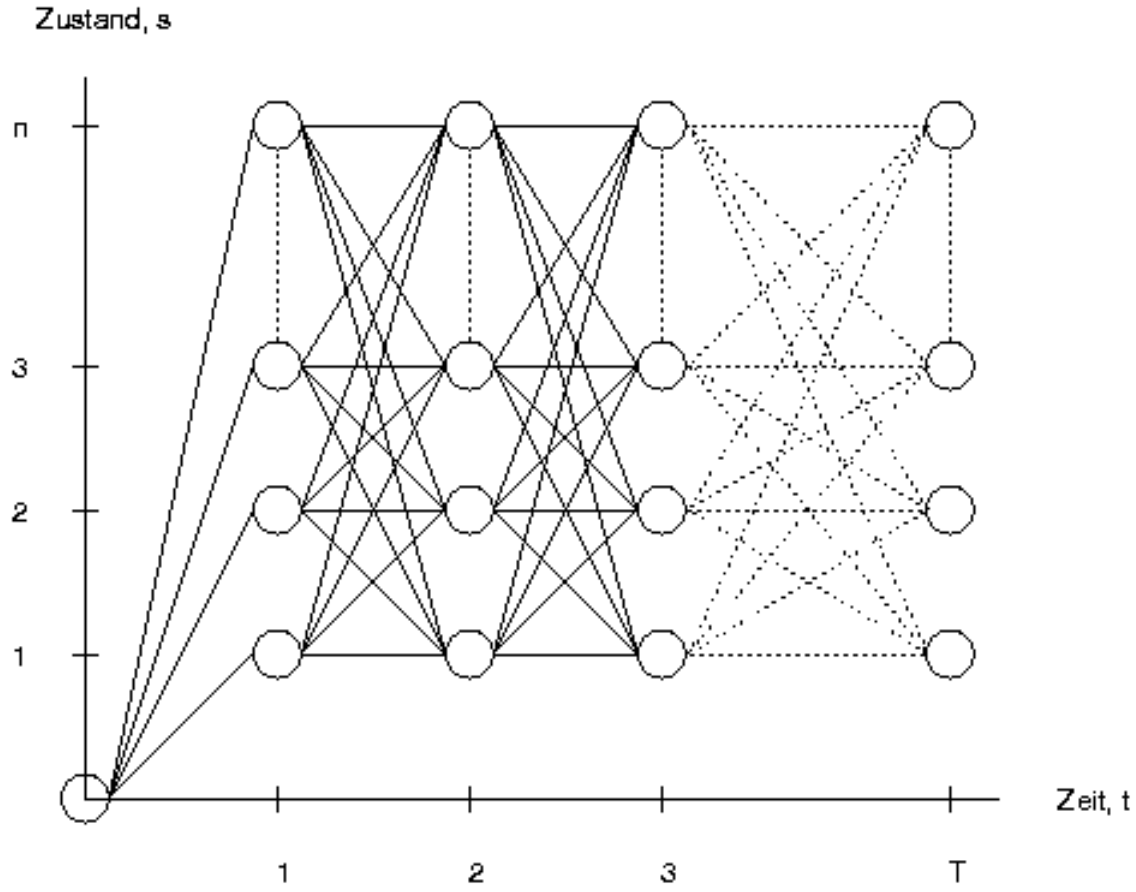


Abbildung 5: Gitterartige Anordnung von Zuständen relativ zur Zeit

Der Akkumulator δ speichert also in jedem Knoten der Gitteranordnung den Wert des wahrscheinlichsten Pfades zu diesem Knoten.

Initialisierung:

(3.12) $$\delta_1(i) = a_{0i} b_{iO_1} \text{ , für } 1 \leq i \leq n$$

Die Initialisierung findet im Zeitpunkt 1 statt. Für jeden Knoten gibt es genau einen Vorgängerknoten, nämlich $(0, 0)$. $\delta_1(i)$ ist jeweils die Wahrscheinlichkeit des Übergangs vom Startzustand zu diesem Zustand i multipliziert mit der Wahrscheinlichkeit der Ausgabe von diesem Zustand i.

Rekursion:

$$\delta_{t+1}(j) = (\max_{1 \leq i \leq n} \delta_t(i) a_{ij}) b_{jO_t} \text{ , für } 2 \leq t \leq T-1 \text{ und } 1 \leq j \leq n \tag{3.13}$$

$$\psi_{t+1}(j) = \operatorname*{argmax}_{1 \leq i \leq n}(\delta_t(i) a_{ij}) \text{ , für } 2 \leq t \leq T-1 \text{ und } 1 \leq j \leq n \tag{3.14}$$

Jeder Knoten $(t+1, j)$ hat n Vorgängerknoten, für jeden dieser Vorgängerknoten (t, i) wird das Produkt aus dessen $\delta_t(i)$ und der Übergangswahrscheinlichkeit berechnet. Aus der Menge all dieser Werte wird der höchste mit der Ausgabewahrscheinlichkeit multipliziert. Das Resultat wird in $\delta_{t+1}(j)$ gespeichert. Damit später noch feststellbar ist, welcher Vorgängerknoten (t, i) den höchsten Wert generiert hat, wird dessen Zustandsnummer i in $\psi_{t+1}(j)$ gespeichert.

Dies wird für alle Knoten der Gitteranordnung durchgeführt.

Terminierung und Pfadrekonstruktion:

$$\hat{X}_T = \operatorname*{argmax}_{1 \leq i \leq n} \delta_T(i) \tag{3.15}$$

$$\hat{X}_t = \psi_{t+1}(X_{t+1}) \text{ , für } T-1 \leq t \leq 1 \tag{3.16}$$

Ist der Zeitpunkt T erreicht, kann der letzte Zustand $\hat{X}_T$ des Pfades lokalisiert werden: Es ist der Zustand, dessen Knoten das grösste $\delta_T(i)$ besitzt. Danach kann der Pfad rückwärts nochmals durchlaufen werden, indem jeweils vom aktuellen Knoten $(t+1, i)$ zu demjenigen Knoten zurückgegangen wird, dessen Zustandsnummer in $\psi_{t+1}(i)$ gespeichert ist.

Beispiel: Viterbi am Getränkeautomaten Die Abbildung 6 auf der nächsten Seite zeigt die Gitteranordnung für den Getränkeautomaten. Der wahrscheinlichste Pfad folgt den fett gezeichneten Pfeilen. Die Tabelle 3 auf der nächsten Seite zeichnet alle Schritte auf, die zur Berechnung des wahrscheinlichsten Pfades für die Ausgabe (Cola, Eistee, Limo) nötig sind.

3.3.2 Parameter bestimmen

Bisher war das Modell bereits schon definiert und alle Wahrscheinlichkeiten von vornherein gegeben. In den meisten Anwendungen ist es aber so, dass das Modell zwar theoretisch

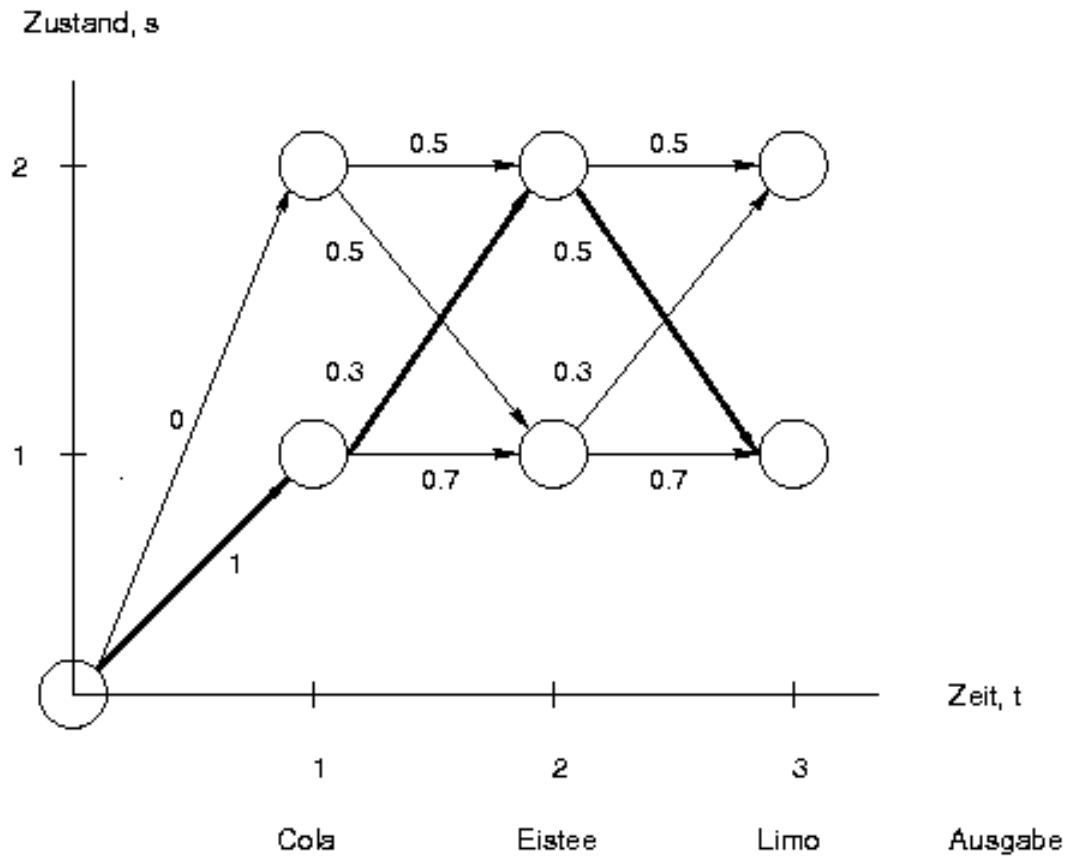


Abbildung 6: Gitteranordnung für die Ausgabe (Cola, Eistee, Limo)

Phase	t	X_t	$\delta_{t-1}(X_{t-1})a_{X_{t-1}X_t}$		$\delta_t(X_t)$		$\psi_t(X_t)$
Initialisierung	1	1	$1 \cdot 1$	$= 1$	$1 \cdot 0.6$	$= 0.6$	**0**
		2	$1 \cdot 0$	$= 0$	$0 \cdot 1$	$= 0$	0
Rekursion	2	1	$0.6 \cdot 0.7$	$= 0.42$	$0.42 \cdot 0.1$	$= 0.042$	1
			$0 \cdot 0.5$	$= 0$			
		2	$0.6 \cdot 0.3$	$= 0.18$	$0.18 \cdot 0.7$	$= 0.126$	**1**
			$0 \cdot 0.5$	$= 0$			
	3	1	$0.042 \cdot 0.7$	$= 0.0294$			
			$0.126 \cdot 0.5$	$= 0.063$	$0.063 \cdot 0.3$	$= 0.0189$	**2**
		2	$0.042 \cdot 0.3$	$= 0.0126$			
			$0.126 \cdot 0.5$	$= 0.063$	$0.063 \cdot 0.2$	$= 0.0126$	2
Rekonstruktion	3	Maximum in Zustand 1					2
	2	Maximum in Zustand 2					1
	1	Maximum in Zustand 1					0

Tabelle 3: Die wahrscheinlichste Zustandsfolge für (Cola, Eistee, Limo)

konzipiert wurde, d.h. dass festgelegt ist, welche realen Fakten das Modell wie repräsentiert, aber das Modell ist noch nicht praktisch erstellt. Das Erstellen eines Markow-Modells verlangt die Berechnung von den Übergangswahrscheinlichkeiten A und den Ausgabewahrscheinlichkeiten B für jeden Zustand. Wie kommt man nun zu diesen Werten?

Im einfachsten Fall besteht die Möglichkeit, annotierte Trainingsdaten zu konsultieren, d.h. auf Daten zuzugreifen, die beide Modellbereiche zeigen. Die Ausgabefolge und die zugrundeliegende Zustandsfolge sind dann bekannt. Die Wahrscheinlichkeiten können so direkt aus einem Trainingskorpus geschätzt werden. Die Schätzung erfolgt mit der relativen Häufigkeit im Korpus, dies entspricht einer Maximum-Likelihood-Schätzung.

$$\hat{a}_{ij} = \frac{f(s_i, s_j)}{f(s_i)} \tag{3.17}$$

$$\hat{b}_{jk} = \frac{f(s_j, \sigma_k)}{f(s_j)} \tag{3.18}$$

$f(s_i, s_j)$ bedeutet in diesem Fall die absolute Häufigkeit, dass die Zustände s_i und s_j in den Trainingsdaten direkt aufeinanderfolgen, also ein Zustandsbigramm darstellen.

Es ist auch möglich, die Modellparameter zu berechnen, wenn aus den Trainingsdaten nur die Ausgabefolge ersichtlich ist und die Zustandsfolge tatsächlich unbekannt ist. Man verwendet dazu z.B. den Baum-Welch-Algorithmus (Baum u. a., 1970). Für eine ausführ-

liche Besprechung dieses Falls sei beispielsweise auf Krenn und Samuelsson (1997) oder Manning und Schütze (1999, S. 333) verwiesen. Ein Training mit einem annotierten Korpus ergibt aber im Allgemeinen bessere Resultate (Elworthy, 1994).

3.3.3 Problem der spärlichen Datengrundlage

Sogar mit sehr grossen Datenmengen entsteht das Problem der spärlichen Datengrundlage (engl. *sparse data problem*), weil Trainingsdaten nie vollständig sind. Damit wird die Tatsache bezeichnet, dass viele Häufigkeiten, die zum Beispiel für die Berechnung der Übergangswahrscheinlichkeit gebraucht werden, null sind ($f(s_1 \dots s_n) = 0$ für viele $s_1 \dots s_n \in S^n$). Daraus entsteht ein unerwünschter Effekt: Wenn eine Zustandsfolge $s_1 \dots s_k, k \geq 1$ eine Teilfolge $s_l \dots s_m, 1 \leq l \leq m \leq k$ enthält, die eine Wahrscheinlichkeit von null aufweist, wird die gesamte Folge $s_1 \dots s_k$ die Wahrscheinlichkeit null erhalten:

$$P(s_l \dots s_m) = 0 \Longrightarrow P(s_1 \dots s_{l-1} s_l \dots s_m s_{m+1} \dots s_k) = 0 \tag{3.19}$$

So erhalten alle Folgen die gleiche Wahrscheinlichkeit, ganz egal, welche Werte $s_1 \dots s_{l-1}$ und $s_{m+1} \dots s_k$ haben. Dieser unerwünschte Effekt sollte vermieden werden. In der Literatur werden verschiedene Methoden zum Vermeiden des Werts null (engl. *smoothing*) vorgeschlagen. Solche Methoden senken in der Regel die Wahrscheinlichkeiten von den in den Trainingsdaten gesehenen Konstellationen, damit noch ein bisschen Wahrscheinlichkeit für die unbekannten Konstellationen übrigbleibt. Eine kleine Auswahl solcher Methoden:

Expected-Likelihood-Schätzung: Alle Häufigkeiten f werden durch $f^* = f + 0.5$ ersetzt. Die neuen Häufigkeiten f^* werden für die Maximum-Likelihood-Schätzung benutzt. (Gale und Church, 1990)

Good-Turing-Methode: Alle Häufigkeiten f werden durch $f^* = (f+1)N_{f+1}/N_f$ ersetzt, wobei N_f die Häufigkeit der Häufigkeit f bezeichnet. (Good, 1953)

Back-Off-Modelle: In diesen Modellen stehen mehrere Modelle zur Auswahl, die sich im Grad der Spezifität unterscheiden. Verwendet wird das detaillierteste Modell, welches zu einem bestimmten Kontext noch genügend verlässliche Informationen liefert. Beispielsweise können zusätzlich zu einem Markow-Modell dritter Ordnung je ein weiteres Modell zweiter und erster Ordnung in Reserve gehalten werden. Bei

seltenen oder unbekannten Ausgaben oder Übergängen, wo im Modell dritter Ordnung keine Daten erhältlich sind, kann auf die Reservemodelle mit mehr Information zurückgegriffen werden. (Katz, 1987)

Für einen ausführlicheren Überblick siehe z.B. Manning und Schütze (1999, S. 197ff.).

4 Transformationsbasiertes Parsen

Brill (1993) präsentiert eine neue Ansicht des Parsing-Problems, die viel Ähnlichkeit zur Lösung unseres Problems aufweist. Er stellt die Aufgabe, für einen primitiv geparsten Text automatisch Transformationsregeln zu lernen, welche die ursprüngliche, primitive Syntaxstruktur Schritt für Schritt einer ausgereifteren Syntaxstruktur annähern. Insbesondere sein Vorschlag, diesen Ansatz auf den Output eines Parsers anzuwenden, hebt die Ähnlichkeit zu unserer Aufgabestellung hervor. In beiden Ansätzen werden Syntaxstrukturen angenähert. Dieses Kapitel stellt Brills Ansatz vor.

Die Menge aller gelernten Transformationsregeln nennt Brill, den Regeln sprachbeschreibenden Charakter zuschreibend, eine transformative Grammatik. Die Syntaxstrukturen liegen in Klammerform vor, sodass die Transformationen bestehende Klammern löschen oder neue Klammern einfügen.

Zu den Vorteilen dieser Technik gehören: Die nur lineare Komplexität, die geringe Anzahl der gelernten Transformationen, der Gebrauch eines kleinen Trainingskorpus und die relativ grosse Robustheit. Die Grundstruktur wird vorerst von unbeschrifteten Binärbäumen gebildet. Nur die Tags von Wörtern sind in der Struktur enthalten, keine sonstigen Kategorien, wie Phrasenkategorien oder funktionale Kategorien. Man beachte, dass dies eine vergleichsweise einfache Struktur ist. Später (siehe Abschnitt 4.4 (S. 29)) erweitert er die Struktur um Beschriftungen auf der Nichtterminalebene.

Ein bereits richtig geparstes Korpus wird als Trainingsreferenz vorausgesetzt. Die vier kennzeichnenden Bestandteile des Parsers sind: Der Start-Annotier-Algorithmus, die Menge der zulässigen Transformationen, die Bewertungsfunktion und die Suchstrategie.

4.1 Start-Annotier-Algorithmus

Das Lernprogramm startet in einem naiven Initialzustand, der von der Phrasenstruktur des Zielkorpus weit entfernt ist. Da englische Syntaxbäume dazu neigen, sich nach rechts zu verzweigen, nimmt Brill (1993) als Initialzustand eine binäre Baumstruktur an, die sich rechts weiterverzweigt. Eine Ausnahme bildet die Satzende-Interpunktion, sie wird oben angehängt. Der Satz „The dog and the old cat ate“ würde beispielsweise im Initialzustand so aussehen:

```
((The/DT (dog/NN (and/KJ (the/DT (old/ADJ (cat/NN ate/VBD)))))).)
```

4.2 Menge der zulässigen Transformationen

Es wird eine Folge von Transformationen gelernt, die auf den bisherigen Zustand angewendet wird, um sich der Struktur des Zielkorpus zu nähern. Dazu müssen erst die möglichen Transformationstypen definiert werden. Es ist sinnvoll, möglichst einfache Veränderungen festzulegen, die von möglichst einfachen Bedingungen (engl. *triggering environments*) ausgelöst werden. Die zwölf zulässigen Transformationen sind:

- (1-4) Füge eine öffnende bzw. schliessende Klammer links bzw. rechts des Tags X hinzu.
- (5-8) Entferne eine öffnende bzw. schliessende Klammer links bzw. rechts des Tags X.
- (9-10) Füge eine öffnende bzw. schliessende Klammer zwischen den Tags X und Y hinzu.
- (11-12) Entferne eine öffnende bzw. schliessende Klammer zwischen den Tags X und Y.

Die Durchführung einer Transformation verlangt einige zusätzliche, einfache Umformungen, um die Anzahl der öffnenden und schliessenden Klammern gleich zu halten und Binärbäume zu garantieren.

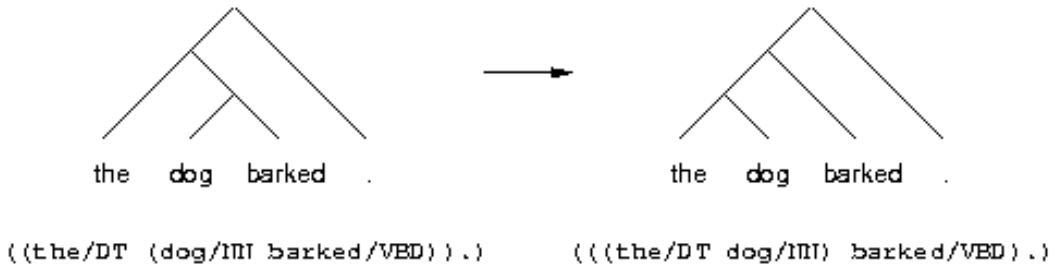


Abbildung 7: Das Beispiel als Baum und als Klammerstruktur

Um z.B. eine schliessende Klammer rechts von einem Nomen einzufügen, finden drei Umformungen nacheinander statt. Die Abbildung 7 zeigt sowohl die ursprüngliche Struktur, als auch die Zielstruktur nach Anwendung der Transformation. Es benötigt dazu vier kleine Schritte:

1. Füge eine schliessende Klammer rechts von NN ein.

2. Füge eine öffnende Klammer links vom Vorgänger von NN ein.

3. Lösche die öffnende Klammer links von NN.

4. Lösche die schliessende Klammer rechts vom Nachfolger von NN.

4.3 Bewertungsfunktion und die Suchstrategie

Jede mögliche Instanz der definierten Transformationstypen wird auf den aktuellen Korpuszustand (am Anfang ist das der Initialzustand) angewendet, die resultierenden Strukturen werden mit denen des Zielkorpus verglichen und ausgewertet. Danach wird die Transformation wieder rückgängig gemacht. Diejenige Transformation, die am meisten Verbesserungen eingebracht hat, wird als Transformation der Folge gespeichert, die gelernt werden soll. Dann wird diese Transformation endgültig angewendet, und es wird wieder von vorn begonnen, bis die Auswirkungen der Transformationen zu keiner substanziellen Verbesserung mehr führen.

Um im Anwendungsfall schliesslich Text zu parsen, der während des Trainings nicht behandelt wurde, wird der Text zuerst in den Initialzustand gebracht. Dann kommt die gelernte Transformationenfolge zum Zuge. Die sukzessive Anwendung jeder Transformation in jeder geeigneten Situation führt schlussendlich zum geparsten Text, d.h. zu den syntaktischen Strukturen.

Nr.	Hinzufügen/Löschen	Klammer	Umgebung
1	Löschen	(	links von NN
2	Löschen	(	links von NNS (plural)
3	Hinzufügen	)	links von ,
4	Löschen	(	zwischen NNP und NNP
5	Löschen	(	rechts von DT
6	Hinzufügen	)	links von ,
7	Löschen	)	links von NNS
8	Löschen	)	zwischen NN und NN
9	Löschen	(	zwischen JJ und JJ (Adjektive)
10	Löschen	(	rechts von $

Tabelle 4: Die ersten zehn Transformationen

Die Tabelle 4 zeigt die ersten zehn Transformationen, die gelernt werden. Sieben davon (1, 2, 4, 5, 7, 8 und 9) extrahieren Nominalphrasen aus dem rechtsverzweigenden Initialzustand. Die Transformationen 3 und 6 lassen sich dadurch erklären, dass ein Komma oft das Ende einer Phrase anzeigt. Die Transformation 10 bindet ein $-Zeichen an die wahrscheinlich nachfolgende Zahl.

4.4 Erweiterungen

Die Aufgabe wird später dadurch erschwert, dass die Bäume um Beschriftungen an jedem Nichtterminalknoten erweitert werden sollen. Auch diese Aufgabe wird transformationsbasiert gelöst. Der Initialannotierer teilt jedem Knoten die Beschriftung *NP* zu. Es gibt nur zwei Transformationstypen:

- Ändere die Knotenbeschriftung nach X, wenn Y eine Tochter des Knotens ist.
- Ändere die Knotenbeschriftung nach X, wenn Y und Z zwei benachbarte Töchter des Knotens sind.

Bei einem weniger naiven Initialzustand verbessern sich die Resultate. Brill (1993) hält es für möglich, dass sich die Werte nochmals verbessern, wenn ausgeklügeltere Transformationstypen definiert werden. Der Ansatz liesse sich leicht in diverse Richtungen verfeinern. Z.B. mit mehr Transformationen oder mit einer anderen Bewertungsfunktion. Das Vorgehen liesse sich auch als Zusatz für andere Grammatikinduktionssysteme verwenden, um die Performanz zu verbessern. Brill hat aber nur noch eine Weiterführung veröffentlicht, in der es um eine effizientere Durchführung geht (siehe Brill und Satta (1996)).

5 Editierdistanz von Baumstrukturen

Die gestellte Aufgabe verlangt unter anderem, einen Syntaxbaum in einen anderen Syntaxbaum umzuwandeln. Dazu müssen erstens die Unterschiede, d.h. die Editierdistanz als Menge von Editieroperationen, zwischen den Baumstrukturen erkannt werden und zweitens die Editieroperationen auf den ersten Syntaxbaum angewendet werden. Wie man die Distanz zwischen Baumstrukturen erkennen kann, wurde bereits erforscht.

Es folgt ein Überblick über die Erforschung des Editierdistanzproblems von Bäumen. Dieses Kapitel behandelt nur jene Aspekte, die für das entwickelte System von Bedeutung sind.

5.1 Terminologie und Notation

Die Editierdistanz ist ein Ähnlichkeitsmass für verschiedene Daten von gleicher Datenstruktur. Sie beantwortet die Frage, welche minimalen Kosten anfallen, um Daten D eines Datentyps in andere Daten D' des gleichen Datentyps mittels bestimmter Editieroperationen zu transformieren. Damit man die Editierdistanz ermitteln kann, müssen

- die Menge O von Editieroperationen: $o(d) \mapsto d'$, wobei gilt: $o \in O$ und $d, d' \in D$, und
- die Kostenfunktion $k : O \to \mathbb{Q}$, welche jeder Operation $o \in O$ einen Kostenwert zuordnet,

definiert sein.

Es gibt unendlich viele Folgen $S = s_1, s_2, \ldots, s_n$ von Editieroperationen, die D in D' umwandeln. Die Kosten $K(S)$ einer Folge S ist die Summe aller Kostenwerte (siehe Gleichung 5.1). Um die Editierdistanz $E(D, D')$ zu ermitteln, muss die billigste aller möglichen Folgen von Editieroperationen berechnet werden (siehe Gleichung 5.2).

$$K(S) = \sum_{i=1}^{n} k(s_i) \tag{5.1}$$

$$E(D, D') = \min K(S) \tag{5.2}$$

In den letzten zwei Jahrzehnten wurde intensiv nach immer effizienteren Algorithmen für die Lösung des Baumkorrekturproblems für geordnete Bäume gesucht. Im entwickelten System steht die Effizienz nicht im Vordergrund. Deshalb liegt der Schwerpunkt dieses

Kapitels jeweils nicht auf der Effizienzsteigerung durch diese oder jene Neuerung im Algorithmus, sondern auf den Definitionen, dem grundsätzlichen Vorgehen und allgemeinen Überlegungen. Beispielsweise sind die Definitionen der Grundtransformationen von Interesse, oder die Art, wie ein Baum durchlaufen wird.

5.2 Grundlagen von Tai

Tai (1979) präsentiert einen Algorithmus, der die Editierdistanz zweier Bäume berechnet. Die grundlegende Datenstruktur, von der er ausgeht, sind gerichtete, geordnete und beschriftete Bäume.

Die Editieroperationen definiert er folgendermassen:[5]

- die Beschriftung eines Knotens durch eine andere **ersetzen**,
- einen Knoten K **löschen**, indem alle Kinder von K zur Mutter von K hinaufgehängt werden und
- einen Knoten K unter den Knoten M **einfügen**, indem optional eine zusammenhängende Reihe von Kindern von M zu Kindern von K werden.

T sei ein Baum. $|T|$ ist die Anzahl Knoten von T. Mit der Zuordnung einer eindeutigen Nummer an jeden Knoten eines Baumes T während eines Preorder-Durchlaufs erhält der Baum eine lineare Komponente. Ein Knoten, der in der Preorder-Position i des Baumes T steht, heisst $T[i]$.[6] Es wird ein Nullknoten Λ eingeführt. Eine Editieroperation notiert Tai als $a \rightarrow b$, wobei a und b jeweils die Beschriftung eines Knotens von T oder Λ ist. $a \rightarrow b$ ist eine Ersetzungsoperation, wenn $a \neq \Lambda$ und $b \neq \Lambda$, eine Löschoperation, wenn $a \neq \Lambda$ und $b = \Lambda$ oder eine Einfügeoperation, wenn $a = \Lambda$ und $b \neq \Lambda$. T' sei der Baum, der entsteht, wenn die Editieroperation $a \rightarrow b$ auf T angewandt wird. Er notiert dies als $T \Rightarrow T'$ via $a \rightarrow b$.

$S = s_1, s_2, \ldots s_m$ ist eine Folge von Editieroperationen. S transformiert den Baum T zum Baum T', wenn es eine Folge von Bäumen $T_0, T_1, \ldots, T_m$ gibt, für die gilt, dass $T = T_0$, $T' = T_m$ und $T_{i-1} \Rightarrow T_i$ via s_i für $1 \leq i \leq m$.

Um das Problem der Ermittlung der Editierdistanz zu vereinfachen, führt Tai (1979) den Begriff der Zuordnung (engl. *mapping*) ein. Eine Zuordnung ist eine kompakte Art, die Information zu speichern, wie eine Folge von Editieroperationen T in T' umwandelt, ohne

[5] Dies tut er in Analogie zu Wagner und Fischer (1974) in der Zeichenkettentransformation.

[6] Die verschiedenen Notationen der Literatur wurden der Übersichtlichkeit halber vereinheitlicht.

dass die Reihenfolge eine Rolle spielt. In der Abbildung 8 sind zwei Bäume T und T' und deren Zuordnung als gestrichelte Linien dargestellt.

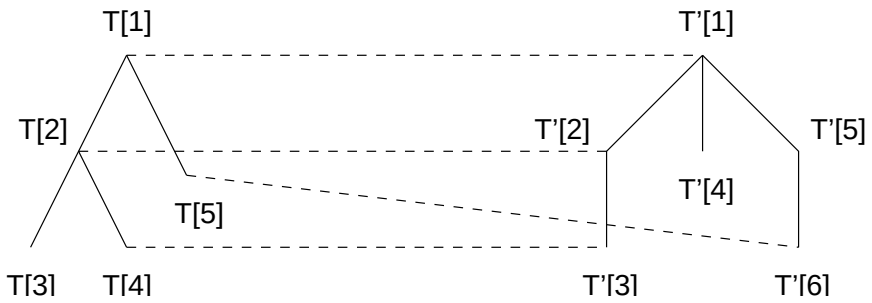


Abbildung 8: Zuordnung von Baumknoten

Eine gestrichelte Linie von einem Knoten $T[i]$ zu einem Knoten $T'[j]$ bedeutet, dass die Beschriftung von $T[i]$ durch die Beschriftung von $T'[j]$ ersetzt werden muss, falls $i \neq j$, dass die Beschriftung von $T[i]$ unverändert bestehen bleibt, falls $i = j$. Jeder Knoten $T[i]$, der nicht von einer gestrichelten Linie berührt wird, muss gelöscht werden und jeder Knoten $T'[j]$, der nicht berührt wird, muss eingefügt werden.

Formal wird eine Zuordnung von T und T' als Tripel $\langle Z, T, T' \rangle$ definiert. Z ist eine Menge von Paaren von ganzen positiven Zahlen $\langle i, j \rangle$, wobei gilt, dass

1. $1 \leq i \leq |T|$ und $1 \leq j \leq |T'|$;
2. Für alle zwei Paare $\langle i_1, j_1 \rangle$ und $\langle i_2, j_2 \rangle$ in Z gilt:
   (a) $i_1 = i_2$ genau dann, wenn $j_1 = j_2$, d.h. jeder Knoten von T und T' darf von höchstens einer Linie berührt werden;
   (b) $i_1 < i_2$ genau dann, wenn $j_1 < j_2$;
   (c) $T[i_1]$ dominiert bzw. wird dominiert von $T[i_2]$ genau dann, wenn $T[j_1]$ $T[j_2]$ dominiert bzw. von $T[j_2]$ dominiert wird.

Jedes Paar $\langle i, j \rangle$ in Z wird als gestrichelte Linie interpretiert, die $T[i]$ und $T[j]$ berührt. Die Bedingungen 2b und 2c stellen sicher, dass nach dem Löschen sämtlicher unberührter Knoten aus T und T' die beiden Strukturen isomorph sind. Die Zuordnung, wie sie Tai (1979) definiert, erhält also Strukturen von T und T', die einander gleichen.

Aus dem Beispiel in der Abbildung 8 ergibt sich die Zuordnung $Z = \{\langle 1, 1 \rangle, \langle 2, 2 \rangle, \langle 4, 3 \rangle, \langle 5, 6 \rangle\}$, indem die Start- und Zielpunkte der gestrichelten Linien notiert werden. I und J

seien die Mengen aller unberührter Knoten von T bzw. T'. Der einzige unberührte Knoten von T ist $T[3]$: $I = \{3\}$. Also muss $T[3]$ gelöscht werden. Die Menge der unberührten Knoten von T' ist $J = \{4, 5\}$, also müssen diese eingefügt werden. Die oben genannten Bedingungen (S. 32) sind anhand der Abbildung 8 auf der vorherigen Seite leicht zu verifizieren: Alle Linien treffen je einen Knoten eines Baumes (siehe Bedingung 1 auf der vorherigen Seite). Kein Knoten hat mehr als eine Zuordnungs-Linie (siehe Bedingung 2a auf der vorherigen Seite). Es gibt keine Zuordnungs-Linien, die sich überkreuzen (siehe Bedingungen 2b auf der vorherigen Seite und 2c auf der vorherigen Seite). Wenn man sich im Beispiel alle Knoten, die nicht von einer Linie berührt werden, wegdenkt, dann ergibt sich Isomorphie.

Ist Z eine Zuordnung von T nach T' und I bzw. J die Mengen aller unberührten Knoten von T bzw. T', kann die Kostenfunktion $cost(Z)$ von Z ganz einfach berechnet werden. Die Kosten ergeben sich aus der Summierung aller Ersetzungs-, Einfüge- und Löschoperationen:

$$cost(Z) = \sum_{\langle i,j\rangle \in Z} k(T[i] \to T'[j]) + \sum_{i \in I} k(T[i] \to \Lambda) + \sum_{j \in J} k(\Lambda \to T'[j]) \tag{5.3}$$

Der genaue Algorithmus von Tai (1979), wie er minimale Zuordnungen berechnet und so die Kosten optimiert, ist hier nicht relevant und kann im Original nachgelesen werden.

5.3 Erweiterungen des Ansatzes von Tai

Zhang und Shasha (1989) entwickeln den Algorithmus von Tai weiter. Das Hauptziel ist es, den Algorithmus effizienter zu gestalten. Unter anderem nummerieren sie die Knoten in einem Postorder-Durchlauf, damit weniger Speicherplatz benötigt wird.

Barnard u. a. (1995) liefern eine schöne Zusammenfassung der älteren Ansätze und passen den Ansatz von Zhang und Shasha (1989) auf Dokumentenbäume an, indem sie vier zusätzliche Operationen einführen, welche vom üblichen Vorgehen beim Editieren eines Dokuments inspiriert sind:

- **Zusammenfügen** (*merge*) zweier Abschnitte
- **Aufteilen** (*split*) eines Abschnitts in zwei
- **Vertauschen** (*permute*) von Text unter einer Struktur
- **Bewegen** (*move*) eines Abschnitts an eine andere Position.

Diese zusätzlichen Operationen können in den meisten Fällen aus den schon bekannten primitiven Operationen zusammengesetzt werden.[7] Barnard u. a. (1995) schlagen vor, diese komplexen Operationen erst in einem nachgeschalteten Verarbeitungsschritt in der Folge der einfachen Operationen zu lokalisieren. Sie stellen dabei fest, dass bei Vertausch- oder Bewege-Aktionen unter Umständen weitere primitive Operationen nützlich wären, wie:

- ganzen **Teilbaum löschen** (*deleteTree*)
- ganzen **Teilbaum einfügen** (*insertTree*)

Chawathe u. a. (1996) stellen fest, dass im Bereich von Dokumentenvergleichen[8] eine zusätzliche Operation nützlich ist, nämlich das Bewegen von Knoten (*move*). Sie übernehmen die Zuordnungsidee von Tai (1979), formulieren aber die Bedingungen, ob zwei Knoten einander zugeordnet werden können, anders. Chawathe u. a. (1996) definieren folgende Editieroperationen:

- Ein Blattknoten N kann als n-tes Kind eines Knoten M eingefügt werden.
- Ein Blattknoten N kann gelöscht werden.
- Die Beschriftung L eines Knotens N kann ersetzt werden.
- Ein Teilbaum S kann von einem Elternknoten zu einem anderen Knoten N in T **bewegt** und zwischen Ns Kinder eingereiht werden.

Das Kostenmodell ist gleich aufgebaut wie bisher. Chawathe u. a. (1996) teilen die Aufgabe, die Editierdistanz zu berechnen, in zwei Teilaufgaben. Die erste Teilaufgabe ist, eine Zuordnung der beiden Bäume T und T' zu finden. Anhand der Zuordnung kann dann in der zweiten Teilaufgabe die minimale Folge der Editieroperationen ermittelt werden. Um die zweite Teilaufgabe zu lösen, müssen Chawathe u. a. (1996) mehr Aufwand in Kauf nehmen, als Tai (1979), weil sie auf einer Folge von Operationen beharren, und die Reihenfolge als wesentlich betrachten. Sie verändern den Baum T in fünf verschiedenen Phasen. Ihr Algorithmus führt alle Editieroperationen unmittelbar bei der Entdeckung auf einer Kopie des Ursprungsbaums T durch.

[7] Obwohl die Ersetz-Operation intuitiv wohl als einfache Transformation gilt, könnte sie auch als eine komplexe Operation betrachtet werden, da ihr Resultat auch mit einer Folge von Löschen und Einfügen erreicht werden kann.

[8] Um ihren Algorithmus zu testen, entwickeln sie ein interessantes System namens *LaDiff*, welches die Änderungen von LaTeX-Dokumenten darstellt.

6 Anwenden von Markow-Modellen

Wie wichtig das Modell für das entwickelte System ist, wurde bereits bei der theoretischen Einführung der Markow-Modelle erwähnt. In diesem Kapitel sind drei Anwendungen von Markow-Modellen zusammengestellt. Die Verwendungsmöglichkeiten und die Kodierungen dieser Modelle in sprachverarbeitenden Systemen sollen beispielhaft gezeigt werden. Das probabilistische Tagging wird im Vordergrund stehen. Zwei Anwendungen zum partiellen Parsen erweitern den Tagging-Ansatz.

6.1 Tagging mit Markow-Modellen

Da das Tagging mit Markow-Modellen als Vorbild für die Konzipierung unseres Systems gedient hat, wurde es für nützlich erachtet, diese probabilistische Anwendung der Verarbeitung der natürlichen Sprache hier zu besprechen.

Der Vorgang, Wörter eines Textes mit Wortarten zu klassifizieren, d.h. den Wörtern eindeutige Tags zuzuweisen, wird Tagging[9] genannt. Tagging wurde bisher mit vielen unterschiedlichen Methoden angegangen, beispielsweise transformationsbasiert (Brill, 1994), mit dem Gebrauch von Lexika und auch probabilistisch.

Die grundlegende Idee des probabilistischen Ansatzes besteht darin, Wörter und Tags in der Trainingsphase mit einem Visible-Markow-Modell zu modellieren, da Wörter und Tags in der Trainingsphase bekannt sind. Dieses Visible-Markow-Modell wird in der Anwendungsphase aber als Hidden-Markow-Modell behandelt, wenn es darum geht, von bekannten Wörtern auf die unbekannten Tags zu schliessen.

6.1.1 Modell kodieren

Wird ein Tagger als Markow-Modell kodiert, repräsentieren jeweils

- die Zustände Tags
- die Ausgaben Wörter
- die Übergangswahrscheinlichkeiten die kontextuellen Wahrscheinlichkeiten, also das direkte Aufeinanderfolgen von Tags,
- die Ausgabewahrscheinlichkeiten die lexikalischen Wahrscheinlichkeiten, d.h. das gleichzeitige Auftreten von Wort und Tag.

[9] Tagging von engl. *part-of-speech tagging*

Übergänge zwischen Zuständen sind n-Gramme von Tags, also etwa Bigramme oder üblicherweise Trigramme. Ungetaggter Text wird als die beobachtete Ausgabefolge behandelt, aufgrund derer die wahrscheinlichste Tagfolge berechnet werden kann.

Es sei $\mathcal{T}$ die Menge aller Tags und V die Menge aller Wörter. Die Wortfolge $W = w_1 \dots w_k \in V^*$ ist im probabilistischen Tagging direkt im Korpustext (z.B. in einem Satz) gegeben. Gesucht wird die entsprechende wahrscheinlichste Tagfolge $T = t_1 \dots t_k \in \mathcal{T}^*$ des Satzes:

$$\operatorname*{argmax}_T P(T \mid W) = \operatorname*{argmax}_T P(T)P(W \mid T). \tag{6.1}$$

Für jedes Paar $\langle w, t \rangle \in V \times \mathcal{T}$ kann die lexikalische Wahrscheinlichkeit $P(w \mid t)$ aus dem Korpus berechnet werden. Auch die kontextuelle Wahrscheinlichkeit $P(t_n \mid t_1 \dots t_{n-1})$ ist implizit für jedes n-Tupel $\langle t_1 \dots t_n \rangle \in \mathcal{T} \times \dots \times \mathcal{T}$ im Korpus enthalten. Diese beiden Wahrscheinlichkeiten können dazu verwendet werden, die gesuchten Wahrscheinlichkeiten zu approximieren. Die Approximation beruht auf zwei Abhängigkeitsannahmen: Erstens, dass die Wortart eines Wortes nur von den vorhergehenden $n-1$ Tags abhängt. „[Man nimmt] an, dass die natürliche Sprache ein Markow-Prozess der Ordnung $n-1$ ist, was zwar nicht wahr ist, aber eine erfolgreiche Approximation. “ [Eigene Übersetzung] (Brants und Samuelsson, 1995, S. 2)

$$\begin{aligned} P(T) &= P(t_1)P(t_2 \mid t_1)P(t_3 \mid t_1, t_2) \dots P(t_k \mid t_1 \dots t_{k-1}) \\ &\approx \prod_{i=1}^{k} P(t_i \mid t_{i-n+1} \dots t_{i-1}) \end{aligned} \tag{6.2}$$

Die zweite Annahme lautet, dass das Wort w_i nur vom Tag t_i abhängt:

$$P(W \mid T) \approx P(w_1|t_1)P(w_2|t_2) \cdots P(w_k|t_k) \tag{6.3}$$

Die Gleichung 6.2 bedingt, dass zusätzliche erste Tags $\langle t_{-n+2} \dots t_0 \rangle$ eingeführt werden, welche den Beginn der Tagfolge anzeigen. In Analogie zu den Startzuständen in Markow-Modellen.

Aus den Gleichungen 6.3 und 6.2 folgt, dass die gemeinsame Wahrscheinlichkeit, dass eine Wortfolge W zusammen mit der Tagfolge T auftritt, annäherungsweise das Produkt

aus der lexikalischen und der kontextuellen Wahrscheinlichkeit ist:

$$P(W,T) = P(T)P(W \mid T) \approx \prod_{i=1}^{k} P(t_i \mid t_{i-n+1} \ldots t_{i-1}) P(w_i \mid t_i). \tag{6.4}$$

Die beste Tagfolge T für eine gegebene Wortfolge W kann also mit

$$\underset{t_1 \ldots t_k}{\text{argmax}} \prod_{i=1}^{k} P(t_i \mid t_{i-n+1} \ldots t_{i-1}) P(w_i \mid t_i) \tag{6.5}$$

berechnet werden. Die Gleichung 6.4 beschreibt ein n-Gramm-Modell von Tags. Ein solches n-Gramm-Modell ist identisch zu einem Markow-Modell der Ordnung $(n-1)$.

6.1.2 Parameter bestimmen

Um ein Modell zu erstellen, muss man die Ausgabe- und Übergangswahrscheinlichkeiten bestimmen. Die lexikalischen Wahrscheinlichkeiten $P(w_i \mid t_i), w_i \in V, t_i \in \mathcal{T}$ und die kontextuellen Wahrscheinlichkeiten $P(t_i \mid t_{i-n+1} \ldots t_{i-1}), t_j \in \mathcal{T}$ müssen dazu berechnet werden.

Diese Aufgabe kann durch Untersuchen eines genügend grossen, annotierten Korpus gelöst werden. Man zählt dazu die Anzahl $f(w,t)$ jedes Paares $\langle w,t \rangle \in V \times \mathcal{T}$ und die Anzahl $f(t_1, \ldots, t_n)$ jedes n-Tupels $\langle t_1 \ldots t_n \rangle \in \mathcal{T} \times \cdots \times \mathcal{T}$. Die Maximum-Likelihood-Schätzung erfolgt mit diesen relativen Häufigkeiten:

$$\hat{P}(w_i \mid t_i) = \frac{f(w_i, t_i)}{\sum_{w \in V} f(w, t_i)} \tag{6.6}$$

$$\hat{P}(t_i \mid t_{i-n+1} \ldots t_{i-1}) = \frac{f(t_{i-n+1} \ldots t_i)}{\sum_{t \in \mathcal{T}} f(t_{i-n+1} \ldots t_{i-1} t)} \tag{6.7}$$

Dies maximiert die Wahrscheinlichkeit, genau diese Häufigkeit zu beobachten.

Es gibt auch Möglichkeiten, die Parameter ohne den Gebrauch eines annotierten Korpus zu bestimmen. Ein Training mit einem annotierten Korpus ergibt aber im Allgemeinen bessere Resultate (Elworthy, 1994).

6.2 Partielles Parsen mit Markow-Modellen

Die folgenden Ansätze arbeiten mit Markow-Modellen und zeigen, dass diese Modelle prinzipiell auch geeignet sind, um Syntaxstrukturen als Ausgabe zu modellieren.

Eine vollständige syntaktische Analyse (Parsen) ermittelt einen kompletten Syntaxbaum für einen Satz aufgrund einer Grammatik. In einigen sprachverarbeitenden Anwendungen ist es gar nicht nötig, eine vollständige Syntaxanalyse zu haben. In diesen Fällen reicht es, wenn eine flache syntaktische Analyse (auch partielles Parsen) durchgeführt wird. Je nach der angestrebten strukturellen Tiefe werden einfache Chunks bis komplexere Phrasen gebildet.

Die folgenden zwei Parsingansätze können nicht in jedem Fall garantieren, dass eine vollständige Syntaxanalyse resultiert. Deshalb fallen sie unter partielles Parsen. Aber die Komplexität der Strukturen übersteigt die von einfachen Chunks wesentlich. Beide Systeme sind im Zusammenhang mit dem NEGRA-Projekt entstanden.

6.2.1 Partielles Parsen in Analogie zum Tagging

Skut (1999) untersucht die Fähigkeit von Markow-Modellen, syntaktische Strukturen zu erkennen und zeigt, dass es durchaus möglich ist, auch komplexere Strukturen aufzubauen.

Einfacher Chunk-Tagger Als Grundlage beschreibt Skut (1999) einen Chunk-Tagger-Ansatz mit Markow-Modellen, indem Klammern innerhalb von Folgen von Tags platziert werden, um einzelne Chunks zu markieren.

Wird ein solcher Chunk-Tagger als Markow-Modell kodiert, repräsentieren

- die Zustände eine Kodierung der Klammerinformation $S = S_1, \ldots, S_n$
- die Ausgaben Tags $T = t_1, \ldots, t_n$
- die Übergangswahrscheinlichkeiten das unmittelbare Aufeinanderfolgen von Klammerungen
- die Ausgabewahrscheinlichkeiten das gemeinsame Auftreten von Tag und Klammerinformation.

Übergänge zwischen Zuständen sind n-Gramme von Klammerinformationen. Die Tagfolge eines getaggten Textes wird als die beobachtete Ausgabefolge betrachtet, aufgrund derer die wahrscheinlichste Zustandsfolge, welche der Klammerinformation entspricht, berechnet werden kann.

Diese noch nicht spezifizierte Klammerinformation ermöglicht eine minimale Kodierung von Chunks auf erster Stufe. Um die Klammerinformation zu kodieren, wird eine Funktion $r : i \rightarrow [0, 1]$ deklariert:

$$r_i = \begin{cases} 1 & \text{falls bei } t_i \text{ ein neuer Chunk beginnt oder } t_i \text{ zu keinem Chunk gehört} \\ 0 & \text{andernfalls} \end{cases} \tag{6.8}$$

Diese Funktion verteilt also Wahrheitswerte, ob bei t_i ein neuer Chunk beginnt oder ob t_i zu keinem Chunk gehört, oder anders formuliert, ob t_i und t_{i-1} nicht in einem Schwesternverhältnis zueinander stehen. Somit weiss man, dass bei einem Tag t_i mit $r_i = 1$ ein Chunk beginnt, welcher genau vor dem nächsten $r_m = 1$ (also bei t_{m-1}) wieder aufhört. Diese Information allein würde nicht für ein leistungsfähiges Markow-Modell ausreichen, deshalb nimmt Skut das Tag auch noch in die Klammerinformation hinein. So ergibt sich dann als Klammerinformation das Paar $S_i = \langle r_i, t_i \rangle$.

Für die Berechnung der wahrscheinlichsten Klammerinformationsfolge kann man den für das Wortarten-Tagging üblichen Viterbi-Algorithmus (Lager, 1997) verwenden. Dieser Chunk-Tagger findet Begrenzungen von einfachen, d.h. nicht verschachtelten, Nominal-, Präpositional- und Adjektivphrasen.

Erweiterter Chunk-Tagger Der einfache Chunk-Tagger lässt sich verallgemeinern, sodass er auch rekursive Strukturen bis zu einer bestimmten Tiefe erkennt. Der Autor hat sich zusätzlich die Bedingungen gesetzt, dass sowohl die Berechnung des wahrscheinlichsten Zustandsfolge als auch die Parameter-Bestimmung analog zum Wortarten-Tagging mit Standard-Methoden durchgeführt werden müssen.

Im Markow-Modell für einen erweiterten Chunk-Tagger repräsentieren

- die Zustände sogenannte Chunk-Tags $S = S_1, \ldots, S_n$
- die Ausgaben Wörter $W = w_1, \ldots, w_n$ zusammen mit ihren Tags $T = t_1, \ldots, t_n$
- die Übergangswahrscheinlichkeiten das Aufeinanderfolgen von Chunk-Tags
- die Ausgabewahrscheinlichkeiten das gemeinsame Auftreten von Wort/Tag mit einem Chunk-Tag.

Übergänge zwischen Zuständen sind n-Gramme von Chunk-Tags. Getaggter Text wird als die beobachtete Ausgabefolge betrachtet, aufgrund derer die wahrscheinlichste Chunk-Tag-Folge berechnet werden kann.

Skut (1999) definiert ein endliches Alphabet von Chunk-Tags, welches die folgende Information kodiert:

- Tag, also die Wortart
- Syntaktische Kategorie des Mutterknotens
- Strukturelles Verhältnis des Wortes w_i zum unmittelbar vorhergehenden Wort w_{i-1}.

Das strukturelle Verhältnis ist eine Funktion, welche die folgenden Werte annimmt:

(6.9)
$$r_i = \begin{cases} 0 & \text{falls die Mutter von } t_i \text{ = Mutter von } t_{i-1} \\ + & \text{falls die Mutter von } t_i \text{ = Grossmutter von } t_{i-1} \\ ++ & \text{falls die Mutter von } t_i \text{ = Urgrossmutter von } t_{i-1} \\ - & \text{falls die Grossmutter von } t_i \text{ = Mutter von } t_{i-1} \\ -- & \text{falls die Urgrossmutter von } t_i \text{ = Mutter von } t_{i-1} \\ = & \text{falls die Grossmutter von } t_i \text{ = Grossmutter von } t_{i-1} \\ 2 & \text{falls } w_i \text{ eine Satzendeinterpunktion ist} \\ 1 & \text{andernfalls} \end{cases}$$

Sollten mehrere Verhältnisse auf ein Tag zutreffen, wird das erste Symbol genommen, um keine unnötige Ambiguität im Modell einzubauen.

Beispiele für solche strukturellen Verhältnisse zwischen Knoten sind in der Abbildung 9 auf der nächsten Seite dargestellt.

Gesucht ist analog zur Wortarten-Tagging-Aufgabe:

(6.10)
$$\underset{S}{\text{argmax}} P(S \mid T) = \underset{S}{\text{argmax}} P(S)P(T \mid S)$$

Jede Folge von Chunk-Tags repräsentiert die syntaktische Struktur für die gesamte Wortfolge. Die seltenen Fälle, dass der erweiterte Chunk-Tagger nicht definitionsgemässe Bäume aufbaut, wenn beispielsweise $r_1 \neq 1$ ist, oder falls inkonsistente Beschriftungen vorgeschlagen werden, können leicht in einem nachgeschalteten Schritt korrigiert werden.

Auch dieser erweiterte Chunk-Tagger funktioniert, so wie er von Skut (1999) präsentiert wird, nur, wenn flache Syntaxstrukturen erwünscht sind. Im Kontext des NEGRA-Projekts ist diese Eigenschaft des Chunk-Taggers gegeben, da NEGRA flache Strukturen bevorzugt

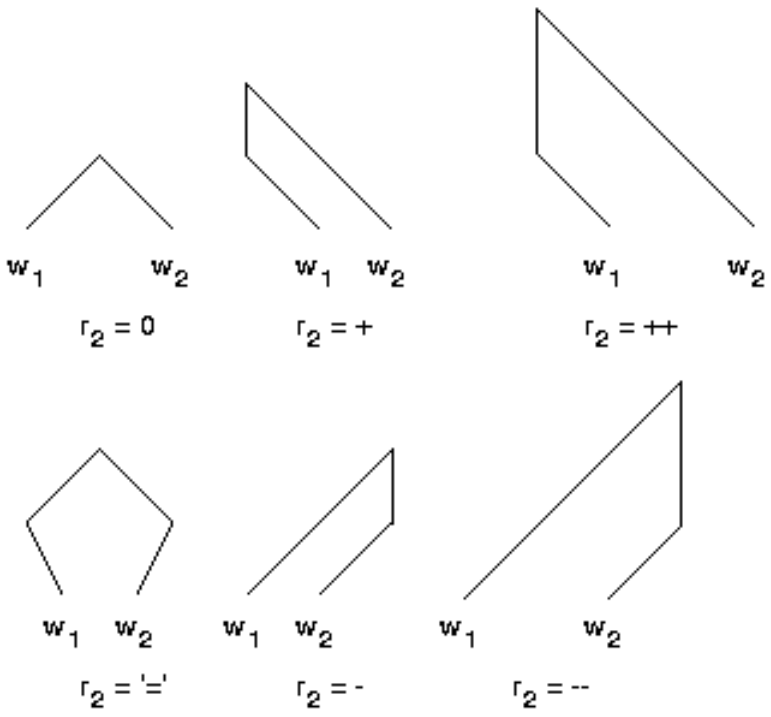


Abbildung 9: r_2 von w_2

(siehe Ausführungen S. 12). Der Chunk-Tagger liesse sich vielleicht auch auf etwas tiefere Strukturen erweitern, indem die strukturellen Verhältnisse ausgebaut würden.

6.2.2 Partielles Parsen mit kaskadierten Markow-Modellen

Brants (1999b) lässt hierarchische Strukturen mithilfe von kaskadierten Markow-Modellen erzeugen.

Die Grundidee ist es, einen Syntaxbaum Schicht für Schicht zu erstellen. Für jede Schicht bestimmt ein probabilistischer Chartparser Phrasenstrukturhypothesen. Aus diesen Hypothesen filtert ein Markow-Modell die besten Phrasenstrukturen heraus. Diese besten Phrasenstrukturen sind dann die Eingabe der nächsthöheren Schicht, welche die Strukturen wiederum zu neuen Phrasenstrukturhypothesen zusammenfasst und von einem neuen Markow-Modell filtern lässt.

In den Markow-Modellen repräsentieren je nach Schicht

- die Zustände Wortarten-Tags (Schicht 0) oder Phrasenkategorien
- die Ausgaben Wörter (Schicht 0) oder Wortarten-Tags bzw. Phrasenstrukturen.

Als Erweiterung des Taggings mit Markow-Modellen (siehe Abschnitt 6.1 (S. 35)) können Zustände hier auch nichtterminale Kategorien repräsentieren. Diese Art von Zuständen emittieren nicht nur Wörter, sondern auch partielle Syntaxbäume (Phrasenstrukturen).

Der Verarbeitungsprozess beginnt mit einer Folge von Wörtern. Die Folge muss nicht unbedingt einen Satz bilden, es kann auch eine Nominalphrase oder sonst eine Phrase sein.

Das System beginnt in der Schicht 0 mit einem Markow-Modell, welches das Tagging erledigt. In den nächsthöheren Schichten produziert ein probabilistischer Chartparser alle möglichen Phrasenstrukturhypothesen über eine Stufe. Der Chartparser verwendet dazu eine Grammatik mit stochastischen kontextfreien Regeln, die aus einem Korpus gelernt wurden. Dann werden die besten Pfade der Chart unter Zuhilfenahme eines Markow-Modells berechnet, welches genau für diese Schicht erstellt wurde.

Das System lässt in jeder Schicht eine vordefinierte Ambiguität zu und disqualifiziert alle Hypothesen, deren Verlässlichkeit unter einem bestimmten Wert liegt. Die gefilterten Phrasenstrukturhypothesen einer Schicht dienen der nächsthöheren Schicht als Eingabe. Die Anzahl der Schichten ist von vornherein festgelegt und jede Schicht wird von einem separaten Markow-Modell behandelt.

Indem mehrere Hypothesen an die nächste Schicht weitergegeben werden, kann das Modell diejenige Hypothese bestätigen, welche am besten in den Kontext der Schicht passt,

auch wenn es sich nicht unbedingt um die beste Hypothese der unteren Schicht handelt. Dieser Mechanismus erlaubt eine Zusammenarbeit zwischen Schichten: Die Hypothesen werden bottom-up ausgesucht und gefiltert, während die letzte Entscheidung top-down gefällt wird.

Wenn n Schichten bzw. Markow-Modelle verwendet werden, können nur Strukturen bis zu einer Tiefe n erkannt werden, alle Phrasen in der Schicht n bleiben so stehen und werden nicht mehr zu höheren Phrasen zusammengefasst. Die beste Hypothese (bzw. eventuelle Folgen von besten Hypothesen) der obersten Schicht ist die erkannte Struktur.

Je nach der Anzahl Schichten kann auch eine grössere Tiefe der aufgebauten Struktur erreicht werden. Die Komplexität der Syntaxstruktur kann so weit über die von einfachen Chunks hinausgehen.

Teil II

System

7 Systemüberblick

Das Ziel dieses Kapitels liegt in der überblicksartigen Darstellung dessen, was das System tut. Es soll also die grundsätzliche Konzipierung des Systems wiedergegeben werden. Wie die einzelnen Teilaufgaben gelöst werden, wird dann in den nächsten zwei Kapiteln behandelt.

Als erstes wird der Zusammenhang zu den im ersten Teil dieser Arbeit präsentierten Ansätzen hergestellt. Darauf folgen die Modellbeschreibung, die Architektur des Systems und die Auswirkungen des Systems. Zum Schluss steht dann die Abgrenzung des Systems von den vorgestellten Arbeiten.

7.1 Bereiche

In dieser Arbeit wird die Aufgabe gestellt, ein System zu entwerfen und zu implementieren, welches ermöglicht, Syntaxbäume einer ersten Art Syntaxbäumen einer anderen Art automatisch anzunähern.

Diese Aufgabe zerfällt in zwei Bereiche:

Distanzbereich: Es müssen Syntaxbäume transformiert werden. Diese Teilaufgabe setzt das Ermitteln der Editierdistanz voraus. Im Kapitel 5 (S. 30) sind verschiedene Ansätze zur Baumtransformation beschrieben. Die Editierdistanz benötigt immer eine Spezifikation der Editieroperationen. Ein Kostenmodell ordnet jeder Operation einen Kostenwert zu. Als Hilfskonstruktion wird meistens ein Abbildungsalgorithmus verwendet, der die ähnlichen Strukturen der zwei Bäume festhält. Die Editierdistanz ist die billigste Folge von Editieroperationen, die den Ausgangsbaum in den Zielbaum verwandelt.

Probabilistischer Bereich: Die Transformation soll automatisch geschehen, ob ein Zielbaum vorhanden ist oder nicht. Die Idee ist es, analog zum Tagging mit einem Markow-Modell, eine Klassifizierungsaufgabe zu lösen (siehe Kapitel 3 (S. 16) und 6 (S. 35)). Es wird angenommen, dass zwischen der lokalen Struktur, die einen Knoten umgibt, also irgendeiner Art von Knoten-Kontext, und der Transformation, die an diesem Knoten vorgenommen wird, ein Zusammenhang besteht, der – ähnlich wie der Zusammenhang von Wort und Wortart – von einem Markow-Modell repräsentiert werden kann. Bei gegebener lokaler Knoten-Struktur im Baum wird die geeignete Transformation für diesen Knoten gesucht. Im Unterschied zum Tagging mit

Markow-Modellen, bei dem die Tags im Referenzkorpus schon gegeben sind, muss in diesem Fall erst die erforderliche Menge von Transformationen im Distanzbereich der Aufgabe gefunden werden.

Der Distanzbereich bereitet einerseits die Grundlagen für den probabilistischen Bereich vor und wendet andererseits die Resultate des probabilistischen Bereichs an. So werden die beiden Bereiche gestaffelt verwendet.

Editieroperationen, die denselben Knoten bearbeiten, bilden eine Transformation. Die gesamte Folge von Transformationen für einen Baum wird hier oft auch Distanz genannt.

7.2 Markow-Modelle für Baumtransformationen

Das Markow-Modell steht im Zentrum des Systems. Obwohl ein Modell die Realität abbilden soll, darf die Spezifität eines probabilistischen Modells nicht zu gross sein, das Modell muss eine Abstraktion ermöglichen, damit allgemein gültige Informationen daraus abgeleitet werden können.

Im Baumtransformationsmodell repräsentieren

- die Zustände Transformationen
- die Ausgaben Knoten-Kontexte
- die Übergangswahrscheinlichkeiten das unmittelbare Aufeinanderfolgen von Transformationen
- die Ausgabewahrscheinlichkeiten das gleichzeitige Auftreten von Transformationen und Knoten-Kontexten
- die Zeitpunkte Knoten im Baum in Postorder-Reihenfolge.

Die Knoten-Kontexte enthalten Informationen über die lokale Baumstruktur um einen Knoten.

Die Zeitkomponente des Markow-Modells bildet den Postorder-Baumdurchlauf ab. Jeder Zeitpunkt betrifft einen Knoten im Baum. Die Transformationen und die Kontexte sind somit durch den Baumdurchlauf synchronisiert: Eine Transformation sendet einen Kontext aus und dies in jedem Knoten des Baumes einmal. Es handelt sich beim Modell um ein Markow-Modell erster Ordnung.

Übergänge zwischen Zuständen sind Bigramme von Transformationen, d.h. die Transformationen betreffen Baumknoten, welche im Baumdurchlauf unmittelbar aufeinander folgen. Die Ausgabewahrscheinlichkeiten entsprechen den Wahrscheinlichkeiten, dass ein Kontext und eine Transformation zusammen bei einem Knoten auftreten. Die Folge von Knoten-Kontexten eines Baumes wird als die beobachtete Ausgabefolge betrachtet, aufgrund derer die wahrscheinlichste Transformationsfolge berechnet werden kann.

Obschon klar ist, dass das Modell Eigenschaften besitzt, welche in der Realität nicht genau so gegeben sind, wird angenommen, dass das Modell eine sinnvolle Annäherung ist.

Es sei nochmals erwähnt, dass alle Editieroperationen, die denselben Knoten bearbeiten, eine Transformation bilden. Eine Distanz ist die Folge aller Transformationen, die denselben Baum betreffen. Die Distanz ist im Modell implizit enthalten, sogenannte START-Symbole repräsentieren den Beginn eines neuen Baumdurchlaufs. Die Editieroperationen und Kontexte werden im Kapitel 8.1 (S. 53) bzw. 8.4 (S. 63) ausführlich behandelt.

7.3 Überblick zur Architektur des Systems

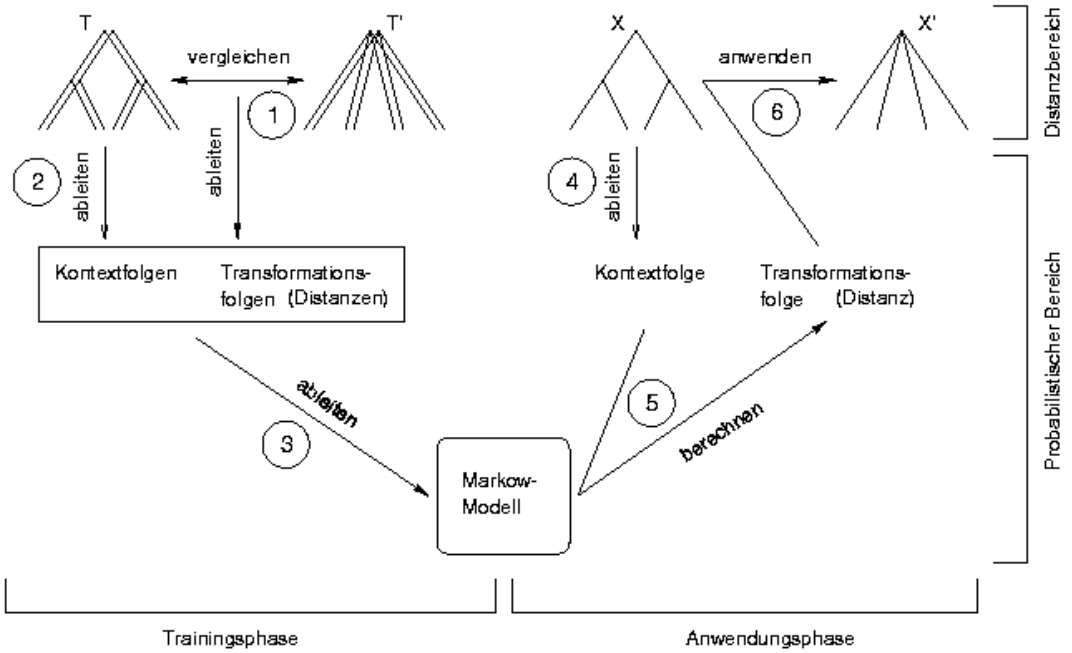


Abbildung 10: Die System-Architektur

Die Verwendung des Systems kann in zwei Phasen, die durch den probabilistischen Bereich motiviert sind, eingeteilt werden. Die einmalige Trainingsphase erstellt ein probabilistisches Modell (siehe auch Kapitel 8 (S. 53)). Dieses Modell wird dann in der darauffolgenden Anwendungsphase immer wieder verwendet (siehe auch Kapitel 9 (S. 67)).

Die Situierung der beiden Phasen und Bereiche im System wird im Folgenden kurz skizziert. Die Abbildung 10 auf der vorherigen Seite stellt die gesamte Architektur des Systems und die Zusammenhänge bildlich dar. Auf diese Abbildung wird in der Folge des öfteren zurückverwiesen.

7.3.1 Trainingsphase in Kürze

Ein annotiertes Textkorpus enthält eine Reihe von Bäumen. Das Ziel der Trainingsphase ist es, ausgehend von zwei annotierten Textkorpora ein Markow-Modell zu erstellen. Dazu müssen die Modell-Parameter, d.h. die Übergangswahrscheinlichkeiten A und die Ausgabewahrscheinlichkeiten B, berechnet werden. Von den beiden Textkorpora bildet eines das Referenzkorpus und das andere das Trainingskorpus.

Gegeben sind das Trainingskorpus $T = T_1, \ldots, T_n$ mit n Bäumen und das Referenzkorpus $T' = T'_1, \ldots, T'_n$ auch mit n Bäumen.

Gesucht sind die Modellparameter A und B für das Markow-Modell M

Es benötigt drei Schritte, um das Ziel zu erreichen. Zur Erleichterung der Orientierung findet sich die Schrittnumerierung auch in der Abbildung 10 auf der vorherigen Seite wieder.

1. Im Baumvergleich wird jeder Baum $T_i \in T$ mit $T'_i \in T'$ verglichen, um eine Distanz D_i zwischen T_i und T'_i zu finden. Eine Distanz besteht aus Transformationen. Es gibt für jeden Knoten $T_i[k]$ im Baum T_i eine Transformation. Das Ergebnis ist ein Distanzkorpus $D = D_1, \ldots, D_n$ mit n Distanzen. Das Distanzkorpus D wird dem Schritt 3 als Grundlage dienen.

2. Aus jedem Baum $T_i \in T$ wird die Kontextfolge K_i abgelesen. Es gibt für jeden Knoten $T_i[k]$ im Baum T_i einen Kontext. Das Resultat ist das Kontextkorpus $K = K_1, \ldots, K_n$ mit n Kontextfolgen. Das Kontextkorpus K wird nächsten Schritt wieder verwendet.

3. Nun kann das Modell M aus den beiden Korpora K (Kontextfolgen) und D (Distanzfolgen) abgeleitet werden. Dazu berechnet man die relativen Häufigkeiten aller Transformationen, die unmittelbar aufeinander folgen (Zustandsübergänge) A und aller Ausgaben der Kontexte von den Distanzen B.

7.3.2 Anwendungsphase in Kürze

Das Ziel der Anwendungsphase ist es, einen Baum in einen entsprechenden unbekannten Zielbaum zu verwandeln. Dazu verwendet man das in der Trainingsphase erstellte Markow-Modell. Die Knoten-Kontexte des Baumes dienen als beobachtete Ausgabe des Modells, womit man die wahrscheinlichste Transformationsfolge berechnen kann. Diese wird auf den Baum angewendet, sodass der wahrscheinlichste Zielbaum entsteht.

Gegeben sind der Originalbaum X und das Modell M.

Gesucht ist der wahrscheinlichste Zielbaum X'.

Drei Schritte sind zur Lösung der Aufgabe nötig. Die Schritte sind auch in der Abbildung 10 (S. 46) zu finden.

4. Während eines Postorder-Durchlaufs durch den Baum X wird die Kontextfolge K_x abgelesen. K_x dient als beobachtete Ausgabe des Modells M.

5. Aus M und der Kontextfolge K_x wird die wahrscheinlichste Distanz D_x mit dem Viterbi-Algorithmus berechnet. Zur Erinnerung: Die Distanz besteht aus einer Folge von Transformationen.

6. Jede Transformation D_{x_i}, die in D_x enthalten ist, wird in eine Menge von z Editieroperationen O_i aufgespalten. Die Vereiningungsmenge aller Editieroperationen für X heisst $O = \bigcup_i O_i$. Jede Editieroperation $O_x \in O$ wird auf den Baum X angewendet, sodass der wahrscheinlichste Zielbaum X' entsteht.

7.3.3 Distanzbereich

Ein Teil der oben erläuterten Arbeitsschritte steht in einer Beziehung zur Ermittlung der Editierdistanz von Baumstrukturen. Es sind die Schritte 1 und 6. Im Schritt 1 ist die klassische Konstellation zur Ermittlung der Editierdistanz gegeben (siehe Kapitel 5 (S. 30)). Ein Baumvergleich ergibt eine Distanz. Die Anwendung der Editieroperationen findet im Schritt 6 statt.

7.3.4 Probabilistischer Bereich

Ein anderer Teil der Arbeitsschritte ist analog zum Tagging mit Hidden-Markow-Modellen. Es sind die Schritte mit den Nummern 2, 3, 4 und 5. In den Schritten 2 und 4 wird von jedem Baum eine Kontextfolge abgelesen. Die Ausgaben des Modells bilden ja die Kontexte ab, deshalb braucht man für die Erstellung des Modells die Kontextfolgen. Im Schritt 3 werden die Modellparameter berechnet. Im Schritt 5 wird die wahrscheinlichste Zustandsfolge mit dem Viterbi-Algorithmus aus dem Modell berechnet. Alle diese Arbeitsschritte sind schon aus dem Kapitel 6 (S. 35) bekannt.

7.4 Auswirkungen der Umwandlung

Es gibt vier Aspekte, unter denen man die Distanz zweier Syntaxbäume, denen der gleiche Satz zugrundeliegt, betrachten kann:

Ähnlichkeit bestimmen: Die Art und Anzahl der Editieroperationen bestimmt die Ähnlichkeit zwischen den beiden Strukturen T und T'.

Fehlerkorrektur: Unter der Annahme, dass T potentiell falsche Strukturen enthält, und hingegen T' grundsätzlich die richtige Struktur erkannt hat, ergibt sich mit der Transformation von T nach T' eine Fehlerkorrektur.

Grammatikwechsel: Ein Syntaxbaum ist die Darstellung der grammatischen Struktur eines Satzes. Unterscheiden sich die Bäume T und T', welche sich auf den gleichen Satz stützen, so unterscheiden sich sehr wahrscheinlich (abgesehen von Fehlern) auch die inhärenten Phrasenstrukturgrammatiken, welche bei der Konstruktion der Bäume verwendet wurde.

Parsen: Wie Brill (1993) demonstriert hat, ist es auch möglich, das Parsingproblem auf eine Strukturannäherung zu reduzieren. Angenommen, der Originalbaum sei ganz primitiv und der Zielbaum ein voll ausgebildeter Syntaxbaum, kann mit dem hier beschriebenen System auch geparst werden.

7.5 Abgrenzung gegenüber den vorgestellten Ansätzen

7.5.1 Kein transformationsbasiertes Lernen

Die im Kapitel 4 (S. 26) beschriebene Lernmethode (= transformationsbasiertes Lernen) unterscheidet sich von der hier verwendeten Lernmethode mit einem Markow-Modell.

Die Transformationsselektion läuft in den beiden Systemen unterschiedlich ab. Brill (1993) bewertet die Transformationen in der Trainingsphase. Die Bewertung wird von negativer Evidenz gesteuert, d.h. diejenigen Transformationen werden gewählt, welche viele Fehler korrigieren und gleichzeitig möglichst wenige neue Fehler verursachen. Die Auswirkungen der Transformationen werden global, d.h. auf den ganzen Korpus bezogen, bewertet. Nach der Trainingsphase sind die Transformationen in einer bestimmten Reihenfolge für das gesamte Korpus gegeben. Die Anwendungsphase muss nur noch die Transformationen der Reihe nach überall, wo der entsprechende lokale Kontext (= *triggering environment*) gegeben ist, anwenden.

Im hier entwickelten System wird die Bewertung der Transformationen erst in der Anwendungsphase vorgenommen, in der Trainingsphase werden die Häufigkeiten der vorkommenden Konstellationen von Kontexten und Transformationen gezählt. Die Bewertung wird von positiver Evidenz gesteuert, indem häufige Transformationen des Modells auch häufig verwendet werden, ohne Berücksichtigung von neuen Fehlern, die so eventuell entstehen können. Wahrscheinlichkeiten von lokalen Knotenkontexten und Vorgänger-Transformationen bilden die Bewertung.

7.5.2 Editierdistanz zwischen zwei Syntaxbäumen

Im Kapitel 5 (S. 30) finden sich diverse Denkanstösse zum Editierdistanzproblem. Unser Problem unterscheidet sich in einigen Punkten vom Editierdistanzproblem.

Weil das Transformationsproblem zusammen mit einem probabilistischen Modell zur Anwendung kommt und deshalb auch den Fall auftritt, dass der Zielbaum T' unbekannt ist, müssen zwingend alle Transformationen am Ausgangsbaum T festgemacht werden. Ausserdem wird die Anwendung der Editieroperationen durch fehlerhaft vorgeschlagene Operationen erschwert.

Zwei verschiedene Syntaxbäume eines Satzes sind geordnet und dürfen in ihrer terminalen Ordnung nicht verändert werden. Terminale dürfen weder gelöscht noch eingefügt werden, weil sonst Wörter aus einem Satz gelöscht würden. Dies motiviert die Einführung der MOVE-Operation, die Teilbäume bewegt. Das Bewegen von Teilbäumen garantiert, dass keine Satzteile gelöscht werden. Im Zusammenhang mit Syntaxbäumen scheinen deshalb die Editieroperationen

- Nichtterminal löschen
- Nichtterminal einfügen

- Knotenbeschriftung ersetzen
- eingeschränktes Teilbaum bewegen

sinnvoll. Weil so weder die Reihenfolge noch die Anzahl der Terminale, die ja Wörter und Tags darstellen, verändert wird.

Das Kostenmodell ist nicht formal spezifiziert, sondern nur implizit im Baumvergleichsalgorithmus (siehe Abschnitt 8.3 (S. 57)) enthalten. Das ist aber nicht weiter nachteilig, weil der Baumvergleich deterministisch konzipiert ist, und so genau eine Lösung berechnet. Diese Lösung entspricht somit der Editierdistanz gemäss einem nicht spezifizierten Kostenmodell.

7.5.3 Komplexität von Markow-Modellen

Das Kapitel 6 (S. 35) behandelt die Verwendung von Markow-Modellen als Tagger. Die dort beschriebene grundsätzliche Vorgehensweise wird in unserem System wiederverwendet. Jedoch sind Unterschiede in der Komplexität des Modells feststellbar.

Ein üblicher Wortarten-Tagger (siehe Abschnitt 6.1 (S. 35)), der Markow-Modelle verwendet, repräsentiert im Modell die Tags mit Zuständen und die Wörter mit Ausgaben. Erstens sind solche Zustände und Ausgaben einfach aufgebaut und zweitens gibt es im Schnitt ca. 70 Zustände. Dafür ist die Anzahl verschiedener Ausgaben sehr hoch, da es viele verschiedene Wortformen gibt.

Der erweiterte Chunk-Tagger, wie er im Abschnitt 6.2.1 (S. 39) beschrieben ist, enthält zusammengesetzte Informationen in den Zuständen und als Ausgabe. Die Ausgaben sind Wörter zusammen mit ihren Tags. Dies ergibt eine etwas höhere Anzahl verschiedener Ausgaben. Die Zustände repräsentieren die Chunk-Tags, welche aus den folgenden Teilen zusammengesetzt ist: Das Tag, die syntaktische Kategorie des Mutterknotens und das strukturelle Verhältnis des Wortes zum unmittelbar vorhergehenden Wort. Die kaskadierten Markow-Modelle repräsentieren als Ausgaben Phrasenstrukturen, also auch komplexe Informationen. Solche Zusammensetzungen bewirken eine Erhöhung der Anzahl der möglichen Zustände gegenüber dem Wortarten-Tagger.

Unser System verwendet ähnlich dem erweiterten Chunk-Tagger von Skut (1999) zusammengesetzte Information als Zustände und Ausgaben des Markow-Modells. Die Komplexität der kodierten Information ist also einiges höher als im Wortarten-Tagger. Die Chunk-Tags gleichen unseren Knoten-Kontexten (siehe 8.4 (S. 63)), doch diese werden von den Ausgaben des Modells repräsentiert. Die Anzahl der Zustände ist sehr hoch im

Vergleich zum Wortarten-Tagger: Ein Modell für 1800 Sätze enthält ca. 3000 verschiedene Zustände. Entsprechend erhöht sich die Anzahl der möglichen Übergänge und die Anzahl möglicher Pfade beim Berechnen der wahrscheinlichsten Zustandsfolge.

8 Trainingsphase

Der Gegenstand dieses Kapitels ist die Trainingsphase des Systems. Es soll ein tieferer Einblick in die Vorbereitung und das supervisierte Lernverfahren selbst gegeben werden. Es werden Definitionen und die Lösungen aller grösseren Teilprobleme des Trainings beschrieben.

Die konkrete Aufgabe der Trainingsphase liegt darin, aufgrund zweier annotierter Textkorpora ein Markow-Modell zu erstellen. Die Trainingsphase setzt sich aus den Schritten 1 bis 3 aus der Abbildung 10 (S. 46) zusammen. Der erste Schritt betrifft den Baumvergleich, anhand dessen das Distanzkorpus erstellt wird. Im zweiten Schritt wird der Kontext jedes Knotens im Baum abgelesen. Schliesslich werden im dritten Schritt die Modellparameter berechnet.

Als erstes stehen Erläuterungen zu den Editieroperationen und Transformationen, dann folgt der detaillierte Baumvergleichsalgorithmus. Danach werden die Möglichkeiten der Knotenkontexte dargelegt, um schliesslich die Kodierung des Modells zu diskutieren.

8.1 Editieroperationen

Die Editieroperationen bilden die Grundlage der Transformationen. Alle Arten von Editieroperationen für Baumstrukturen beziehen sich auf genau einen Knoten, den sie bearbeiten.

Editieroperationen können Operationstypen oder -instanzen sein. Die allgemeine Beschreibung bezieht sich immer auf Typen. Im Anwendungsfall werden in der Regel Operationsinstanzen behandelt.

Einige Bemerkungen zur Notation und Terminologie: Die Menge von Knotennummerpaaren $\langle i, j \rangle$ bildet die Knotenzuordnung Z. Jeder Knoten kann höchstens einem Knoten zugeordnet werden, die beiden Knoten heissen auch Partner. Ein Knoten des Baumes T mit der Knotennummer i heisst $T[i]$. Die Beschriftung eines Knotens $T[i]$ heisst $l(i)$.

8.1.1 OK

Die OK-Operation $\mathsf{ok}(i)$ beschreibt die Gleichheit der Beschriftung von Partnerknoten. Der Knoten $T[i]$ ist mit der gleichen Kategorie $l(i)$ wie der entsprechende Referenzknoten ausgezeichnet. OK besagt, dass die Beschriftung des Knotens unverändert bleiben soll. Diese Operation erscheint vielleicht etwas redundant, doch die explizite Feststellung, dass die Beschriftung übereinstimmt, hilft beim Erstellen eines aussagekräftigen Modells.

8.1.2 DELETE

Die DELETE-Operation $\mathsf{del}(i)$ zeigt an, dass der Knoten $T[i]$ im Referenzbaum keinen Partnerknoten hat. $T[i]$ muss gelöscht werden. Allfällige Kinder von $T[i]$ werden neu zu Kindern der Mutter $T[j]$ von $T[i]$. Terminale haben immer einen Partnerknoten und dürfen nie gelöscht werden.

8.1.3 INSERT

Die INSERT-Operation $\mathsf{ins}(i, j, l(j))$ konstatiert, dass in T ein Knoten fehlt, weil der Knoten $T'[j]$ keinen Partnerknoten hat. Ein neuer Partnerknoten zu $T'[j]$ mit der Beschriftung $l(j)$ soll als Mutter von $T[i]$ eingefügt werden.[10] Terminale dürfen nie eingefügt werden.

8.1.4 REPLACE

Die REPLACE-Operation $\mathsf{rpl}(i, l(j))$ hält fest, dass der Knoten $T[i]$ die Kategorie $l(i)$ hat, welche sich von der Kategorie $l(j)$ des Partnerknotens $T'[j]$ unterscheidet. $l(i)$ muss durch $l(j)$ ersetzt werden.

8.1.5 MOVE

Die MOVE-Operation $\mathsf{mov}(i, j)$ besagt, dass der Knoten $T[i]$ unter der falschen Mutter hängt. $T'[j]$ ist der Partner von $T[m]$. Der gesamte Teilbaum, der bei $T[i]$ startet, muss unter die neue Mutter $T[m]$ gehängt werden. Die MOVE-Operation enthält eine Knotennummer des Zielbaumes. Es ist aber möglich, von j über die Zuordnung auf den Zielknoten m im Originalbaum zu gelangen.

MOVE unterscheidet sich in einigen Punkten von den andern Operationen. Es ist die komplexeste Operation: Sie simuliert das gleichzeitige Löschen und Einfügen eines ganzen Teilbaums.

Diese Operation drängt sich speziell bei Syntaxbäumen auf, da Syntaxbäume ja geordnete Bäume sind, welche in ihrer Ordnung nicht verändert werden dürfen. Das Löschen eines Teilbaums darf nicht ohne dessen Einfügen (und umgekehrt) angewandt werden, deshalb ist für Terminale nur die Kombination von Löschen und Einfügen von Teilbäumen

[10] Es wären auch andere Strategien vorstellbar, wann und wo man einen einzufügenden Knoten am Originalbaum festmacht. Z.B. könnte man definieren, dass $T[i]$ der Mutterknoten des neuen Knotens sei. Dann müsste man allerdings noch bestimmen, wo der neue Knoten in den Kindern des Mutterknotens eingereiht werden sollte. Indem wir die Tochter als Anknüpfungspunkt wählen, gibt es nur eine Möglichkeit, wo der neue Knoten im Baum positioniert werden kann.

erlaubt. MOVE ist zusätzlich aufgrund der unveränderbaren Reihenfolge der Terminale nur eingeschränkt anwendbar.

Es dürfen keine Operationen durchgeführt werden, bei denen überkreuzende Kanten entstehen können. Zu diesem Zweck wird eine Wartebank benutzt, die sicherstellt, dass Operationen in einer idealen Reihenfolge durchgeführt werden, sodass keine defizitären Bäume (mit überkreuzenden Kanten oder abgesägten Teilbäumen) resultieren.

8.2 Transformation: Menge der Operationen pro Knoten

Die beschriebenen Editieroperationen sind in einem probabilistischen Modell nicht direkt verwendbar, da sie erstens zu spezifisch und zweitens nicht synchron mit den Knotenkontexten sind. Deshalb müssen die Editieroperationen zugleich abstrahiert und in Transformationen verpackt werden.

8.2.1 Verpackung der Editieroperationen in Transformationen

Es gibt pro Knoten einen Kontext. An jedem Knoten können aber 1 bis 3 Operationen notwendig sein. Diese Operationen zusammen ergeben dann eine Transformation des Knotens. Eine Transformation zerfällt in zwei Teile:

1. Im ersten Teil der Transformation steht eine Operation, die den Knoten an sich betrifft. Die möglichen Operationen für den ersten Teil sind: $\mathsf{ok}(i)$, $\mathsf{rpl}(i, l(j))$ und $\mathsf{del}(i)$.

2. Der zweite Teil ist optional und darf *nicht* nach einer DELETE-Operation realisiert werden. Die Operationen des zweiten Teils betreffen die Verbindung zur Mutter. Die möglichen Operationen für den zweiten Teil sind: $\mathsf{ins}(i, j, l(j))$ und $\mathsf{mov}(i, j)$.

Der reguläre Ausdruck $(\mathsf{del}|((\mathsf{ok}|\mathsf{rpl})[[\mathsf{ins}]\mathsf{mov}]))$ beschreibt die Form der möglichen Transformationen.

8.2.2 Abstraktion

Die Gemeinsamkeit aller Editieroperationen besteht darin, dass sie immer genau einen Knoten mit der Nummer i bearbeiten. Da das probabilistische Modell eine Abstraktion der Wirklichkeit dargestellt, müssen die Editieroperationen innerhalb der Transformationen abstrahiert werden.

Operation	**abstrahierte Operation**	**Darstellung in PROLOG**
$\mathsf{ok}(i)$	$\mathsf{ok}(l(i))$	`ok(Tag)`
$\mathsf{del}(i)$	$\mathsf{del}(l(i))$	`d(Tag)`
$\mathsf{rpl}(i, l(j))$	$\mathsf{rpl}(l(i), l(j))$	`r(AltesTag, NeuesTag)`
$\mathsf{ins}(i, j, l(j))$	$\mathsf{ins}(l(j))$	`i(Tag)`
$\mathsf{mov}(i, j)$	$\mathsf{mov}(l(i), p(i, m))$	`m(Tag, path(Hinauf,Hinab,ZielTag))`

Tabelle 5: Die Abstraktionen der Editieroperationen

Die Tabelle 5 zeigt die abstrahierten Operationen, wobei $i \in T, j \in T'$ und $m \in T$. Zum Zeitpunkt der Verpackung der Operationen in Transformationen wird grundsätzlich jede Knotennummer i durch die Knotenbeschriftung ersetzt.

Die INSERT-Operation wird auf die Beschriftung des neuen Knotens reduziert, da i bzw. $l(i)$ schon im ersten Teil der Transformation vorkommt.

Die Abstraktion der MOVE-Operation verwendet den Pfad $p(i, m)$ vom Knoten $T[i]$ zum Knoten $T[m]$. $T[m]$ ist der Partner von $T'[j]$. Der Pfad beschreibt den Weg im Baum, an welchem entlang der MOVE-Knoten bewegt werden soll. Der Pfad bezieht sich immer auf den Originalbaum T, an dem noch gar keine Operationen durchgeführt worden sind. Der Bezug zum Originalbaum hat den Vorteil, dass sich der Pfad nicht laufend mit der Durchführung der Operationen ändert. Problematisch sind MOVE-Operationen auf Knoten, die erst noch eingefügt werden müssen, weil es den Pfad zu diesem Knoten im Originalbaum nicht gibt. Es wird deshalb die Hilfskonstruktion des speziellen Pfads $pp(i, m)$ eingeführt. Der spezielle Pfad macht sich die Tatsache zunutze, dass ein einzufügender Knoten von einem existierenden Kinderknoten des Baum eingefügt wird. Das erste Wegstück des Pfades zu einem einzufügenden Knoten besteht aus dem Pfad zu seinem Kinderknoten im Originalbaum. Das zweite Wegstück führt dann vom Kinderknoten im aktuellen Baum aus ein Schritt hinauf zum eingefügten Knoten. Sollten an einem Knoten mehrere Knoten eingefügt werden, erhöht sich eventuell, je nachdem, welcher Knoten als Ziel betroffen ist, die Anzahl der letzten Aufwärtsschritte.

Der normale Pfad wird in Prolog als `path(Hinauf, Hinab, ZielTag)` kodiert. Hierbei bedeutet `Hinauf` die Anzahl der Schritte, die im Baum Richtung Wurzel gegangen werden müssen, `Hinab` kodiert die Abwärtsschritte und `ZielTag` bezeichnet die Beschriftung des Partners des Zielknotens. Der spezielle Pfad $pp(i, m)$ erhält seine Entsprechung im Term `ipath(Hinauf, Hinab, ZielTag, ZusatzSchritte)`. Die Variable `ZusatzSchritte` enthält die Anzahl der zusätzlichen Aufwärtsschritte im aktuellen

Baum.

Ein zweiter Spezialfall bildet der Pfad zur Wurzel. Da die Wurzel in jedem Baum ein Fixpunkt ist, bedeutet das Atom `toRoot` die Kurzvariante vom Pfad zur Wurzel. Diese Kurzvariante bildet eine weitere Abstraktion über der MOVE-Operation, da dieser Spezialpfad in jeder Baumstruktur seine spezifische Entsprechung hat.

8.3 Schritt 1: Baumvergleich

In der Überblicksabbildung 10 (S. 46) ist der Baumvergleich mit der Nummer 1 bezeichnet. Das Ziel des Baumvergleichs ist es, die Distanz zu finden, welche T in T' verwandelt. Die Distanz ist eine Folge von Transformationen, welche wiederum aus einer Menge von Editieroperationen zusammengesetzt sind.

Es wurde zwar die Idee der Zuordnung von Tai (1979) übernommen, doch die Bedingungen, unter denen ein Knoten zugeordnet werden kann, sind anders bestimmt:

1. Jeder Knoten kann höchstens einem anderen Knoten zugeordnet sein.

2. Die Wurzel und alle Terminale der beiden Bäume sind immer zugeordnet.

Die Zuordnung hat unter diesen Bedingungen nicht dieselben Eigenschaften, wie sie im Abschnitt 5.2 beschrieben sind. Sie stellt in diesem Fall nur die Editieroperationen OK, REPLACE, DELETE und INSERT komprimiert dar. Die MOVE-Operation ist nicht direkt daraus ersichtlich. Ausserdem sind gemäss diesen Bedingungen auch überkreuzende Zuordnungslinien möglich.

Im Baumvergleich wird zunächst eine Menge von Editieroperationen gesucht. Dazu prüft man in einem Baumdurchlauf jeden Knoten $T[i]$ auf Beschriftung, Mutterbeziehung und seinen allfälligen Partner $T'[j]$. Dieser erste Durchgang ergibt alle OK-, REPLACE-, DELETE-Operationen, vorläufige MOVE-Operationen und eine vorläufige Knotenzuordnung, welches als Hilfsstruktur dient. Dann müssen die vorläufigen MOVE-Operationen definitiv angepasst und die nötigen INSERT-Operationen abgeleitet werden.

Schliesslich stellt man aus der Menge der Operationen und der Postorder-Reihenfolge die gesuchte Folge von Transformationen zusammen.

8.3.1 Ablauf des Baumvergleichs

Initialisierung Von Anfang an sind die folgenden Paare in der Knotenzuordnung Z:

- Die Wurzel von T mit der von T'

- Jedes Terminal von T mit dem entsprechenden Terminal von T'.

Die Menge D aller Editieroperationen von T und T' enthält die passenden Instanzen der Editieroperationen. Am Anfang ist D immer leer: $D = \emptyset$.

Postorder-Durchlauf Es werden alle Knoten von T in Postorder-Reihenfolge nacheinander untersucht. Die Untersuchung jedes Knotens $T[i]$ läuft nach dem folgenden Schema ab:

1. Ist das Paar $\langle i, j \rangle \in Z$?

   Ja: Ist $l(i)$ gleich $l(j)$?

   Ja: Füge die Operation $\mathsf{ok}(i)$ in D hinzu.

   Nein: Füge die Operation $\mathsf{rpl}(i, l(j))$ in D hinzu.

   Nein: Füge die Operation $\mathsf{del}(i)$ in D hinzu und beende die Untersuchung von $T[i]$.

2. Falls $T[i]$ und sein Partner $T'[j]$ je eine Mutter haben: Ist es möglich, die Mutter $T[m]$ von $T[i]$ der Mutter $T'[n]$ von $T'[j]$ zuzuordnen? (Zur Erinnerung: Jeder Knoten kann höchstens einmal zugeordnet werden.)

   Ja: Füge das Paar $\langle m, n \rangle$ in Z hinzu.

   Nein: Füge die Operation $\mathsf{mov}(i, n)$ in D hinzu und beende die Untersuchung von $T[i]$.

Dieser einmalige Baumdurchlauf liefert bereits alle OK-, REPLACE- und DELETE-Operationen. Die MOVE-Operationen werden danach nochmals separat bearbeitet, um die nötigen INSERT-Operationen zu finden.

Finden der notwendigen INSERT-Operationen Nachdem alle Knoten von T einem Vergleich unterzogen wurden, fehlen nun noch die INSERT-Operationen. Ein sicherer Indikator für eine INSERT-Operation ist ein MOVE auf einen Knoten, der nicht in T enthalten ist. Die MOVE-Operation $\mathsf{mov}(i, n)$ bedeutet ja, dass $T[i]$ unter den Knoten $T[m]$ bewegt werden muss, wobei $T[m]$ der Partner von $T'[n]$ ist. Die Partnerbeziehung ist in der Zuordnung Z gespeichert. Hat nun ein Knoten $T'[n]$ keinen Partner in T, muss ein neuer Knoten $T[m]$ eingefügt werden. Das Einfügen geschieht immer beim ersten Knoten, der auf $T[m]$ bewegt wird.

Als Hilfsgrösse wird die Menge MOV eingeführt, welche alle MOVE-Operationen enthält: $MOV = \{\mathsf{mov}(i,n) | \mathsf{mov}(i,n) \in D\}$.

Jedes Element von MOV muss nach dem folgenden Schema untersucht werden:

- Ist $\langle m, n\rangle \in Z$?

  Ja: Die Operation $\mathsf{mov}(i,n)$ ist in Ordnung und bleibt definitiv so stehen, die Untersuchung dieser Operation kann beendet werden.

  Nein: Es muss also ein Knoten in T eingefügt werden.

  1. Bestimme eine neue Knotennummer m für den in T einzufügenden Knoten $T[m]$. Ersetze die Operation $\mathsf{mov}(i,n)$ durch $\mathsf{ins}(i,m,l(n))$.
  2. Die Mutter von $T'[n]$ ist $T'[x]$. Füge die neue Operation $\mathsf{mov}(m,x)$ in MOV ein, denn der neu eingefügte Knoten muss nach dem Einfügen unter seine richtige Mutter, das ist der Partner von $T'[x]$, bewegt werden.
  3. Füge das Paar $\langle m, n\rangle$ in Z ein. Dies verhindert ein abermaliges Einfügen von $T[m]$.
  4. Beende die Untersuchung von $\mathsf{mov}(i,n)$ und untersuche nun nach demselben Schema $\mathsf{mov}(m,x)$.

Ein kleines Beispiel veranschaulicht diese schematische Untersuchung. Die Abbildung 11 zeigt die beiden Bäume T und T'.

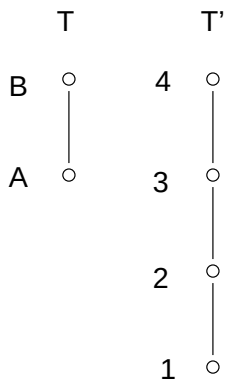


Abbildung 11: Die Bäume T und T'

1. Die Initialisierung und der Baumdurchlauf des Beispiels haben folgende Mengen ergeben:

$$Z_1 = \{\langle A, 1\rangle, \langle B, 4\rangle\} \tag{8.1}$$

$$D_1 = \{\mathsf{ok}(A), \mathsf{mov}(A, 2), \mathsf{ok}(B)\} \tag{8.2}$$

$$MOV_1 = \{\mathsf{mov}(A, 2)\} \tag{8.3}$$

   Untersuche $\mathsf{mov}(A, 2)$: $\langle x, 2\rangle \notin Z$, also ersetze $\mathsf{mov}(A, 2)$ durch $\mathsf{ins}(A, C, l(2))$, $\mathsf{mov}(C, 3)$ und füge $\langle C, 2\rangle$ in Z ein.

2. Die Grundlagen haben sich verändert:

$$Z_2 = \{\langle A, 1\rangle, \langle B, 4\rangle, \langle C, 2\rangle\} \tag{8.4}$$

$$D_2 = \{\mathsf{ok}(A), \mathsf{ins}(A, C, l(2)), \mathsf{mov}(C, 3), \mathsf{ok}(B)\} \tag{8.5}$$

$$MOV_2 = \{\mathsf{mov}(C, 3)\} \tag{8.6}$$

   Untersuche $\mathsf{mov}(C, 3)$: $\langle y, 3\rangle \notin Z$, also ersetze $\mathsf{mov}(C, 3)$ durch $\mathsf{ins}(C, D, l(3))$, $\mathsf{mov}(D, 4)$ und füge $\langle D, 3\rangle$ in Z ein.

3. Die Grundlagen haben sich erneut verändert:

$$Z_3 = \{\langle A, 1\rangle, \langle B, 4\rangle, \langle C, 2\rangle\, \langle D, 3\rangle\} \tag{8.7}$$

$$D_3 = \{\mathsf{ok}(A), \mathsf{ins}(A, C, l(2)), \mathsf{ins}(C, D, l(3)), \mathsf{mov}(D, 4), \mathsf{ok}(B)\} \tag{8.8}$$

$$MOV_3 = \{\mathsf{mov}(D, 4)\} \tag{8.9}$$

   Untersuche $\mathsf{mov}(D, 4)$: $\langle B, 4\rangle \in Z$, also ist $\mathsf{mov}(D, 4)$ in Ordnung. Es ist nun kein MOVE mehr in D_3, welches nicht untersucht wurde. Es wurden alle INSERT-Operationen gefunden.

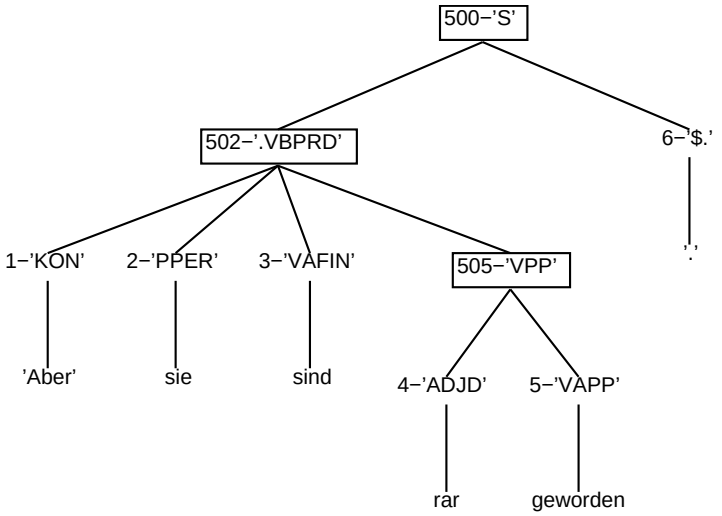
Abstrahieren und Transformationen zusammenstellen Es werden jeweils alle Editieroperationen, die denselben Knoten $T[i]$ bearbeiten, zusammen in eine Transformation gepackt und abstrahiert. Knotennummern lassen sich recht einfach durch Verwenden der Knotenbeschriftung abstrahieren. Die aufwändigere Abstraktion fordert die MOVE-Operation durch die Pfadinformation (siehe Abschnitt 8.2 (S. 55)). Die Postorder-Reihenfolge der Knoten ergibt die Reihenfolge der Transformationen.

8.3.2 Beispiel mit DELETE

Gegeben sind der zu transformierende Baum T und der Zielbaum T' (siehe Abbildung 12).

Gesucht gesucht wird die Distanz D zwischen T und T'.

T



T'

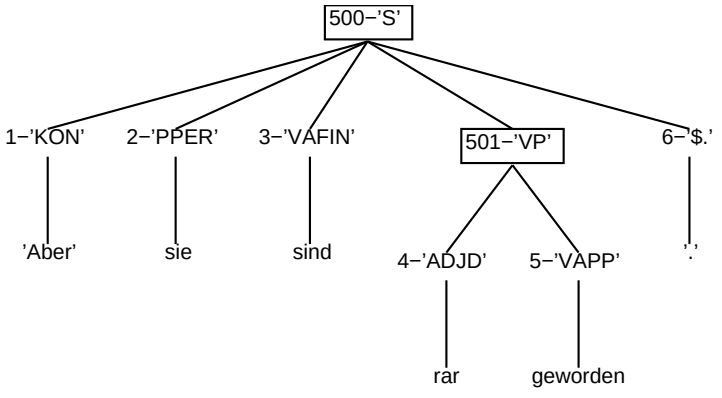


Abbildung 12: Zwei zu vergleichende Bäume T und T'

Initialisierung

$$\begin{aligned} D_{start} &= \emptyset \\ Z_{start} &= \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle, \langle 4,4\rangle, \langle 5,5\rangle, \langle 6,6\rangle, \langle 500,500\rangle\} \end{aligned}$$

Baumdurchlauf Der direkte Baumvergleich ergibt die Tabelle 6 .[11]

KnotenNr von T:	**Partner:**	**Mutter:**	**Z erweitern:**	**D erweitern:**
1	1	502 statt 500′		$\mathsf{ok}(1), \mathsf{mov}(1,500)$
2	2	502 statt 500′		$\mathsf{ok}(2), \mathsf{mov}(2,500)$
3	3	502 statt 500′		$\mathsf{ok}(3), \mathsf{mov}(3,500)$
4	4	505	$\langle 505,501\rangle$	$\mathsf{ok}(4)$
5	5	505		$\mathsf{ok}(5)$
505	501	502 statt 500′		$\mathsf{rpl}(505, \mathrm{VP})$
502		500		$\mathsf{del}(502)$
6	6	500		$\mathsf{ok}(6)$
500	500			$\mathsf{ok}(500)$

Tabelle 6: Vorläufige Operationen

Z wird sich nun nicht mehr verändern und D ist vorläufig aus der Tabelle ablesbar:

$$\begin{aligned} D_9 &= \{\mathsf{ok}(1), \mathsf{mov}(1,500), \mathsf{ok}(2), \mathsf{mov}(2,500), \mathsf{ok}(3), \mathsf{mov}(3,500), \mathsf{ok}(4), \mathsf{ok}(5), \\ &\quad \mathsf{ok}(6), \mathsf{rpl}(505, \mathrm{VP}), \mathsf{del}(502), \mathsf{ok}(500)\} \\ Z_{end} &= \{\langle 1,1\rangle, \langle 2,2\rangle, \langle 3,3\rangle, \langle 4,4\rangle, \langle 5,5\rangle, \langle 6,6\rangle, \langle 500,500\rangle, \langle 505,501\rangle\} \end{aligned}$$

MOVE und INSERT bearbeiten Die Menge $MOV = \{\mathsf{mov}(1,500), \mathsf{mov}(2,500), \mathsf{mov}(3,500)\}$ besteht aus allen vorläufigen MOVE-Operationen. Jedes Element dieser Menge wird abstrahiert. Da diese MOVES alle auf die Wurzel des Baumes gehen, muss kein Knoten eingefügt werden.

Abstraktion und Transformationen zusammenstellen Die Abstraktionen der Editieroperationen sind hier trivial: Die Knotennummern werden durch die Knotenbeschriftungen ersetzt und die Pfade in den MOVES gehen alle zur Wurzel. Die Tabelle 7 auf der nächsten Seite listet die Transformationen auf.

[11] X' bedeutet Partner von X

Nr.	Transformation	in PROLOG
1	$\mathsf{ok}(\mathrm{KON}), \mathsf{mov}(\mathrm{KON}, p(1,500))$	`transformation(S,1,[ok('KON'), m('KON',toRoot])).`
2	$\mathsf{ok}(\mathrm{PPER}), \mathsf{mov}(\mathrm{PPER}, p(2,500))$	`transformation(S,2,[ok('PPER'), m('PPER',toRoot])).`
3	$\mathsf{ok}(\mathrm{VAFIN}), \mathsf{mov}(\mathrm{VAFIN}, p(3,500))$	`transformation(S,3,[ok('VAFIN'), m('VAFIN', toRoot]).`
4	$\mathsf{ok}(\mathrm{ADJD})$	`transformation(S,4,[ok('ADJD')]).`
5	$\mathsf{ok}(\mathrm{VAPP})$	`transformation(S,5,[ok('VAPP')]).`
505	$\mathsf{rpl}(\mathrm{VPP}, \mathrm{VP})$	`transformation(S,505,[r('VPP', 'VP')]).`
502	$\mathsf{del}(.\mathrm{VBPRD})$	`transformation(S,502,[d('.VBPRD')]).`
6	$\mathsf{ok}(\$.)$	`transformation(S,6,[ok('$.')]).`
500	$\mathsf{ok}(\mathrm{S})$	`transformation(S,500,[ok('S')]).`

Tabelle 7: Transformationen

8.4 Schritt 2 und 4: Knoten-Kontexte

Die Schritte Nummer 2 und 4 aus der Abbildung 10 (S. 46) widmen sich dem Zusammenstellen der Knoten-Kontexte. Die Knoten-Kontexte enthalten Informationen über die lokale Baumstruktur um einen Knoten. Es sind zahlreiche Teilinformationen vorstellbar, die ein Kontext enthalten könnte:

Eigenschaften des aktuellen Knotens: Darunter fallen die Beschriftung, das Niveau, die Art des Knotens, der Pfad zur Wurzel.

Beschriftungen von engen Verwandten: Interessant sind Knoten, die in einer engen Beziehung zum aktuellen Knoten stehen. Dies sind beispielsweise der Mutter- und der Grossmutterknoten, die Geschwister- oder Kinderknoten etc.

Eigenschaften des Baumes: Z.B. die Länge des Satzes, die Höhe des Baums und die Anzahl der Knoten könnten interessant sein.

Linguistische Informationen: Eine wichtige linguistische Information stellt die Beschriftung dar, sie bezeichnet ja die lexikalische bzw. syntaktische Kategorie des Knotens. Eine zusätzliche Unterscheidung von Phrasentypen, morphologischen Informationen oder Kopf (engl. *head*) einer Phrase ist auch vorstellbar.

Die aufgezeigten Eigenschaften dürfen nicht alle gemeinsam im Kontext enthalten sein. Denn dies würde zwangsläufig zu einer massiven Überspezifizierung des Modells und zu

allzu spärlicher Datengrundlage führen. Es muss ein optimaler Grad an Spezifität ermittelt werden.

Eine Evaluierung hat ergeben, dass die Kombination der Beschriftungen des aktuellen Knotens, neben derjenigen der Mutter, der Grossmutter und der linken Schwester vernünftige Resultate liefert. Diese Beschriftungsinformationen sind die wichtigsten Indikatoren für eine Veränderung am Knoten. Alle anderen Informationen sind zwar an sich interessant und würden wohl mehr Klarheit in das Modell bringen, haben aber den Nachteil, dass sie das Modell zu sehr spezifizieren und keine gute Grundlage bieten, um genügend Abstraktion im Modell zu erreichen.

Der Kompromiss zwischen Spezifität und Allgemeinheit im Modell führt insbesondere bei der MOVE-Operation zu einer Schwierigkeit, die nicht gelöst werden konnte: Die Spannweite der Pfadinformation geht oft weit über den definierten lokalen Knotenkontext hinaus. Eine erste Idee zur Abhilfe wäre, die Kontextinformation zu erweitern, um die Pfade zu erfassen. Dagegen spricht, dass die Kontextinformation nicht zu spezifisch sein darf (siehe Abschnitt 8.4 (S. 63)). Eine zweiter Vorschlag erlaubt nur Pfade von einer bestimmten Länge. Dies ergibt das Problem, dass die Transformationen dann entweder nicht mehr synchron zu den Kontexten sind oder der ganze Baumvergleich neu gestaltet werden müsste. Eine dritte Variante vergrössert die Kontextinformation nur im gegebenen Fall, dass ein MOVE auftritt. Dies hat aber zur Folge, dass im Anwendungsfall die Kontexte nicht von Anfang an alle bekannt sind. Das Modell kann also nicht alle notwendigen Informationen speichern. Es wird deshalb erwartet, dass die MOVE-Operation häufiger Fehler enthalten wird als die übrigen Operationen (siehe Abbildung 24 (S. 88)).

In der Implementation werden Listen für die Zusammenstellung der Kontext-Information verwendet. Das Atom `null` findet immer dann Anwendung, wenn Information nicht verfügbar ist, um die Listenlänge konstant zu halten.

8.5 Schritt 3: Modell kodieren

Das Modellkonzept ist im Kapitel 7.2 (S. 45) ausführlich beschrieben. Die Zustandsübergänge repräsentieren Bigramme von Transformationen. Das Aussenden von Ausgaben bedeutet, dass bei einem Baumknoten ein Knoten-Kontext zusammen mit einer Transformation auftritt. Wie sich eine Transformation aus Editieroperationen zusammensetzt, ist im Kapitel 8.2 (S. 55) aufgeführt. Ein Modell ist durch seine Modellparameter zu kodieren. In einem Markow-Modell gibt es einen Parameter A für die Zustandsübergänge und B für das

Aussenden von Ausgaben (siehe 3.3.2 (S. 22) und 6.1 (S. 35)). In der Abbildung 10 (S. 46) des System-Ablaufs ist das Kodieren des Modells mit der Nummer 3 gekennzeichnet.

In der Implementierung der Trainingsphase werden die Transformationen als PROLOG-Fakten dargestellt: `transformation(SatzNummer, KnotenNummer, EditOpsListe)`. Die `EditOpsListe` ist eine Liste, die alle ermittelten Editieroperationen für den Knoten mit der `KnotenNummer` enthält. Die Information der `SatzNummer` wird benutzt, um festzustellen, wann ein neuer Satz beginnt.

Die Kontextinformation ist analog zur Transformation aufgebaut: `context(SatzNummer, KnotenNummer, InfoListe)`. Die `InfoListe` enthält alle Informationen zum Knoten-Kontext des Knotens mit dieser `KnotenNummer`.

Aus diesen Daten können die Modellparameter berechnet werden.

8.5.1 Parameter bestimmen

Nachdem alle Bäume des Trainingskorpus mit dem Referenzkorpus verglichen wurden, stehen alle `transformation/3`- und `context/3`-Klauseln zum Auszählen zur Verfügung. Aus diesen beiden Prädikaten werden alle Zustandsübergänge in Form von `transition/2`-Klauseln und Ausgaben als `output/2`-Klauseln zusammengestellt. Ein Zustandsübergang repräsentiert ja zwei Transformationen, die von Baumknoten stammen, welche im Postorder-Durchlauf unmittelbar aufeinanderfolgen. Jedes Transformationsbigramm wird in einer Klausel `transition(EditOpsListe1, EditOpsListe2)` verpackt. Das Spezialatom `start` steht stellvertretend für einen Startzustand, der keinen Kontext aussendet und nur Verbindungen zur ersten Transformation eines Baumes besitzt. Das Aussenden einer Ausgabe in einem Zustand repräsentiert das gemeinsame Auftreten von einem Kontext und einer Transformation in einem Knoten, d.h. die Variable `Knotennummer` der `transformation/3`-Klausel ist mit jener der `context/3`-Klausel unifizierbar. Jedes vorkommende Kontext-Transformationspaar gelangt in eine Klausel `output(InfoListe, EditOpsListe)`.

Die Wahrscheinlichkeit für einen Übergang wird aufgrund der Anzahl aller Klauseln `transition(T1, T2)` und `transition(T1, _)` berechnet. Analog dazu ergeben die Klauseln `output(K, T)` und `output(_, T)` die Ausgabewahrscheinlichkeit. Die Übergangswahrscheinlichkeiten werden als Fakten der Form `transprob(T1, T2, UebergangsWKeit)` und die Ausgabewahrscheinlichkeit mit `outprob(K, T, AusgabeWKeit)` gespeichert.

Die Modellparameter werden analog zum Tagging mit Markow-Modellen mit der Maximum-Likelihood-Schätzung berechnet (siehe 6.1.2 (S. 37)).[12]

$$\texttt{UebergangsWKeit} = \frac{\#\,\texttt{transition(T1, T2)}}{\#\,\texttt{transition(T1, \_)}} \tag{8.10}$$

$$\texttt{AusgabeWKeit} = \frac{\#\,\texttt{output(K, T)}}{\#\,\texttt{output(\_, T)}} \tag{8.11}$$

Das Listing aller `transprob/3` und `outprob/3` bildet das berechnete Modell.

[12] # steht für Anzahl

9 Anwendungsphase

In der Anwendungsphase wird das probabilistische Modell als Hidden-Markow-Modell interpretiert und es werden damit Rückschlüsse von Knoten-Kontexten auf Transformationen gezogen. Auch die Anwendung der Transformationen auf einen Baum gehört in die Anwendungsphase. In der grafischen Darstellung des Ablaufs 10 (S. 46) betrifft die Anwendungsphase die Schritte 4 bis 6. Das Resultat des vierten Schritts sind die Knoten-Kontexte eines Baums. Für eine Beschreibung von Kontexten siehe Abschnitt 8.4 (S. 63)). Die Kontexte bilden im Schritt 5 zusammen mit dem Markow-Modell die Berechnungsgrundlage für die Ableitung der wahrscheinlichsten Transformationsfolge. Der letzte Schritt wandelt den Baum durch Anwenden der Transformationen um.

9.1 Schritt 5: Benutzen des Markow-Modells

Das Markow-Modell ist durch die Prädikate `transprob/3` und `outprob/3` spezifiziert (für eine Beschreibung der beiden Prädikate siehe Abschnitt 8.5.1 (S. 65)). Die Berechnung der wahrscheinlichsten Zustandsfolge wird mit dem Viterbi-Algorithmus (siehe Abschnitt 3.3.1 (S. 20)) durchgeführt. Trotz der Beschränkung der Komplexität durch den Viterbi-Algorithmus stellt die Berechnung der wahrscheinlichsten Distanz eines Baumes einen enormen Rechenaufwand und damit den Engpass des Systems dar.

In einem durchschnittlichen Modell, beispielsweise gebildet aufgrund eines Trainingskorpus von 1800 Sätzen, gibt es durchschnittlich ungefähr 3000 verschiedene Zustände. Ein kurzer Satz von 10 Wörtern hat je nach Grammatik ca. 20 Knoten. Pro Zeiteinheit gibt es im Modell 3000^2 Zustandsübergänge und, wenn ein Zustand ungefähr 10 Ausgaben hat, $3000 \cdot 10$ Ausgaben. Um alle Akkumulatoren $\delta_t(i)$ zum Zeitpunkt t zu berechnen, braucht es also schätzungsweise $30000 \cdot 3000^2 = 270$ Milliarden Rechenoperationen. Für einen kleinen Baum von 20 Knoten sind das stolze 5400 Milliarden kleiner Rechnungen.

Solche Überlegungen haben dazu geführt, dass bei der Viterbi-Implementierung speziell auf eine effiziente Abarbeitung geachtet wurde. Die Implementierung baut auf der schlanken Version eines Markow-Modell-Taggers von Lager (1997) auf. Es folgen jeweils die Definitionen der wesentlichen Prädikate mit anschliessender ausführlicher Diskussion.

```
% most_probable_hmm_path/2
most_probable_hmm_path(Output, MaxPath) :-
        probable_paths(Output, [1-[start]], PPaths),
        keymax(PPaths, _M, Path1),
```

```
        lists:reverse(Path1, [start|MaxPath]).
```

Um den wahrscheinlichsten Pfad (`MaxPath`) zu bestimmen, werden die relevanten Pfade (`PPaths`) ermittelt. Aus `PPaths` ist der Pfad mit der höchsten Wahrscheinlichkeit (`Path1`) massgebend. Das Prädikat `keymax/3` findet in einer Liste von `Num-Val`-Paaren dasjenige mit dem höchsten `Num`-Wert. `Path1` muss schliesslich noch umgedreht werden, gleichzeitig wird auch der Startzustand, im Atom `start` verkörpert, entfernt.

```
% probable_paths/3
probable_paths([], Delta_T, Delta_T).
probable_paths([Out|Output], Delta_t, Delta_T) :-
        findall(P-MaxPath,
                (   model:outprob(Out, State, POut),
                    max_delta(Delta_t, State, -1, [], MaxP, Max-
Path),
                    P is MaxP * POut),
                MaxDeltas),
        probable_paths(Output, MaxDeltas, Delta_T).
```

Die Berechnung der relevanten Pfade (`Delta_T`) geschieht rekursiv für jede Ausgabe. Die Berechnung ist zu Ende, wenn keine Ausgabe mehr bearbeitet werden muss. Im rekursiven Fall werden für die aktuelle Ausgabe (`Out`) und die maximalen Vorgänger-Deltas (`Delta_t`) die maximalen Deltas (`MaxDeltas`) aller relevanten aktueller Zustände berechnet. Ein Delta ist ein komplexer Term, der aus einer Wahrscheinlichkeit und einem Pfad zusammengesetzt ist: `P-MaxPath`. Je ein maximales Delta (`MaxP` und `MaxPath`) wird ausschliesslich für Zustände bestimmt, welche diese Ausgabe auch tatsächlich ausgeben. Dies schränkt die Anzahl der Zustände wesentlich ein.

```
% max_delta/6
max_delta([], _State, PMax, Path, PMax, Path) :- !.
max_delta([P1-[State1|States]|Delta_t], State, PAkku, Path, R,
          Result) :- !,
        (   P1 < PAkku
        ->  max_delta(Delta_t, State, PAkku, Path, R,Result)
        ;   model:transprob(State1, State, PT), !,
            P2 is PT * P1,
            (P2 > PAkku
```

```
-> max_delta(Delta_t, State, P2, [State,State1|States],
             R, Result)
;  max_delta(Delta_t, State, PAkku, Path, R,
             Result),!)).
```

Das Berechnen des maximalen Deltas eines Zustands (`State`) wird rekursiv gelöst. Die Berechnung ist abgeschlossen, wenn kein Vorgängerknoten-Delta mehr zu bearbeiten ist. Im rekursiven Fall muss jedes Vorgängerknoten-Delta entweder ignoriert werden, weil es keine Chance hat, einen grösseren Wert als das bisherige Maximum (`PAkku`) zu generieren[13], oder mit der Übergangswahrscheinlichkeit vom Vorgänger zum aktuellen Zustand (`PT`) multipliziert werden. Falls das Resultat (`P2`) grösser als `PAkku` ist, wird es zum neuen Zwischenresultat.

9.1.1 Unbekannte Konstellationen behandeln

Bei der Verwendung eines Hidden-Markow-Modells kann das Problem entstehen, dass eine unbekannte Ausgabe oder eine unbekannter Übergang auftaucht, d.h. eine Ausgabe bzw. ein Übergang, die/der im Trainingskorpus nicht vorkam. Wird dieses Problem nicht behandelt, ergibt die Berechnung des wahrscheinlichsten Pfades kein Resultat. Wie im Abschnitt 3.3.3 (S. 24) beschrieben, gibt es mehr oder weniger anspruchsvolle Wege, das Problem der Wahrscheinlichkeiten, die den Wert Null haben, anzugehen. Meist wird das Modell dabei künstlich um Übergänge erweitert, was allerdings die unbekannten Ausgaben nicht behandelt.

Wenn in der Anwendungsphase ein unbekannter Kontext auftaucht, ist es sinnvoll, nur eine OK-Operation durchzuführen. Da eine solche Struktur offenbar nie im Training vorgekommen ist, soll die Struktur nicht verändert werden. Es wäre auch denkbar, unbekannte Kontexte auf irgendeine Weise zu verallgemeinern und dann nochmals im Modell nachzusehen, ob nun ein ähnlicher Kontext zur Anwendung kommen könnte. Die Ausgabe unbekannter Kontexte erhält einen kleinen Wahrscheinlichkeitswert zugewiesen, um den Wert Null zu vermeiden. Analog dazu werden unbekannte Übergänge mit einer kleinen Wahrscheinlichkeit ausgestattet. Die verwendeten Zahlen wurden im Zuge einer Evaluierung auf 0.00001 für unbekannte Ausgaben und 0.000001 für Übergänge festgelegt. Dieses einfache Verfahren verhindert ein Aufblähen des Modells.

[13] Siehe Test: `P1 < PAkku`. `P2 is P1 * PT`, wobei `P2` nur dann interessant ist, wenn es grösser als `PAkku` ist. Ist `P1` kleiner als `PAkku`, wird es auch nach Multiplikation mit dem grössten möglichen Wert von `PT`, nämlich $\texttt{PT} = 1$ immer noch kleiner sein als `PAkku`.

9.2 Schritt 6: Anwendung der Editieroperationen

Die Transformationen sind eine Anweisung, wie vorgegangen werden kann, um einen Syntaxbaum in einen anderen bzw. neuen Baum umzuwandeln. Die Anzahl und Art der Transformationen ergeben die Distanz zweier Daten.

Wie schon im Abschnitt 7.5.2 (S. 50) erwähnt, unterscheidet sich die Situation vom Editierdistanzproblem, indem der Zielbaum nicht vorhanden ist. Die Selektion der Editieroperationen im Anwendungsfall beruht auf der Verwendung eines Modells. Die komfortable Situation, beide Bäume zur Hand zu haben, tritt nur in der Trainingsphase auf. Eine solche Selektion bringt zwangsläufig Fehler mit sich. Also muss in der Anwendung der Operationen auch der Fall abgedeckt sein, unpassende Operationen ausgewählt werden, die bei unbedachter Anwendung inkonsistente Baumstrukturen zur Folge haben. Trotzdem soll wenigstens versucht werden, auch die problematischen Operationen durchzuführen. Je komplexer die Operation ist, umso stärker fallen solche Unterschiede ins Gewicht.

9.2.1 OK

Der einfachste Fall tritt mit der OK-Operation ein. Es muss nichts getan werden. Auch ein fälschlicherweise berechnetes OK richtet keinen Schaden an und wird deshalb als nicht destruktiv angesehen. Wo nichts getan wird, entstehen auch keine neuen Fehler.

9.2.2 DELETE

Die DELETE-Operation könnte bei sicheren Daten blindlings und ohne die Berücksichtigung von Tochterknoten durchgeführt werden. Da beim Postorder-Durchlauf immer zuerst die Kinder behandelt werden, sind schon alle Kinder, die ursprünglich unter einem zu löschenden Knoten hängen, wegbewegt oder selbst gelöscht worden. Doch im Anwendungsfall, d.h. wenn die wahrscheinlichste Operation eines Knotens die richtige Operation ersetzen muss, kommt es öfter vor, dass ein Knoten gelöscht wird, der noch Kinderknoten hat. Obwohl es offensichtlich nicht korrekt sein kann, dass noch Kinder da sind, werden vorsichtigerweise alle noch vorhandenen Kinderknoten zur Mutter des zu löschenden Knoten bewegt, weil sie ja irgendwohin bewegt werden müssen. Weil das Löschen eine destruktive Operation ist, wird ein falsches DELETE den Baum weiter vom richtigen Zielbaum entfernen.

9.2.3 INSERT

Die Einfügeoperation ist durch den Baumvergleichsalgorithmus immer am ersten Tochterknoten festgebunden. Dies hilft, zu vermeiden, dass ein Knoten zu einem einzufügenden Knoten bewegt werden soll, der noch nicht existiert. Ein Knoten wird zwischen dem ersten Kinderknoten und der bisherigen Mutter des Kinderknotens eingefügt.

9.2.4 REPLACE

Die REPLACE-Operation ist die einfachste aller Transformationen, bei denen wirklich etwas getan wird. Die Beschriftung eines Knotens wird ersetzt.

9.2.5 Legale MOVE-Zielknoten ermitteln

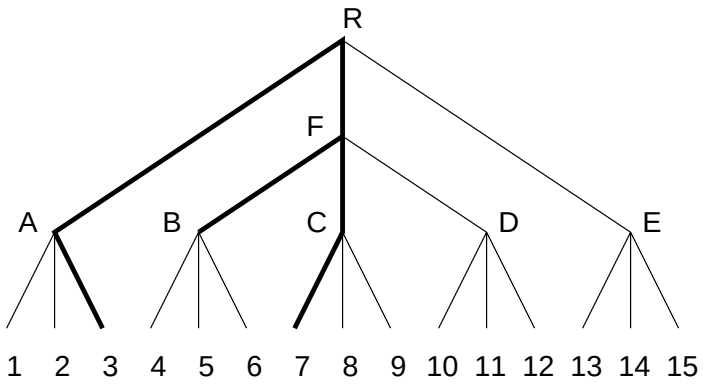


Abbildung 13: Legale Zielknoten für B

Aufgrund der Fehleranfälligkeit der MOVE-Operation lohnt es sich, in der Anwendung des MOVE immer zuerst die möglichen Zielknoten im aktuellen Baum zu bestimmen. Betrachtet man Abbildung 13 , ist es offensichtlich, dass für den Knoten B jeweils alle Nichtterminalknoten in Betracht gezogen werden können, die auf der fett gezeichneten Linie liegen, welche sich aus drei Pfaden zusammensetzt. Um diese Linie zu bestimmen, kommt folgendes Vorgehen zum Einsatz:

1. Bestimme den vordersten und den hintersten Terminalknoten T_V und T_H, welche der zu verschiebende Knoten K dominiert. Im Beispiel aus der Abbildung 13 ist $T_V = 4$ und $T_H = 6$.

2. Bestimme den Startknoten T_S, welcher eine Position weiter links von T_V steht. Im Beispiel ist $T_S = 3$.

3. Bestimme den Endknoten T_E, welcher eine Position weiter rechts von T_H steht. Im Beispiel ist $T_E = 7$.

4. Bestimme die drei Pfade[14]:

   - P_S führt vom Startknoten T_S zur Wurzel. Hier: $P_S = (A, R)$.
   - P_K führt vom Verschiebeknoten K zur Wurzel. Hier: $P_K = (F, R)$.
   - P_E führt vom Endknoten T_E zur Wurzel. Hier: $P_E = (C, F, R)$.

5. Bestimme den Scheitelknoten G. G ist der tiefste gemeinsame Knoten von P_S und P_E. Der Scheitelknoten ist der höchste aller möglichen Zielknoten. Hier: $G = R$.

6. Die Menge aller möglichen Zielknoten Z besteht nun aus dem Pfadabschnitt P_S bis und mit Scheitelknoten G, dem Pfadabschnitt P_K bis G und dem Pfadabschnitt P_E bis G. In unserem Beispiel ergibt sich: $Z = (A, R)+(F, R)+(C, F, R) = \{A, R, F, C\}$.

9.2.6 MOVE

Im Bereich der Syntaxbaumtransformation ist die weitaus komplexeste Operation das MOVE. Sie ist besonders sensibel, was die Reihenfolge betrifft. Der Pfad als zusätzliche Information, die sie braucht, schraubt die Komplexität in die Höhe. Falsche Bewegungen können unter Umständen viel Unheil anrichten, beispielsweise ganze Teilbäume absägen, Terminalreihenfolge verändern. Das Verschieben eines Knotens läuft nach dem folgenden Schema ab:

1. Ermittle den Zielknoten mittels Pfad aus dem Originalbaum (siehe Abschnitt 8.2 (S. 55)).

2. Ist die Kategorie des Zielknotens gleich wie die verlangte Kategorie im Pfad?

   Ja: Ermittle alle legalen Zielknoten im aktuellen Baum anhand des MOVE-Knotens, des vorhergehenden und des nachfolgenden Terminalknotens (siehe Abschnitt 9.2.5 (S. 71)).

   Nein: Beende die Anwendung der MOVE-Operation.[15]

[14] Pfad wird hier definiert als Weg von einem Startknoten zur Wurzel, wobei der Startknoten nicht inbegriffen ist.

[15] Es wird angenommen, dass bei Nichtübereinstimmung der Kategorien die vorgeschlagene MOVE-Operation nicht auf diese Baumstruktur passt.

3. Ist der Zielknoten ein Element der Menge aller möglichen Zielknoten?

   Ja: Lösche den gesamten Teilbaum des MOVE-Knotens aus dem bisherigen Baum und füge ihn als zusätzliches Kind des Zielknotens in den Baum ein. Probiere danach alle Bewegungen, die sich auf der Wartebank befinden, erneut aus.

   Nein: Setze die Operation auf die Wartebank.

   Es gibt zwei Gründe, dass ein Zielknoten kein legaler Zielknoten ist:

   (a) Es würden in der Bewegung überkreuzende Kanten entstehen bzw. die fixe Reihenfolge der Terminale würde durcheinandergebracht.

   (b) Der Zielknoten ist im aktuellen Transformationsbaum ein Nachkomme des MOVE-Knotens. Diese Situation kann vorkommen, wenn der Zielknoten eigentlich schon aus dem Teilbaum des MOVE-Knotens herausbewegt worden sein sollte, aber nicht wurde, weil

      i. bei der Anwendung ein Fehler geschehen ist und der Nachkomme gar nie verschoben wird. In diesem Fall soll nichts verschoben werden, da nicht klar würde, wo der Nachkomme hingehört. Oder weil

      ii. überkreuzende Kanten dies vorerst verhindert haben, d.h. der Nachkomme sitzt auf der Wartebank. In dem Falle muss eine blockierende Situation auf der Wartebank verhindert werden.

Eine Schwierigkeit stellt der Umgang mit potentiell “gefährlichen” MOVE-Operationen dar. Bei jeder MOVE-Operation, deren Zielknoten nicht zugänglich ist, besteht das Risiko, dass dem Baum Schaden zugefügt wird. Diese Operationen sind aber leider nicht selten. Deshalb wurde das Wartebank-System eingeführt. Doch blockierende Situationen in der Wartebank treten auch häufig auf. Also werden unter Umständen Operationen trotz Risiko durchgeführt, um eine blockierende Situation zu vermeiden. So kann es trotz aller Sicherheitsmassnahmen vorkommen, dass vereinzelte Bäume unbemerkt Schaden nehmen. Dies kommt aber selten vor. Eine umsichtigere Lösung konnte aus Zeitgründen leider nicht mehr konzipiert und realisiert werden und wird deshalb Gegenstand zukünftiger Arbeiten werden müssen.

9.2.7 Beispiel

Gegeben sind der Baum T und die Menge der Transformationen D.

Gesucht wird T'.

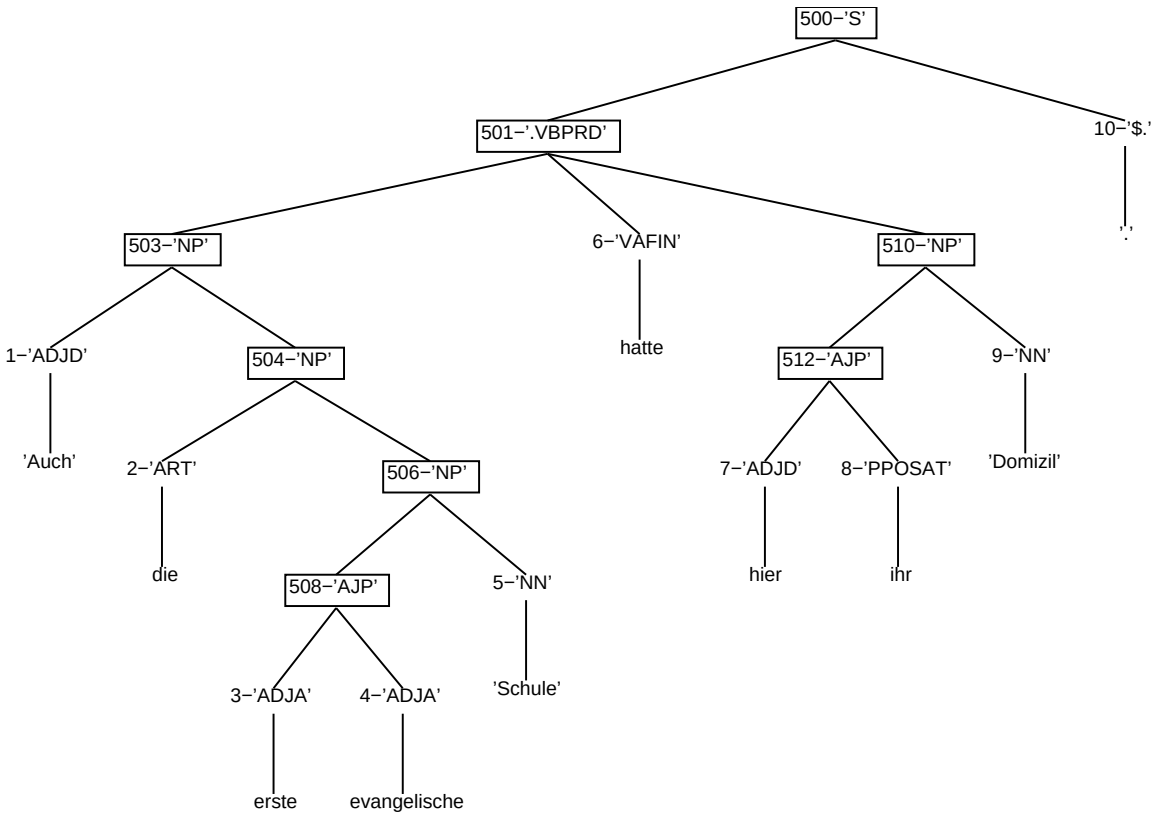


Abbildung 14: Der Originalbaum T

Dieses Beispiel behandelt den Satz: „Auch die erste evangelische Schule hatte hier ihr Domizil“. Der Satz stammt aus dem NEGRA-Korpus. T (siehe Abbildung 14) zeigt die Struktur, die der GOJOL-Parser[16] dem Satz zugeordnet hat. T ist der zu transformierende Baum, während T' gesucht wird. Für eine Abbildung des Zielbaums siehe Abbildung 19 (S. 79). Die Zielstruktur T' stimmt mit der NEGRA-Struktur überein. D wurde in diesem Fall durch einen Baumvergleich bereits ermittelt.

Aus dem ursprünglichen D kann eine sinnvolle Anwendungssequenz abgeleitet werden. Alle OK-Operationen werden ignoriert, da dort ja keine Aktion notwendig ist. Das grundsätzliche Vorgehen beginnt mit den einfachen Operationen, wie REPLACE und DELETE, und führt erst danach die komplexen Operationen durch, wie INSERT und MOVE, weil die komplexen Operationen teilweise störanfällig sind. Die INSERT-Operationen finden

[16]Der Entwickler, V. Gojol, kann bei Interesse am statistischen Parsing-System per Email kontaktiert werden:
`gojol@sunu.rnc.ro`

vor den MOVE-Operationen statt, weil Knoten teilweise auch auf eingefügte bewegt werden müssen. Ein noch nicht gelöschter Knoten kann zum Beispiel eine zulässige MOVE-Operation unter Umständen verhindern. Es kommt auch vor, dass ein MOVE auf einen eingefügten Knoten vorgenommen werden soll, deshalb müssen erst die Einfügungen gemacht werden. Wir ordnen also die Folge D um. Zuerst sollen die REPLACE-Operationen, dann die DELETE- bzw. INSERT-Operationen und schliesslich die MOVE-Operationen durchgeführt werden. D als einem Listing von PROLOG-Fakten:

```
   % rpl(SatzNr, KnotenNr, FromTag, ToTag)
rpl(17, 1, 'ADJD', 'ADV').
rpl(17, 7, 'ADJD', 'ADV').
rpl(17, 512, 'AJP', 'NP').

% del(SatzNr, KnotenNr, Tag)
del(17, 508, 'AJP').
del(17, 506, 'NP').
del(17, 504, 'NP').
del(17, 510, 'NP').
del(17, 501, '.VBPRD').

% mov(SatzNr, KnotenNr, Tag, Pfad)
mov(17, 503, 'NP', toRoot).
mov(17, 6, 'VAFIN', toRoot).
mov(17, 7, 'ADJD', toRoot).
mov(17, 512, 'AJP', toRoot).
mov(17, 9, 'NN', path(1,[0],'NP')).
```

1. Als erstes sollen also die REPLACE-Operationen durchgeführt werden, weil diese vergleichsweise einfach sind. Aus dem Listing entnimmt man:
   ```
   rpl(17, 1, 'ADJD', 'ADV').
   rpl(17, 7, 'ADJD', 'ADV').
   rpl(17, 512, 'AJP', 'NP').
   ```
   Die Beschriftungen der Knoten 1, 7 und 512 von T sind betroffen und werden ersetzt. Den resultierenden Baum T_1 sieht man in der Abbildung 15 auf der nächsten Seite. Danach sind die DELETE-Operationen an der Reihe.

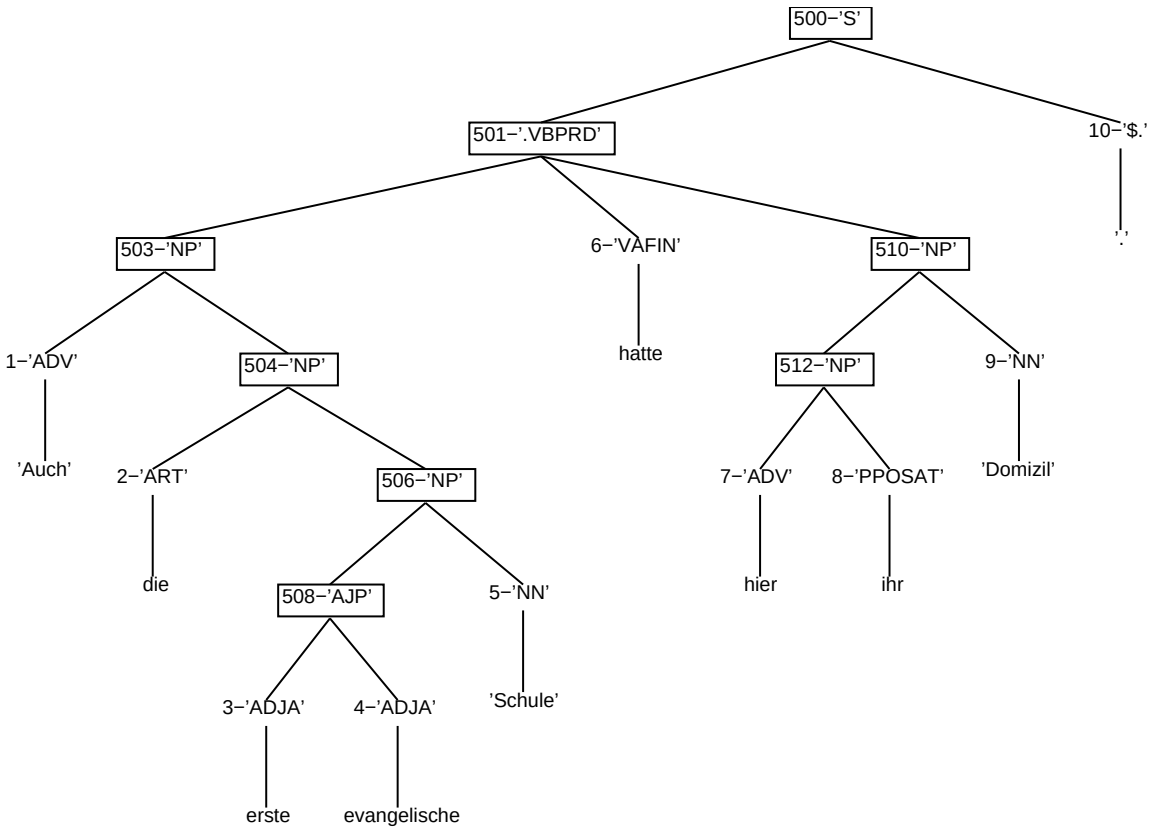


Abbildung 15: Baum T_1: Nach Durchführung aller REPLACE-Operationen

2. `del(17, 508, 'AJP').` Der Knoten 508 mit der Beschriftung 'AJP' soll gelöscht werden. Seine zwei Kinder (die Knoten Nr. 3 und 4) werden eine Stufe nach oben verschoben und in derselben Reihenfolge unter die neue Mutter (Knoten Nr. 506) gehängt (siehe Abbildung 16 auf der nächsten Seite).

3. `del(17, 506, 'NP').` Der Knoten 506 mit der Beschriftung 'NP' wird als nächstes gelöscht. Es ist der Knoten, der soeben zwei neue Kinder erhalten hat, doch das macht keinen Unterschied. Der Knoten 506 wird gelöscht und seine Kinder rutschen eine Stufe höher.

4. `del(17, 504, 'NP').`
   `del(17, 510, 'NP').`
   `del(17, 501, '.VBPRD').`
   Auch die Knoten 504, 510 und 501 werden nacheinander gelöscht. Der resultierende Baum ist in der Abbildung 17 (S. 78) abgebildet.

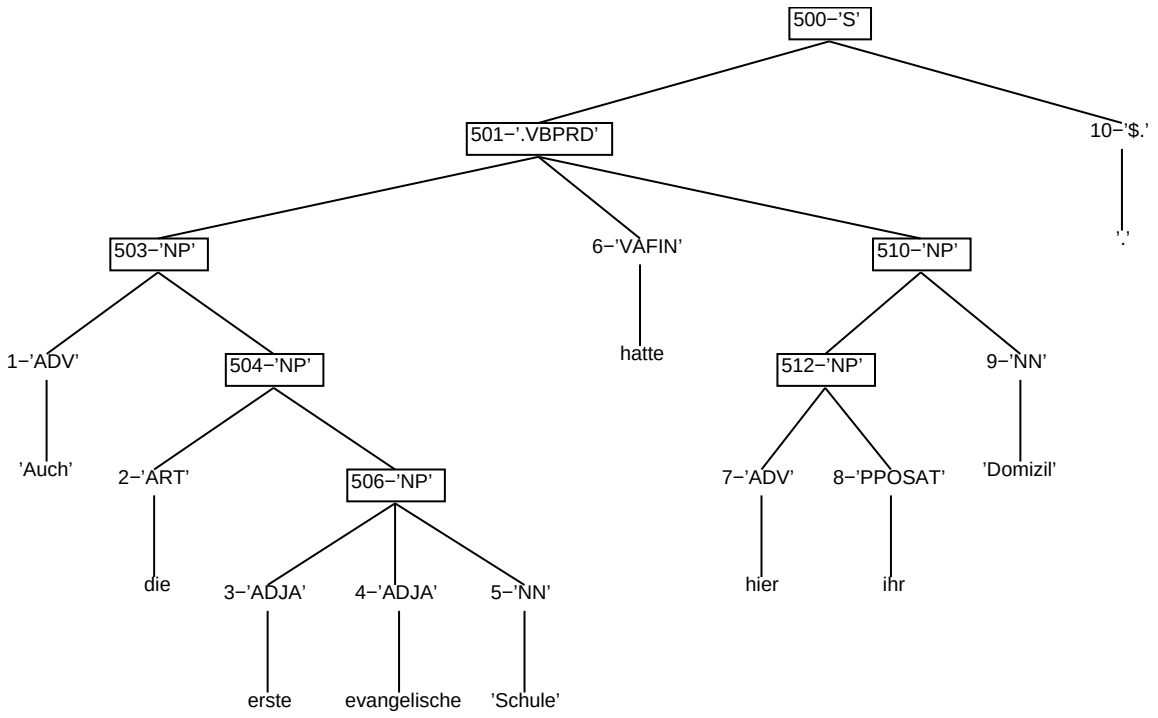


Abbildung 16: Baum T_2: Knoten 508 gelöscht

5. INSERT-Operationen gibt es keine. Bedingt durch den Baumvergleichsalgorithmus kommt es äusserst selten vor, dass in einem Satz gleichzeitig DELETE- und INSERT-Operationen vorkommen.

6. Zuletzt werden die MOVE-Operationen durchgeführt.
   `mov(17, 503, 'NP', toRoot).`
   `mov(17, 6, 'VAFIN', toRoot).`
   Der spezielle Pfad `'toRoot'` bedeutet, dass der Knoten ein Kind des Wurzelknotens werden soll. Diese Bewegung ist sehr häufig, wenn man das NEGRA-Format als Zielstruktur hat, weil NEGRA sehr flache Strukturen bevorzugt. Die Knoten 503 und 6 würden also nach oben bewegt, wenn sie nicht schon dort situiert wären.

7. Der Baum T_7 nach der Operation `mov(17, 7, 'ADJD', toRoot)`
   ist in Abbildung 18 auf der nächsten Seite zu sehen. Der Knoten 7 wurde zur Wurzel geschoben.

8. Die Operation `mov(17, 512, 'AJP', toRoot)` hat sich durch die Löschoperationen schon erledigt.

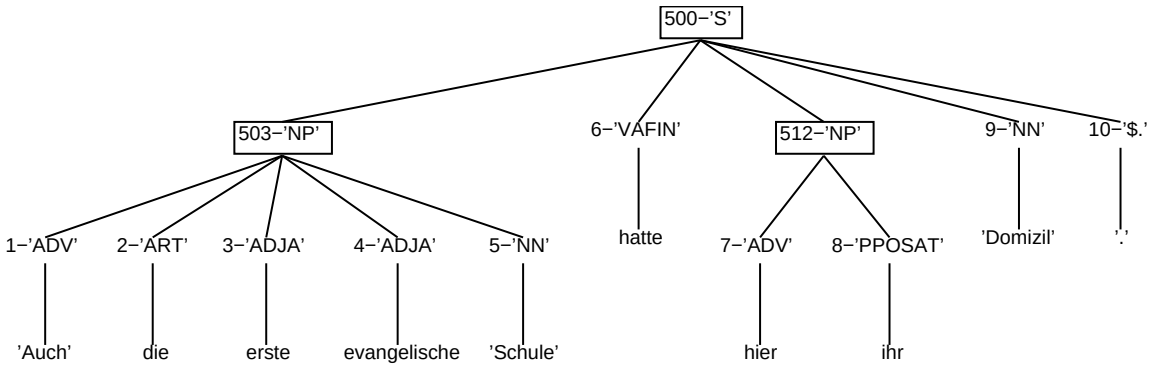


Abbildung 17: Baum T_6: Knoten 506, 504, 510 und 501 gelöscht

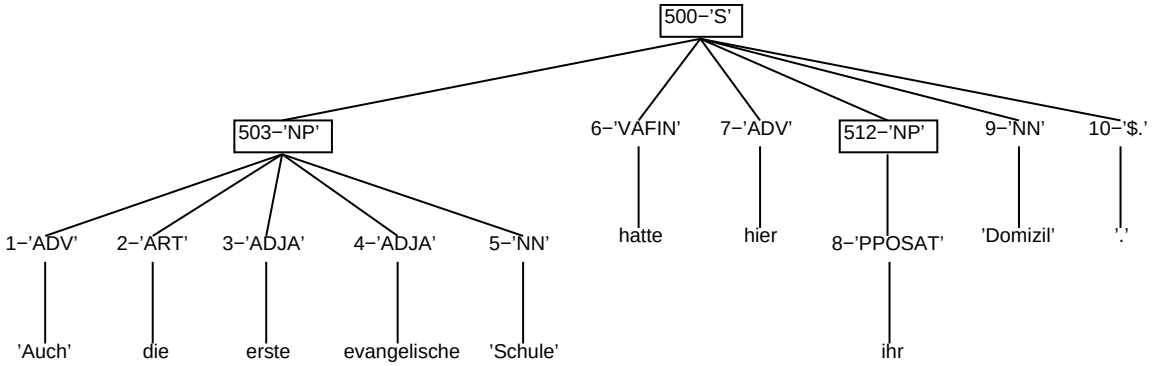


Abbildung 18: Baum T_7: Knoten 7 zur Wurzel verschoben

9. Der Pfad in `mov(17, 9, 'NN', path(1,[0],'NP'))` ist etwas ausführlicher als bisher. Seine Bedeutung lautet: Gehe im Originalbaum (siehe Abbildung 14 (S. 74)) vom Knoten 9 einen Schritt nach oben (510-`'NP'`), von dort zum ersten Kind `[0]` (512-`'AJP'`, in T_7: 512-`'NP'`). Und hänge dort den Teilbaum, der beim Knoten 9 startet, als Kind an. Der resultierende Baum T' ist in Abbildung 19 auf der nächsten Seite zu finden.

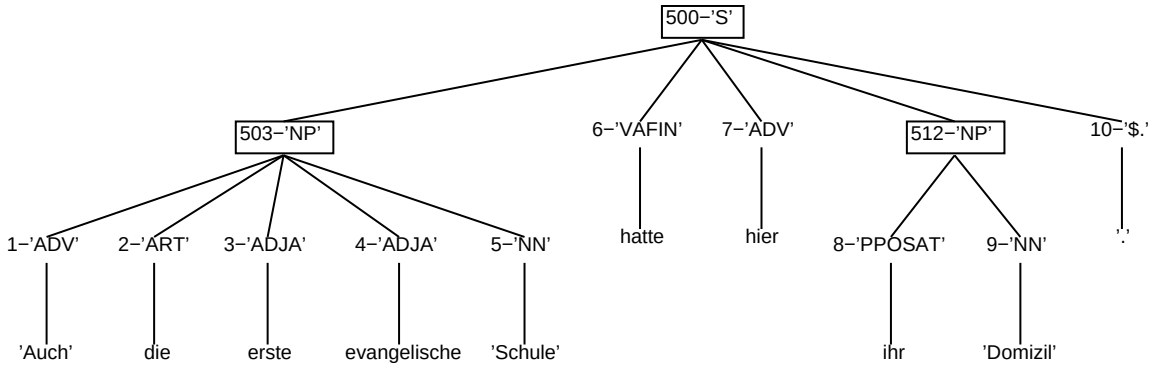


Abbildung 19: Baum T': Knoten 9 unter die NP verschoben

10 Evaluierung

Das entwickelte System wurde implementiert und anschliessend evaluiert. Die Evaluierungsmethoden und -ergebnisse werden in diesem Kapitel ausführlich diskutiert.

10.1 Methoden zur Evaluierung

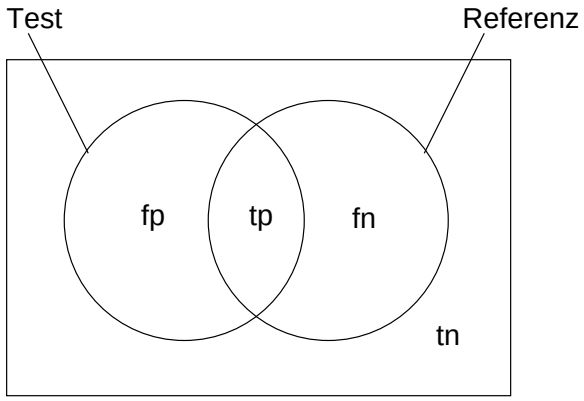
Im vorliegenden Fall sind verschiedene Evaluierungsarten denkbar. Die Ausgabe des Systems besteht aus Baumstrukturen. Es liegt also auf der Hand, Baumstrukturen zu vergleichen. Der probabilistische Bereich des Systems betrifft die Berechnung der wahrscheinlichsten Transformationen, deshalb sind auch Vergleiche von Transformationen durchführbar.

10.1.1 Baumvergleich mit eigenem Systemteil

In der Trainingsphase werden jeweils zwei Bäume miteinander verglichen. Die Distanz wird anhand einer Menge von Transformationen wiedergegeben. Die Menge der Transformationen sind ein Mass dafür, wie ähnlich bzw. verschieden zwei Bäume sind. Die einfachste aller Evaluierungsmethoden für unser System wäre also die Anwendung des Baumvergleichs-Programms auf die Ausgabe der Anwendung. Doch dieses Ähnlichkeitsmass ist nicht einfach zu interpretieren und lässt v.a. keine Vergleiche mit anderen Arbeiten zu. Aus diesen Gründen wird davon abgesehen, hier näher darauf einzugehen.

10.1.2 PARSEVAL

Es wird ein Testkorpus mit einem Referenzkorpus verglichen. Dabei gilt die Annahme, dass die Strukturen im Referenzkorpus richtig und die Strukturen des Testkorpus zu beurteilen sind. Ausgehend von der Abbildung 20 auf der nächsten Seite kann man feststellen, dass es in einer Kreuzklassifikation vier Fälle gibt. **True** bedeutet, dass sich Test- und Referenzkorpus einig sind. Die Uneinigkeit stellt **false** fest. **Positive** bedeutet, dass im Testkorpus die Struktur (ob richtig oder falsch) enthalten ist. **Negative** heisst, dass die Struktur (ob richtig oder falsch) nicht im Testkorpus enthalten ist. **True positives** tp heissen demnach alle richtigen Teilstrukturen vom Testkorpus. Alle falschen Teilstrukturen vom Testkorpus fp heissen **false positives**. Alle fehlenden Teilstrukturen vom Testkorpus, die im Referenzkorpus vorkommen, heissen **false negatives** fn.



tp = true positives

fp = false positives

fn = false negatives

tn = true negatives

Abbildung 20: Evaluieren mit Kreuzklassifikation

Bereits 1991 wurde das weit verbreitete Evaluationsverfahren PARSEVAL von Black u. a. (1991) und Harrison u. a. (1991) entwickelt, um die Vergleichbarkeit von Phrasenstrukturen zu erhöhen, die von Parsern aufgebaut werden. PARSEVAL misst die folgenden drei Werte:

Die Ausbeute (*recall*) Die Ausbeute R bezeichnet den Anteil an richtigen Phrasenstrukturen im Testkorpus in Bezug auf das Referenzkorpus.

(10.1) $$R = \frac{tp}{tp + fn} \mathrel{\hat{=}} \frac{\text{\# korrekte aus Testkorpus}}{\text{\# Total aus Referenzkorpus}}$$

Die Präzision (*precision*) Die Präzision P bezeichnet den Anteil an richtigen Phrasenstrukturen im Testkorpus in Bezug auf das Testkorpus.

(10.2) $$P = \frac{tp}{tp + fp} \mathrel{\hat{=}} \frac{\text{\# korrekte aus Testkorpus}}{\text{\# Total aus Testkorpus}}$$

Die überkreuzenden Strukturen (*crossing brackets*) Wie es Clematide (2002, S.1) ausdrückt:

„Das Mass der sich überkreuzenden Klammerungen ist die durchschnitt-

liche Anzahl von Strukturen pro Satz, die im Testkorpus vorkommen und von mindestens einer Struktur aus dem Referenzkorpus nur die Anfangs- oder Endposition überspannen."

Ein Beispiel:

(10.3) $(A\ (B\ C))$ vs. $((A\ B)\ C)$

Ausbeute und Präzision können sowohl rein strukturell berechnet werden, was bedeutet, dass eine Konstituente nur durch ihre Start- und Endposition definiert ist, als auch verschärft, wobei zusätzlich noch die Konstituentenkategorie übereinstimmen muss, damit die Phrasenstruktur als korrekt erkannt gilt.

Es ist bekannt, dass diese drei Masse den Nachteil haben, dass sie bei unterschiedlich tiefen und feinen Analysen aus linguistischer Sicht schnell verzerrend wirken, insbesondere dann, wenn das Referenzkorpus flache Strukturen enthält. In unserer Aufgabe steckt allerdings die Zielsetzung, dass sich Strukturen angleichen, deshalb ist PARSEVAL das richtige Mass.

Die Interpretation der PARSEVAL-Resultate ist jedoch schwierig. Beispielsweise sind die Zahlen schlechter, wenn die Referenz sehr flache Strukturen aufweist, da sich hier ein falscher Knoten mehr auswirkt. Je weniger Struktur die Referenz aufweist, umso mehr fallen die falschen Strukturen ins Gewicht. Ausbeute und Präzision sind Masse für die Ähnlichkeit von zwei Strukturen. Eine kleine Differenz der beiden Masse bedeutet eine ähnliche Anzahl von Teilstrukturen im Test- und im Referenzkorpus.

Zusätzliche interessante Zahlen Das F-Mass ist eine gewichtete Kombination von Ausbeute R und Präzision P und nach der folgenden Formel definiert:

(10.4) $$F = \frac{(\beta^2 + 1)PR}{\beta^2 P + R}$$

β ist ein Parameter, der die Gewichtung der Ausbeute und der Präzision bestimmt. Für ein gleiches Gewicht beider Werte wird $\beta = 1$ gesetzt (siehe beispielsweise Brants (1999a)).

Die exakte Übereinstimmung (engl. *exact match*) zeigt an, welcher Anteil der Sätze, für die ein Parser Strukturen, Wortarten und Phrasenkategorien berechnet hat, identisch mit den Strukturen und Beschriftungen der Referenz sind. Dies ist ein äusserst striktes Mass, weil ein einziger, winziger Fehler – wie zum Beispiel eine falsche Beschriftung eines

Knotens – bereits die ganze Struktur verdirbt, und sie somit in einem Topf mit gänzlich falschen Strukturen landet. Deshalb wird die exakte Übereinstimmung üblicherweise, falls überhaupt, dann nur in Kombination mit der Ausbeute und der Präzision angegeben.

Um die Strenge des Masses etwas zu lockern, kann man auch hier statt der exakten, die strukturelle Übereinstimmung (engl. *structural match*) messen. Sie beachtet nur die Strukturen, nicht die Beschriftungen. Die strukturelle Übereinstimmung misst, welcher Anteil der Sätze, für die ein Parser Strukturen berechnet hat, identisch sind mit den Strukturen des Referenzkorpus.

10.1.3 Editieroperationen evaluieren

Es können auch einzelne Editieroperationen beurteilt werden. In der Testphase sind jeweils beide Operationen gegeben: Die richtige (**goldene**) Operation aus dem Baumvergleich und die wahrscheinlichste (**silberne**) Operation, welche aus dem Modell berechnet wurde. Mit einem Vergleich der beiden Operationen können beispielsweise die folgenden drei Operationsgruppen gebildet werden.

Gute Operationen Wenn die silberne Operation mit der goldenen exakt übereinstimmt, ist dies als Erfolg zu werten. Der Baum wird nach Anwendung der silbernen Operation dem Zielbaum ähnlicher sein.

Indifferente Operationen Die silberne Operation stimmt nicht überein mit der goldenen. Die silberne schlägt eine zu kleine Veränderung am Baum vor. Die goldene Operation verändert mehr am Baum. Der Baum wird nach Anwendung der silbernen Operation dem Zielbaum weniger ähnlich sein, als wenn die goldene angewendet wird. Aber er wird auch nicht noch weiter davon entfernt sein als vor der Anwendung. Es tritt zumindest keine Verschlechterung des Zustandes auf.

In diese Kategorie gehören die folgenden Fälle:

- Eine Beschriftung bleibt falsch: ok statt rpl; rpl(S) statt rpl(G)
- Ein überzähliger Knoten bleibt drin: ok oder rpl statt del
- Ein Knoten bleibt am falschen Ort: nichts statt mov
- Ein fehlender Knoten wird nicht eingefügt: mov statt insmov
- Ein Knoten bleibt am falschen Ort und ein fehlender Knoten wird nicht eingefügt: nichts statt ins mov

Schlechte Operationen Die silberne Operation schlägt eine Veränderung vor, die den Baum noch weiter vom Zielbaum entfernt. Sie verändert zuviel. Dies ist als Misserfolg zu werten. Eine genauere Betrachtung dieser Gruppe kann einen Hinweis darauf geben, ob eine Operation besonders häufig Fehler verursacht.

Zu dieser Kategorie zählen die Konstellationen:

- Eine richtige Beschriftung wird verfälscht: rpl statt ok
- Ein nicht überzähliger Knoten wird gelöscht: del statt ok oder rpl
- Ein richtig stehender Knoten wird bewegt: mov statt nichts
- Ein nicht richtig stehender Knoten wird an einen falschen Ort bewegt: mov(S) statt mov(G)
- Ein Knoten wird fälschlicherweise eingefügt: ins mov statt mov; ins mov anstelle von keiner weiteren Operation
- Ein einzufügender Knoten bekommt die falsche Beschriftung: ins(S) statt ins(G)

10.2 Verteilungen

Der Ambiguitätsgrad eines Modells kann mit statistischen Verteilungen verdeutlicht werden. Je mehr verschiedene Zustände einem Zustand folgen und je mehr Zustände Urheber einer Ausgabe sein können, umso mehr Ambiguität ist im Modell enthalten.

Die Verteilung der Ambiguität in einem konkreten Modell mit 3266 Zuständen ist in der Abbildung 21 auf der nächsten Seite mit logarithmischer Skala dargestellt. Die Y-Achse zeigt die Häufigkeit, dass eine Transformation von x verschiedenen Transformationen gefolgt wird (in der Abbildung mit Übergangsverteilung bezeichnet) bzw. dass ein Kontext von x verschiedenen Transformationen ausgegeben wird (Ausgabeverteilung). x ist jeweils auf der X-Achse eingetragen.

Die maximal ambige Transformation `d('NP')` wird von 648 verschiedenen Transformationen gefolgt. Es sind 111 hochambige Übergänge, d.h. mehr als 10 verschiedene Nachfolge-Transformationen kommen bei einer Transformation infrage. Insgesamt sind es 725 ambige Transformationen und 2561 eindeutige Transformationsvorgänger. Der Startzustand wird von 92 verschiedenen Zuständen gefolgt, d.h. es gibt 92 verschiedene Transformationen am Satzanfang.

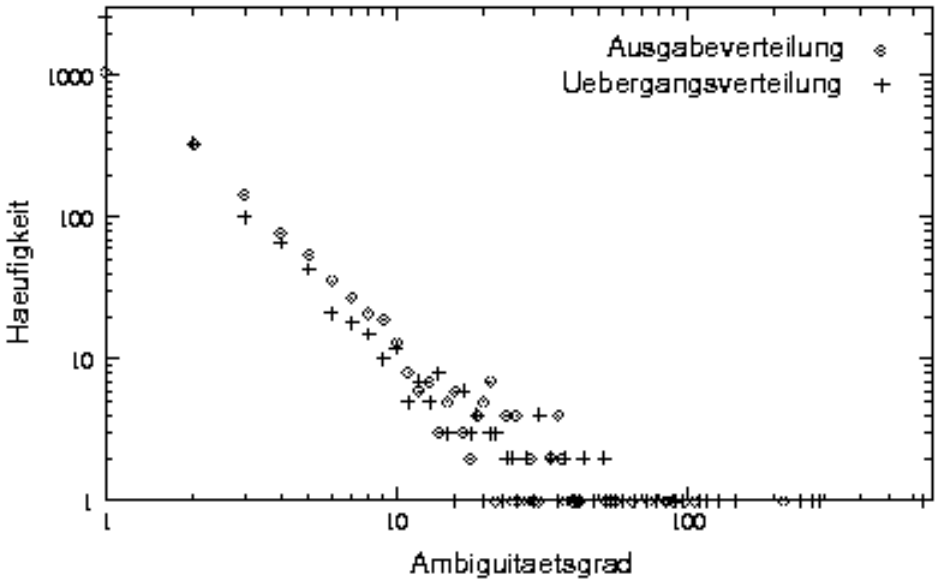


Abbildung 21: Verteilung der Ambiguitäten

Der Kontext `['NN','N1','N2',null]` wird von 217 Zuständen ausgegeben und kann deshalb mit 217 Transformationen in Beziehung stehen. Gerade dieser Kontext kommt in den Bäumen auch noch häufig vor. Es sind 98 hochambige Kontexte, d.h. mehr als 10 verschiedene Transformationen geben diesen Kontext aus. Gesamthaft sind es 817 ambige Kontexte und 1050 eindeutige. Knapp die Hälfte aller Kontexte wird von mehreren Zuständen ausgegeben. Diese Zahlenverhältnisse weisen insgesamt auf einen relativ hohen Ambiguitätsgrad im Modell hin.

10.3 Experimente und Ergebnisse

Die Durchführung der Experimente erfordert zwei Korpora. Ein Ausgangskorpus enthält die Ausgangsstrukturen T, welche vom System verändert werden sollen, und ein Zielkorpus liefert die Zielstrukturen T'. Beide Korpora beinhalten Strukturen zu den gleichen Sätzen in derselben Reihenfolge. Die zufällige Aufspaltung beider Korpora ergibt einen Trainings- und einen Testteil im Grössenverhältnis 9:1. Das probabilistische Modell wird mit dem Trainingsteil erstellt und mit dem Testteil wird die Evaluierung durchgeführt. Statistisch zuverlässigere Resultate können erreicht werden, indem die Experimente mehrmals (übli-

cherweise zehnmal) wiederholt und Durchschnittswerte der Resultate mit Standardabweichung angegeben werden. Man nennt dieses Vorgehen auch zehnfache Kreuzvalidierung.

In den durchgeführten Experimenten dienen Strukturen vom Lopar-Parser (siehe Schmid (2000)) als Ausgangskorpus und Teile vom NEGRA-Korpus (siehe Skut u. a. (1998) und Kapitel 2.3 (S. 11)) als Zielkorpus. Es wurden jeweils die folgenden Werte gemessen: Ausbeute, Präzision, F-Mass mit gleicher Gewichtung von Ausbeute und Präzision, überkreuzende Strukturen, exakte Übereinstimmung, Tagging-Genauigkeit, gute, indifferente und schlechte Operationen. Die verwendete Kontextinformation beim Evaluieren besteht aus den Tags der folgenden Knoten: Aktueller Knoten, Mutter, Grossmutter und linke Schwester.

Eine geeignete absolute Grössenordnung des Trainingsteils kann experimentell ermittelt werden, indem eine Reihe von Grössen getestet werden und eine Lernkurve erstellt wird. Die besten Resultate weisen auf die optimale Trainingsgrösse. Die Lernkurve für das F-Mass ergab eine erstaunlich flache Kurve, die bereits ab einem Trainings-/Testkorpus von 1600 Sätzen nicht mehr wesentlich steigt oder abfällt (siehe Anhang Abschnitt C.14 (S. 113)).

Die Aufgabe lautete, eine Ausgangsstruktur einer Zielstruktur anzunähern. Sie wurde mittels eines probabilistischen Modells gelöst, welches die Distanzen zwischen den zwei Strukturen abbildet. Aufgrund einiger Untersuchungen am Modell (Verteilungen siehe Abschnitt 10.2 (S. 84)) und theoretischer Überlegungen (Problem mit Pfad und Kontext siehe 8.4 (S. 63)) ist zu erwarten, dass das Ziel zwar erreicht wird, aber Einschränkungen in der Abbildungskapazität des Modells und in der Anwendung der Operationen befürchtet werden müssen. Diese Defizite werden sich wohl durch eher tiefe Erfolgsquoten in den Experimenten abzeichnen. Es ist ausserdem anzunehmen, dass die MOVE-Operation am meisten Fehler verursacht, einerseits aufgrund der im Abschnitt 8.4 (S. 63) angesprochenen Problematik der unterschiedlichen Spannweite von Kontextinformation und Pfadinformation, andererseits wegen der Schwierigkeiten in der Durchführung von fehlerhaften MOVE-Operationen (siehe 9.2 (S. 70)), welche Sicherheitsmassnahmen erfordern, die wiederum auch einige richtige MOVE-Operationen verhindern.

Die Abbildung 22 auf der nächsten Seite zeigt auf, welche Ähnlichkeitswerte die Original-Syntaxbäume von Lopar im Vergleich mit den NEGRA-Strukturen aufweisen. Alle Werte wurden mit Berücksichtigung der Beschriftung gemessen. Zusätzlich wurde eine rein strukturelle Messung der Ausbeute und der Präzision vorgenommen. Die Werte zeigen, dass die Ähnlichkeit zwischen Ausgangs- und Zielkorpus relativ niedrig ist. Die

Mass	**beschriftet (%)**	**strukturell (%)**
Ausbeute	47.13	76.67
Präzision	8.58	13.96
F-Mass	14.52	23.62
Exakte Übereinstimmung ..	0.00	
Kein Überkreuzen	32.58	
2 oder weniger Überkreuzen	57.97	
Tagging-Genauigkeit	49.99	

Abbildung 22: Ähnlichkeit zwischen Lopar und NEGRA (Vorher)

starke Differenz von Ausbeute und Präzision weist darauf hin, dass Lopar massiv tiefere Bäume aufweist im Vergleich mit den flachen NEGRA-Strukturen.

Die strukturelle Ausbeute von 76.67 % erscheint hoch, wird aber durch die tiefe Präzision von 13.96 % insofern relativiert, dass zwar viele korrekte Strukturen gefunden wurden, diese sind aber in einer grossen Masse von nicht korrekten Strukturen versteckt. Dementsprechend niedrig ist das F-Mass von 23.62 %. Überkreuzende Strukturen sind hier noch häufig, da nur gerade 57.97 % der Bäume zwei oder weniger Überkreuzende Strukturen enthalten. Die schwache Tagging-Genauigkeit von 49.99 % muss nicht bedeuten, dass ein schlechter Tagger am Werk war, sondern ist mehrheitlich auf ein abweichendes Tagset von Lopar zurückzuführen.

Zum Vergleich enthält die Abbildung 23 auf der nächsten Seite die Mittelwerte inklusive Standardabweichung des besten Experiments. Es wurden nur die beschrifteten Werte gemessen. Ausbeute und Präzision sind nun viel näher beieinander (66.00 % und 64.43 %), es wurden also viele überzählige Strukturen gelöscht. Darunter sicher auch ein paar, die eigentlich korrekt gewesen wären. Das Ziel, die Ausgangsstrukturen der Zielstruktur anzunähern, ist zweifellos erreicht. Dennoch sind nach der Transformation nur 12.79 % der Syntaxstrukturen identisch mit der Zielstruktur. Hierin zeigen sich eben die Schwierigkeiten und Probleme, die man erwarten musste. Die Reduzierung der überkreuzenden Strukturen ist hingegen sehr erfreulich: 96.18 % der Bäume enthalten nur zwei oder weniger überkreuzende Strukturen. Auch die Tagging-Genauigkeit von 94.40 % kann sich sehen lassen, vor allem, wenn man bedenkt, dass die Neubeschriftung ohne lexikalische Information geschieht.

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	66.00	1.333
Präzision	64.43	1.362
F-Mass	65.21	
Exakte Übereinstimmung ..	12.79	2.502
Kein Überkreuzen	73.62	3.558
2 oder weniger Überkreuzen	96.18	0.854
Tagging-Genauigkeit	94.40	0.700

Abbildung 23: Ähnlichkeit zwischen transformiertem Lopar und NEGRA (Nachher)

Es wurden auch die berechneten Operationen beurteilt. Die Werte sind in der Abbildung 24 zusammengestellt.

Mass	Wert	
	Mittel (%)	Standardabweichung
Total Operationen (nicht in %)	14639.30	319.146
Anteil gute Operationen	84.84	0.977
Anteil schlechte Operationen	8.47	0.514
Anteil indifferente Operationen	6.55	0.504
Anteil gute und indifferente	91.43	0.514
Anteil schlechte und indifferente	15.06	0.977
Anteil REPLACE von den schlechten .	3.55	0.493
Anteil DELETE von den schlechten ...	28.74	1.098
Anteil MOVE von den schlechten	36.60	1.521
Anteil falscher Pfad von den schlechten	30.91	1.297

Abbildung 24: Auswertung der berechneten Operationen

Es fällt auf, dass der Anteil richtig berechneter Operationen mit 84.84 % erfreulich hoch ist. Die Anteile der schlechten und indifferenten Operationen liegen in derselben Grössenordnung und ergeben zusammen 15.06 %. Alle berechneten Operationen bewirken die Annäherung der Bäume, welche oben besprochen wurde.

In der Abbildung 25 auf der nächsten Seite ist jeweils der Anteil einer Operation von den schlechten Operationen aufgeführt. Die falschen Pfade zusammen mit den unnötigen

MOVE-Operationen ergeben ca. die Hälfte aller schlechten Operationen. Es ist deutlich, dass MOVE insgesamt tatsächlich die problematischste Operation ist.

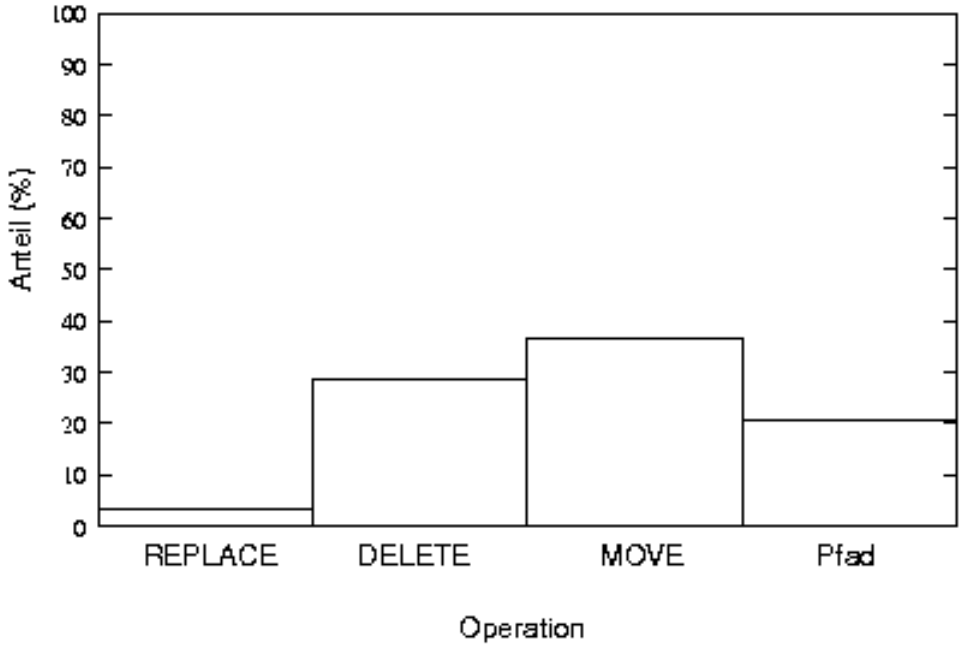


Abbildung 25: Verteilung der schlechten Operationen

Die nicht überragenden Ausbeute- und Präzisionswerte werden als ein Indiz dafür aufgefasst, dass das Modell die realen Verhältnisse nicht optimal abbilden kann. Die Gründe dafür können vielfältig sein: Möglicherweise ist das Problem zu komplex, die Datenmenge zu klein, die Parameter ungünstig eingestellt, die Abstraktion zu niedrig oder zu hoch. Es ist denkbar, dass ein grösseres Trainingskorpus, insbesondere kombiniert mit einer anderen Zusammenstellung der Kontextinformation, bessere Resultate liefert.

11 Schlussfolgerungen, Fragen und Ausblick

Es konnte gezeigt werden, dass es möglich ist, Syntaxbäume einer ersten Art Zielbäumen einer zweiten Art automatisch anzunähern, indem mit Markow-Modellen die generellen Unterschiede der beiden Arten repräsentiert werden und daraus Rückschlüsse für unbekannte Syntaxbäume der ersten Art angestellt werden können.

Es wurde ein solches System entwickelt und dessen Funktionsweise beschrieben. In der Trainingsphase des Systems wird eine Menge von Transformationen ermittelt. Daraus wird ein Markow-Modell erstellt, welches die Transformationen in Bezug auf lokale Baumstrukturen kodiert. Dieses Modell wird in der Anwendungsphase dazu verwendet, die wahrscheinlichsten Transformationen für einen beliebigen Baum der ersten Art zu berechnen. Schliesslich werden diese Transformationen auf den Baum angewendet, sodass ein der zweiten Art ähnlicherer Baum resultiert.

Die wesentlichen Überlegungen und Entscheidungen zu den wichtigsten Bestandteilen des Systems wurden beschrieben: Das Konzept des probabilistischen Modells in Abschnitt 7.2 (S. 45), der Baumvergleich in Abschnitt 8.3 (S. 57), in Abschnitt 8.1 (S. 53) die Editieroperationen, der Begriff der Transformation in Abschnitt 8.2 (S. 55), die lokale Baumstruktur in Form von Kontextinformationen (siehe Abschnitt 8.4 (S. 63)), der Modellbau in 8.5 (S. 64) und die -benutzung in 9.1 (S. 67) und die Anwendung der Editieroperationen in einem Syntaxbaum (siehe Abschnitt 9.2 (S. 70)).

Sollte ein Interesse an einer Übersicht über die Implementierungsarchitektur bestehen, sei auf den Anhang B (S. 99) verwiesen.

Die Leistung der Implementation dieses Systems wurde im Kapitel 10 (S. 80) getestet und bewertet. Als Quellkorpora wurde die Ausgabe vom Lopar-Parser und das NEGRA-Korpus verwendet. Das System funktioniert soweit gemäss Aufgabestellung. Es wäre interessant, die Ergebnisse mit einem anderen Parser zu vergleichen. Allerdings könnten die Ergebnisse im Allgemeinen nach genauerem Studium der Modellverhältnisse noch durch einige Erweiterungen und Neukonzipierungen verbessert werden.

Das im Abschnitt 8.4 (S. 63) angesprochene Problem, wie ein Überschreiten der Kontextinformation durch die MOVE-Pfadinformation vermieden werden kann, ist nach wie vor ungelöst. Ebenfalls nicht optimal ist der Umgang mit MOVE-Operationen, deren Zielknoten im aktuellen Baum nicht zugänglich ist und die deswegen auf der Wartebank sitzen (siehe Absatz 9.2.6 (S. 73)).

Es gibt noch einige Optionen, welche evtl. zu einer Verbesserung des Systems beitragen könnten, aber den Rahmen einer Lizenziatsarbeit sprengen würden.

Die Implementierung einer spezialisierten Methode, das Modell ausgeglichener zu gestalten, beispielsweise um Null-Wahrscheinlichkeiten zu verhindern, stünde an der ersten Stelle. Ein Back-off-Verfahren würde sich anbieten. Damit wäre sogar eine Lexikalisierung der Kontexte möglich, welche u.a. Hoffnung auf noch bessere Tagging-Genauigkeit weckt.

Das Entwerfen eines probabilistisch und linguistisch motivierten Kostenmodells für die Berechnung einer voll spezifizierten Editierdistanz stellt eine nächste Herausforderung dar, welche angepackt werden könnte.

Als Alternative zum Ein-Modell-Konzept ist auch eine Aufteilung der Arbeit auf mehrere Modelle sehr gut vorstellbar. Inspirationen hierzu könnte Brants (1999b) liefern (siehe Abschnitt 6.2.2 (S. 41)).

Teil III

Anhang

A NEGRA-Kategorien

A.1 Wortarten

Das NEGRA-Korpus benutzt das Stuttgart/Tübinger Tagset (STTS), um Wortarten zu bezeichnen.

Tag	Bedeutung	Beispiel
ADJA	attributives Adjektiv	[das] große [Haus]
ADJD	adverbiales oder prädikatives Adjektiv	[er fährt] schnell, [er ist] schnell
ADV	Adverb	schon, bald, doch
APPR	Präposition; Zirkumposition links	in [der Stadt], ohne [mich]
APPRART	Präposition mit Artikel	im [Haus], zur [Sache]
APPO	Postposition	[ihm] zufolge, [der Sache] wegen
APZR	Zirkumposition rechts	[von jetzt] an
ART	bestimmter oder unbestimmter Artikel	der, die, das, ein, eine, ...
CARD	Kardinalzahl	zwei [Männer], [im Jahre] 1994
FM	Fremdsprachliches Material	[Er hat das mit “] A big fish [” übersetzt]
ITJ	Interjektion	mhm, ach, tja
ORD	Ordinalzahl	[der] neunte [August]
KOUI	unterordnende Konjunktion mit “zu” und Infinitiv	um [zu leben], anstatt [zu fragen]
KOUS	unterordnende Konjunktion mit Satz	weil, daß, damit, wenn, ob
KON	nebenordnende Konjunktion	und, oder, aber
KOKOM	Vergleichskonjunktion	als, wie
NN	normales Nomen	Tisch, Herr, [das] Reisen
NE	Eigennamen	Hans, Hamburg, HSV
PDS	substituierendes Demonstrativpronomen	dieser, jener
PDAT	attribuierendes Demonstrativpronomen	jener [Mensch]
PIS	substituierendes Indefinitpronomen	keiner, viele, man, niemand

Tag	Bedeutung	Beispiel
PIAT	attribuierendes Indefinitpronomen ohne Determiner	kein [Mensch], irgendein [Glas]
PIDAT	attribuierendes Indefinitpronomen mit Determiner	[ein] wenig [Wasser], [die] beiden [Brüder]
PPER	irreflexives Personalpronomen	ich, er, ihm, mich, dir
PPOSS	substituierendes Possessivpronomen	meins, deiner
PPOSAT	attribuierendes Possessivpronomen	mein [Buch], deine [Mutter]
PRELS	substituierendes Relativpronomen	[der Hund ,] der
PRELAT	attribuierendes Relativpronomen	[der Mann ,] dessen [Hund]
PRF	reflexives Personalpronomen	sich, einander, dich, mir
PWS	substituierendes Interrogativpronomen	wer, was
PWAT	attribuierendes Interrogativpronomen	welche [Farbe], wessen [Hut]
PWAV	adverbiales Interrogativ- oder Relativpronomen	warum, wo, wann, worüber, wobei
PAV	Pronominaladverb	dafür, dabei, deswegen, trotzdem
PTKZU	“zu” vor Infinitiv	zu [gehen]
PTKNEG	Negationspartikel	nicht
PTKVZ	abgetrennter Verbzusatz	[er kommt] an, [er fährt] rad
PTKANT	Antwortpartikel	ja, nein, danke, bitte
PTKA	Partikel bei Adjektiv oder Adverb	am [schönsten], zu [schnell]
SGML	SGML Markup	
SPELL	Buchstabierfolge	S-C-H-W-E-I-K-L
TRUNC	Kompositions-Erstglied	An- [und Abreise]
VVFIN	finites Verb, voll	[du] gehst, [wir] kommen [an]
VVIMP	Imperativ, voll	komm [!]
VVINF	Infinitiv, voll	gehen, ankommen
VVIZU	Infinitiv mit “zu”, voll	anzukommen, loszulassen
VVPP	Partizip Perfekt, voll	gegangen, angekommen
VAFIN	finites Verb, aux	[du] bist, [wir] werden
VAIMP	Imperativ, aux	sei [ruhig !]

Tag	Bedeutung	Beispiel
VAINF	Infinitiv, aux	werden, sein
VAPP	Partizip Perfekt, aux	gewesen
VMFIN	finites Verb, modal	dürft
VMINF	Infinitiv, modal	wollen
VMPP	Partizip Perfekt, modal	gekonnt, [er hat gehen] können
XY	Nichtwort, Sonderzeichen enthaltend	3:7, H2O, D2XW3
$,	Komma	,
$.	Satzbeendende Interpunktion	. ? ! ; :
$(	sonstige Satzzeichen	; satzintern- [,]()

A.2 Phrasenkategorien

Tag	Bedeutung	Beispiel
AA	superlative Phrase mit “am”	Karl lachte [am lautesten], der [AP: [AA: am lautesten] lachende] Mann
AP	Adjektivphrase: Adjektiv (ADJA, ADJD, MTA, CARD) mit seinen Dependenzen	das [vom Repertoir her unerschrockene] Ensemble, die [an sich netten] Melodien, Peter ist [an sich ganz nett]
	Partizipialphrasen	der [nach Hamburg fahrende] Mann
	Modifizierte Artikel und Zahlen	[gar keine] Freude, [ca. 10] Jugendliche, [AP: rund [NM: 10000]] Jahre
AVP	Adverbialphrase	[AVP: gar nicht], [AVP: so] wichtig, [AVP: dass...]
CAC	Koordinierte Adpositionen (Prä- oder Postpositionen, APPR, APPO, APZR)	die Züge [von und nach] Hamburg
CAP	Koordinierte Adjektivphrase (AP, CAP, Adjektive)	die [CAP: alten und neuen] Ideen

Tag	Bedeutung	Beispiel
CAVP	Koordinierte Adverbialphrase (AVP, CAVP, ADV)	[CAVP: [AVP: nur heute] oder nie], [CAVP: heute und morgen]
CCP	Koordiniertes Komplement	[ob und wann] er kommt
CH	Chunk	
CNP	Koordinierte Nominalphrase (NP, NN, NE, MPN)	[CNP: [NP: ein Mann] und [NP: eine Frau]] [CNP: Peter und [NP: sein Onkel]], [NP: die [CNP: Jungen und Mädchen]]
CO	Koordination von verschiedenen Kategorien	[CO: [NP: Jeden Tag] oder [PP: zumindest am kommenden Freitag]]
CPP	Koordinierte Adpositionalphrase	[CPP: [in der Stadt] und [auf dem Lande]], [CPP: [für die Reformen] oder dagegen]
CS	Koordinierter Satz oder Satzchunks	[CS: [S: Peter kommt] und [S: Paul geht]], [CS: [S: Er kam], sah und siegte]
CVP	Koordinierte infinite Verbalphrase	Er wollte [CVP: [VP: Peter besuchen] und [VP: Hans anrufen]], Er wollte [CVP: [VP: uns besuchen] oder anrufen]
CVZ	Koordinierter Infinitiv mit “zu”	[[zu vergessen] und [zu vergeben]], [einzusteigen und [zu bleiben]]
DL	Ein Diskurselement, ohne explizite Abhängigkeiten zwischen den Komponenten	[[“Lass mich in Ruhe!”] [ärgerte sich Peter]]
ISU	Idiosynkratisches Element	
MPN	Eigenname aus mehreren Worten	[Karl Schulz], [Bad Münstereifel]
MTA	Adjektiv aus mehreren Worten	die [MTA: Bad Godesberger] Bürger
NM	Zahl aus mehreren Worten	[NP: [NM: eine Million] Menschen], [NP: [NM: 10000] Menschen]

Tag	Bedeutung	Beispiel
NP	Nominalphrase	[NP: der Mann], [NP: der alte Mann], [NP: der Mann [PP: aus Hamburg]]
PP	Präpositionalphrase (beachte: keine eingebetteten Nominalphrasen!)	[PP: in der Stadt], [PP: meiner Meinung nach], [PP: um die Stadt herum], [PP: dagegen, [dass er kommt]]
QL	Quasisprache	
S	Satz	[S: Hans liebt Maria], [S: Gut, [S: dass du kommst]]
VP	infinite Verbalphrase	Peter will [VP: uns besuchen]
VZ	Infinitiv mit "zu" (nur, wenn "zu" ein separates Wort ist)	Er versucht, uns [VZ: zu täuschen], ABER: Er versucht, wegzufahren=PROAV

A.3 NEGRA-Formate

Informationen zu den NEGRA-Formaten sind online unter der URL `http://www.coli.uni-sb.de/sfb378/negra-corpus/negra-corpus.html` erhältlich.

B Implementierung

Der Quellkode kann unter der URL `http://www.cl.unizh.ch/broder/liz/code` besichtigt werden.

Es folgt eine Übersicht, wie die wichtigsten Module zusammengefügt sind. Und welches Modul, welche Aufgaben erledigt. Grundsätzlich weist das Schlüsselwort `top` auf Koordinationsmodule hin.

B.1 Modulübersicht

In der Abbildung 26 auf der nächsten Seite sind die wichtigsten Zusammenhänge der meisten Module und die Unterverzeichnisse schematisch dargestellt.

B.1.1 Oberste Stufe

Die hierarchisch oberen Module laden die benötigten Module, um eine bestimmte Aufgabe zu erfüllen.

Das Modul `top` hält alle Einstellungen zum System fest. Die Einstellungen betreffen Dateinamen und Pfade und auch welche Kontextinformation benutzt werden soll. Ausserdem lädt `top` sowohl die Bibliotheksdateien von SICStus, als auch alle Bibliotheksdateien, die sich im Unterverzeichnis `libs` befinden.

`top_train` bereitet das Trainingsphase vor. Hier werden die Module `top`, `training`, `compare` und `transform` konsultiert.

`top_apply` konsultiert die Module, die für die Anwendungsphase erforderlich sind, das sind: `top`, `application`, `tags`, `model`, `transform`, `code` und `vit_rec`.

`top_tree_eval` erledigt die Baumevaluierung. Dazu werden die Bäume in ein bestimmtes Textformat in Klammerform umgewandelt und ausgegeben. Zum Evaluieren wird das externe Programm `evalb`[17] benutzt.

`top_eval` bereitet die Operations-Evaluierung vor. Dazu sind die Module `evaluation`, `top_train` und `top_apply` notwendig.

[17] von Satoshi Sekine (NYU) und Mike Collins (UPenn) 1997 siehe URL: `http://nlp.cs.nyu.edu/evalb/` – 18.2.2003

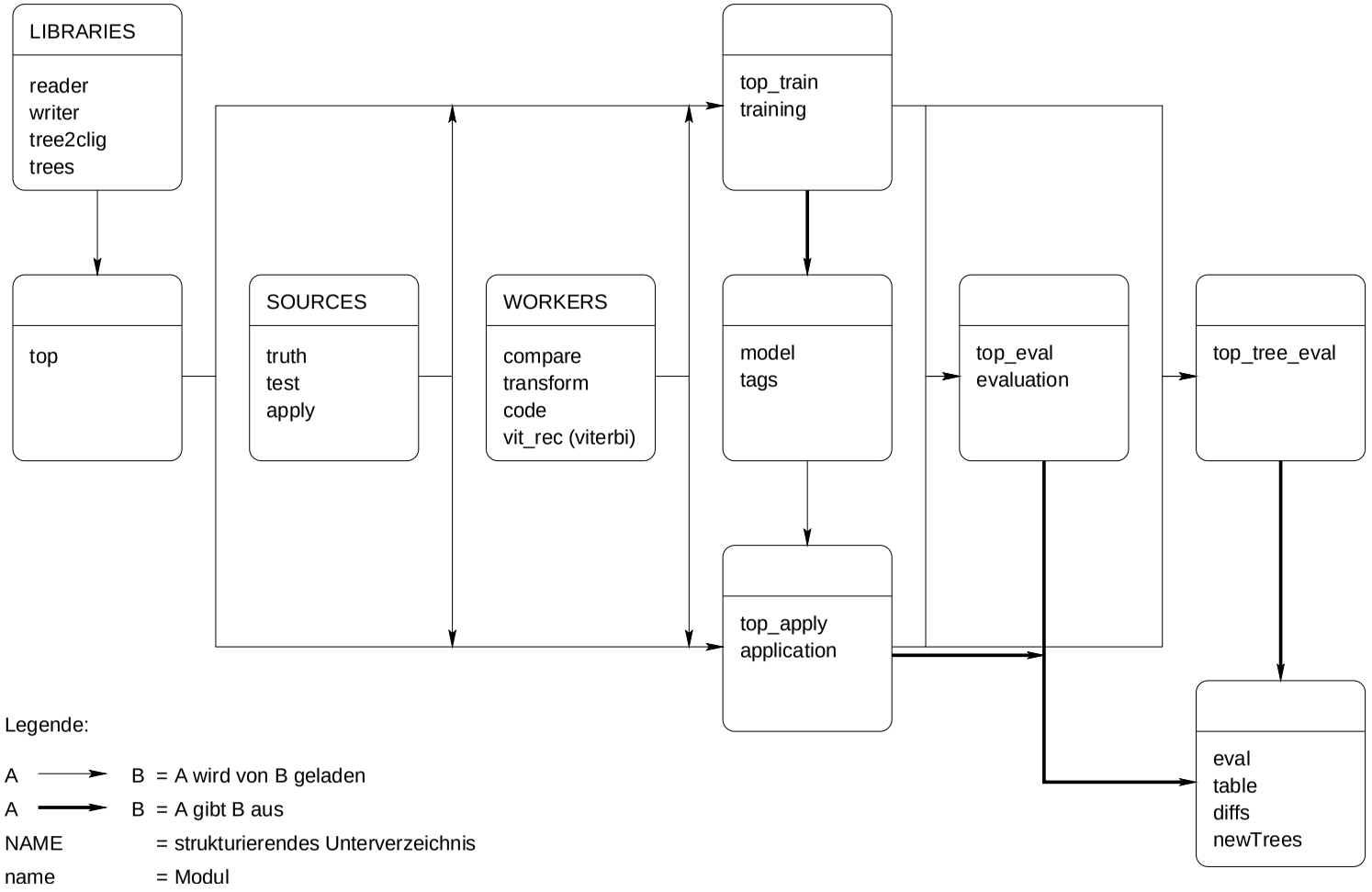


Abbildung 26: Die Unterverzeichnisse, die wichtigsten Module und ihre Zusammenhänge

Alle Ausgabedateien befinden sich auch in der obersten Ebene, um die Zugriffe möglichst einfach zu gestalten. Die wichtigsten Ausgabedateien sind:

`model` enthält das gesamte probabilistische Modell, welches in der Trainingsphase erstellt wird. Die Daten sind in den PROLOG-Prädikaten `outprob(Kontext, Transformation, AusgabeWahrscheinlichkeit)` und `transprob(Transformation1, Transformation2, UebergangsWahrscheinlichkeit)` enthalten. Die Kontexte und Transformationen sind als PROLOG-Atome kodiert, um die Indexierung von SICStus auszunutzen.

`tags` listet alle in der Trainingsphase angetroffenen Tags auf. Die Prädikate `tag(Tag, TagID)` und `numtag(TagID, Tag)` ordnen jedem Tag eine eindeutige Identifikationsnummer zu. Diese Information wird beim (De-)Kodieren benötigt.

B.1.2 Phasenmodule

Im Verzeichnis 'mains' sind alle Hauptprogramme situiert:

`training` organisiert das Training. Das Training besteht im Wesentlichen aus dem Vergleichen von Bäumen und dem Berechnen der Modellparameter.

`application` lässt die Anwendung ablaufen. Mit dem Viterbi-Algorithmus werden die wahrscheinlichsten Transformationen berechnet und schliesslich auf den Baum angewandt.

`evaluation` vergleicht die richtigen mit den wahrscheinlichsten Operationen.

Hauptprogramme organisieren den Ablauf einer Sitzung. Sie lesen Eingabematerial, schreiben die Ausgabedateien und enthalten die äusseren Programmschleifen.

B.2 Strukturierende Unterverzeichnisse

B.2.1 Bibliotheken

Im Unterverzeichnis `libs` sind alle Bibliotheksdateien, welche jederzeit zur Verfügung stehen. Bibliotheksdateien sind sehr allgemein gehalten. Sie beinhalten Prädikate, die von mehreren anderen Modulen benutzt werden.

reader stellt Routinen zur Verfügung, die Streams öffnen und schliessen und sehr grosse Dateien im ID/LP-Format satzweise einlesen und in einem PROLOG-Modul zur Verfügung halten.

writer wird für die Ausgabe von Dateien und Modulen benutzt.

tree2clig wandelt id/lp-Bäume in eine spezielle Notation für ein externes Programm namens `clig`[18] um, welches Bäume graphisch darstellen kann.

trees ist ein Modul, welches allerlei Operationen in und an Bäumen im ID/LP-Format anbietet. Die Anwendungsmöglichkeiten sind vielfältig, es kann beispielsweise einen Baum in Postorder durchlaufen oder Bäume kopieren, Kontextinformationen eines Baumes zusammensuchen, alle Kinder eines Knotens einsammeln und ordnen, Pfade von einem Knoten zum andern finden, die Editieroperationen anwenden etc.

B.2.2 Eingabekorpora

Die Korpora befinden sich im Verzeichnis `sources`. Die Quellen sind im ID/LP-Format. Es sind immer zwei Korpora miteinander aligniert, d.h. sie enthalten gleichviele Sätze in der gleichen Reihenfolge. Die NEGRA-Sätze wurden gefiltert, es sind keine überkreuzende Strukturen enthalten (siehe Kapitel 2.7 (S. 14)).

gojol_input und negra_input 600 NEGRA-Sätze, die auch vom GOJOL-Parser eine Struktur erhalten haben.

lopar und negra 2477 NEGRA-Sätze, die auch vom LOPAR-Parser eine Struktur erhalten haben.

B.2.3 Wichtige Arbeiter

Module, die wesentliche Arbeitsschritte durchführen sind vor allem im Verzeichnis `workers` zu finden.

compare übernimmt den Vergleich von zwei Bäumen.

transform führt die Baumtransformationen durch.

[18] von der Abteilung für Computerlinguistik der Universität Saarbrücken, siehe URL: `http://www.ags.uni-sb.de/~konrad/clig.html` – 18.2.2003

`code` kodiert alle Transformationen zu PROLOG-Atomen bzw. dekodiert alle Atome zu Transformationen, um eine schnellere Suche zu ermöglichen.

`vit_rec` berechnet mit dem Viterbi-Algorithmus den wahrscheinlichsten Pfad aus der Kontextinformation und dem Modell.

C Zusammenstellung der wichtigsten Evaluierungsdaten

- Zehnfache Kreuzvalidierung
- Transformation von Lopar nach NEGRA
- Grösse des Ausgangskorpus variiert, eine Kreuzvalidierung pro Grösse
- Kontext-Information: Beschriftung des aktuellen Knotens, der Mutter, der Grossmutter und der linken Schwester
- Die Baumvergleichswerte sind alle verschärfte Werte, d.h. mit Beschriftungskontrolle

In den folgenden Tabellen ist der Mittelwert 'Total Operationen' keine Prozentangabe, sondern der Mittelwert der absoluten Anzahl aller berechneten Operationen.

C.1 Grösse des Korpus: 100

Mass	**Wert**	
	Mittel (%)	**Standardabweichung**
Ausbeute	62.71	4.155
Präzision	45.44	9.705
F-Mass	52.70	
Exakte Übereinstimmung	6.00	9.661
Kein Überkreuzen	66.00	8.433
2 oder weniger Überkreuzen	87.00	8.233
Tagging-Genauigkeit	86.67	2.800
Total Operationen	630.10	79.745
Anteil gute Operationen	75.85	3.898
Anteil schlechte Operationen	7.43	1.525
Anteil indifferente Operationen	16.59	3.883
Anteil gute und indifferente	92.47	1.525
Anteil schlechte und indifferente	24.05	3.898
Anteil REPLACE von den schlechten	4.35	3.363
Anteil DELETE von den schlechten	31.51	5.680
Anteil MOVE von den schlechten	33.47	7.049
Anteil falscher Pfad von den schlechten	30.52	7.933

C.2 Grösse des Korpus: 200

Mass	Wert	
	Mittel (%)	**Standardabweichung**
Ausbeute	63.62	6.294
Präzision	55.41	6.559
F-Mass	59.23	
Exakte Übereinstimmung	12.08	7.925
Kein Überkreuzen	70.40	12.488
2 oder weniger Überkreuzen	93.47	5.289
Tagging-Genauigkeit	89.86	1.345
Total Operationen	1211.60	112.442
Anteil gute Operationen	80.29	2.054
Anteil schlechte Operationen	8.55	1.107
Anteil indifferente Operationen	11.01	1.622
Anteil gute und indifferente	91.35	1.107
Anteil schlechte und indifferente	19.61	2.054
Anteil REPLACE von den schlechten	7.24	2.522
Anteil DELETE von den schlechten	31.45	5.898
Anteil MOVE von den schlechten	32.89	7.267
Anteil falscher Pfad von den schlechten	28.22	7.234

C.3 Grösse des Korpus: 400

Mass	Wert	
	Mittel (%)	**Standardabweichung**
Ausbeute	64.50	3.552
Präzision	57.08	4.411
F-Mass	60.56	
Exakte Übereinstimmung	11.03	5.912
Kein Überkreuzen	70.40	9.175
2 oder weniger Überkreuzen	92.22	4.816
Tagging-Genauigkeit	91.75	1.999
Total Operationen	2387.00	142.734

Mass	Wert	
	Mittel (%)	Standardabweichung
Anteil gute Operationen	81.72	2.114
Anteil schlechte Operationen	8.34	0.782
Anteil indifferente Operationen	9.80	1.430
Anteil gute und indifferente	91.56	0.782
Anteil schlechte und indifferente	18.18	2.114
Anteil REPLACE von den schlechten .	5.68	1.253
Anteil DELETE von den schlechten ...	29.50	2.193
Anteil MOVE von den schlechten	33.36	2.081
Anteil falscher Pfad von den schlechten	31.30	1.420

C.4 Grösse des Korpus: 600

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	63.99	3.942
Präzision	59.35	4.407
F-Mass	61.58	
Exakte Übereinstimmung	10.26	4.290
Kein Überkreuzen	70.25	6.556
2 oder weniger Überkreuzen	95.12	3.047
Tagging-Genauigkeit	92.90	1.110
Total Operationen	3609.70	244.630
Anteil gute Operationen	82.84	1.407
Anteil schlechte Operationen	8.50	0.730
Anteil indifferente Operationen	8.50	0.921
Anteil gute und indifferente	91.40	0.730
Anteil schlechte und indifferente	17.06	1.407
Anteil REPLACE von den schlechten .	4.99	0.927
Anteil DELETE von den schlechten ...	29.22	2.208
Anteil MOVE von den schlechten	32.29	2.876
Anteil falscher Pfad von den schlechten	33.32	3.556

C.5 Grösse des Korpus: 800

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	64.28	2.214
Präzision	60.93	2.126
F-Mass	62.56	
Exakte Übereinstimmung	14.07	4.364
Kein Überkreuzen	73.86	4.781
2 oder weniger Überkreuzen	94.59	2.717
Tagging-Genauigkeit	93.58	0.776
Total Operationen	4842.20	121.476
Anteil gute Operationen	83.52	0.779
Anteil schlechte Operationen	8.59	0.534
Anteil indifferente Operationen	7.74	0.650
Anteil gute und indifferente	91.31	0.534
Anteil schlechte und indifferente	16.38	0.779
Anteil REPLACE von den schlechten	4.56	0.867
Anteil DELETE von den schlechten	29.77	1.708
Anteil MOVE von den schlechten	35.20	1.476
Anteil falscher Pfad von den schlechten	30.25	1.876

C.6 Grösse des Korpus: 1000

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	63.78	1.978
Präzision	60.71	1.893
F-Mass	62.21	
Exakte Übereinstimmung	11.95	2.481
Kein Überkreuzen	68.97	4.236
2 oder weniger Überkreuzen	93.88	2.787
Tagging-Genauigkeit	93.59	0.681
Total Operationen	6198.90	156.546

Mass	Wert	
	Mittel (%)	Standardabweichung
Anteil gute Operationen	83.32	1.126
Anteil schlechte Operationen	8.93	0.562
Anteil indifferente Operationen	7.62	0.764
Anteil gute und indifferente	90.97	0.562
Anteil schlechte und indifferente	16.58	1.126
Anteil REPLACE von den schlechten .	4.15	0.992
Anteil DELETE von den schlechten ...	27.56	1.461
Anteil MOVE von den schlechten	36.13	1.980
Anteil falscher Pfad von den schlechten	31.96	1.590

C.7 Grösse des Korpus: 1200

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	64.81	1.842
Präzision	62.39	2.096
F-Mass	63.58	
Exakte Übereinstimmung	11.24	2.544
Kein Überkreuzen	72.88	4.874
2 oder weniger Überkreuzen	95.89	2.144
Tagging-Genauigkeit	93.73	0.526
Total Operationen	7330.90	281.052
Anteil gute Operationen	83.81	0.874
Anteil schlechte Operationen	8.57	0.406
Anteil indifferente Operationen	7.50	0.544
Anteil gute und indifferente	91.33	0.406
Anteil schlechte und indifferente	16.09	0.874
Anteil REPLACE von den schlechten .	3.95	0.481
Anteil DELETE von den schlechten ...	28.88	2.029
Anteil MOVE von den schlechten	36.62	2.530
Anteil falscher Pfad von den schlechten	30.37	1.711

C.8 Grösse des Korpus: 1400

Mass	Wert	
	Mittel (%)	**Standardabweichung**
Ausbeute	65.94	1.983
Präzision	63.19	2.295
F-Mass	64.54	
Exakte Übereinstimmung	14.61	2.819
Kein Überkreuzen	73.01	3.257
2 oder weniger Überkreuzen	95.33	2.116
Tagging-Genauigkeit	94.02	0.749
Total Operationen	8433.80	263.779
Anteil gute Operationen	84.70	1.070
Anteil schlechte Operationen	8.25	0.649
Anteil indifferente Operationen	6.91	0.633
Anteil gute und indifferente	91.65	0.649
Anteil schlechte und indifferente	15.20	1.070
Anteil REPLACE von den schlechten	4.25	0.873
Anteil DELETE von den schlechten	28.11	1.289
Anteil MOVE von den schlechten	36.14	1.909
Anteil falscher Pfad von den schlechten	31.32	2.162

C.9 Grösse des Korpus: 1600

Mass	Wert	
	Mittel (%)	**Standardabweichung**
Ausbeute	65.30	1.014
Präzision	64.27	1.093
F-Mass	64.78	
Exakte Übereinstimmung	13.59	2.665
Kein Überkreuzen	72.18	2.847
2 oder weniger Überkreuzen	96.52	1.238
Tagging-Genauigkeit	94.51	0.527
Total Operationen	9658.20	478.662

Mass	Wert	
	Mittel (%)	Standardabweichung
Anteil gute Operationen	84.83	0.709
Anteil schlechte Operationen	8.44	0.467
Anteil indifferente Operationen	6.59	0.348
Anteil gute und indifferente	91.46	0.467
Anteil schlechte und indifferente	15.07	0.709
Anteil REPLACE von den schlechten .	2.98	0.879
Anteil DELETE von den schlechten ...	29.12	0.655
Anteil MOVE von den schlechten	37.46	1.560
Anteil falscher Pfad von den schlechten	30.23	1.555

C.10 Grösse des Korpus: 1800

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	65.31	2.306
Präzision	64.09	2.839
F-Mass	64.69	
Exakte Übereinstimmung	13.30	2.101
Kein Überkreuzen	70.27	4.412
2 oder weniger Überkreuzen	94.75	1.737
Tagging-Genauigkeit	94.28	0.652
Total Operationen	11065.60	501.276
Anteil gute Operationen	84.64	0.830
Anteil schlechte Operationen	8.40	0.365
Anteil indifferente Operationen	6.80	0.516
Anteil gute und indifferente	91.50	0.365
Anteil schlechte und indifferente	15.26	0.830
Anteil REPLACE von den schlechten .	3.34	0.517
Anteil DELETE von den schlechten ...	29.47	1.780
Anteil MOVE von den schlechten	36.91	2.324
Anteil falscher Pfad von den schlechten	30.07	2.329

C.11 Grösse des Korpus: 2000

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	65.58	1.739
Präzision	63.54	1.367
F-Mass	64.54	
Exakte Übereinstimmung	12.60	2.589
Kein Überkreuzen	72.31	2.848
2 oder weniger Überkreuzen	95.72	0.944
Tagging-Genauigkeit	94.46	0.545
Total Operationen	12356.00	444.692
Anteil gute Operationen	84.80	0.800
Anteil schlechte Operationen	8.22	0.405
Anteil indifferente Operationen	6.84	0.448
Anteil gute und indifferente	91.68	0.405
Anteil schlechte und indifferente	15.10	0.800
Anteil REPLACE von den schlechten	3.50	0.730
Anteil DELETE von den schlechten	29.24	1.007
Anteil MOVE von den schlechten	36.53	0.989
Anteil falscher Pfad von den schlechten	30.53	1.315

C.12 Grösse des Korpus: 2200

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	65.19	1.349
Präzision	63.00	1.864
F-Mass	64.08	
Exakte Übereinstimmung	12.40	1.654
Kein Überkreuzen	73.83	2.596
2 oder weniger Überkreuzen	96.65	0.722
Tagging-Genauigkeit	94.73	0.960
Total Operationen	13281.80	299.908

Mass	Wert	
	Mittel (%)	Standardabweichung
Anteil gute Operationen	84.68	1.307
Anteil schlechte Operationen	8.31	0.647
Anteil indifferente Operationen	6.89	0.732
Anteil gute und indifferente	91.59	0.647
Anteil schlechte und indifferente	15.22	1.307
Anteil REPLACE von den schlechten .	2.94	0.538
Anteil DELETE von den schlechten ...	29.50	1.547
Anteil MOVE von den schlechten	35.38	1.570
Anteil falscher Pfad von den schlechten	31.98	1.580

C.13 Grösse des Korpus: 2400

Mass	Wert	
	Mittel (%)	Standardabweichung
Ausbeute	66.00	1.333
Präzision	64.43	1.362
F-Mass	65.21	
Exakte Übereinstimmung	12.79	2.502
Kein Überkreuzen	73.62	3.558
2 oder weniger Überkreuzen	96.18	0.854
Tagging-Genauigkeit	94.40	0.700
Total Operationen	14639.30	319.146
Anteil gute Operationen	84.84	0.977
Anteil schlechte Operationen	8.47	0.514
Anteil indifferente Operationen	6.55	0.504
Anteil gute und indifferente	91.43	0.514
Anteil schlechte und indifferente	15.06	0.977
Anteil REPLACE von den schlechten .	3.55	0.493
Anteil DELETE von den schlechten ...	28.74	1.098
Anteil MOVE von den schlechten	36.60	1.521
Anteil falscher Pfad von den schlechten	30.91	1.297

C.14 Lernkurven

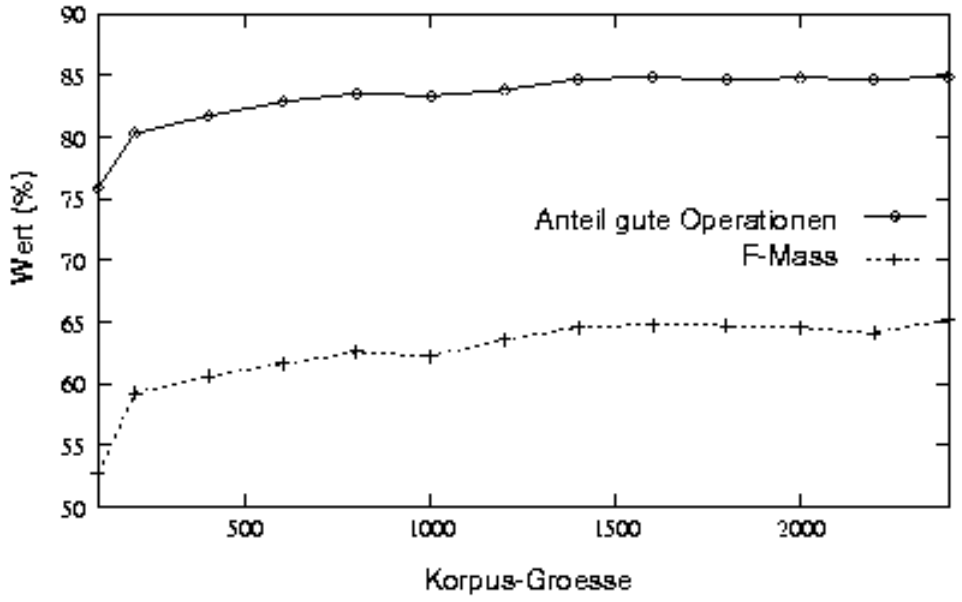


Abbildung 27: F-Mass und Gute Operationen

Literatur

[Barnard u. a. 1995] BARNARD, David T. ; CLARKE, Gwen ; DUNCAN, Nicholas: Tree-to-tree Correction for Document Trees / Department of Computing and Information Science, Queen's University. Kingston, 1995. – Forschungsbericht. – URL citeseer.nj.nec.com/barnard95treetotree.html

[Baum u. a. 1970] BAUM, L. E. ; PETRIE, T. ; SOULES, G. ; WEISS, N.: A maximization technique occurring in the statistical analysis of probabilistic functions in Markov chains. In: *The Annals of Mathematical Statistics* (1970), Nr. 41, S. 164–171

[Black u. a. 1991] BLACK, E. ; ABNEY, S. ; FLICKENGER, D. ; GDANIEC, C. ; GRISHMAN, R. ; HARRISON, P. ; HINDLE, D. ; INGRIA, R. ; JELINEK, F. ; KLAVANS, J. ; LIBERMAN, M. ; MARCUS, M. ; ROUKOS, S. ; SANTORINI, B. ; STRZALKOWSKI, T.: A procedure for quantitatively comparing the syntactic coverage of English grammars. In: *Proceedings of the Fourth DARPA Speech and Natural Language Workshop*, Februar 1991, S. 306–311

[Brants 1999a] BRANTS, Thorsten: Cascaded Markov Models. In: *Proceedings of 9th Conference of the European Chapter of the Association for Computational Linguistics EACL-99*. Bergen, Norway, 1999

[Brants 1999b] BRANTS, Thorsten: *Saarbrücken Dissertations in Computational Linguistics and Language Technology*. Bd. 6: *Tagging and Parsing with Cascaded Markov Models, Automation of Corpus Annotation*. Saarbrücken : Universität des Saarlandes. Deutsches Forschungszentrum für künstliche Intelligenz GmbH, DFKI, April 1999

[Brants und Samuelsson 1995] BRANTS, Thorsten ; SAMUELSSON, Christer: Tagging the Teleman Corpus. In: *Proceedings of the 10th Nordic Conference of Computational Linguistics NODALIDA-95*. Helsinki, Finland, 1995. – Also available as CLAUS Report 54

[Brill 1993] BRILL, E.: Transformation-Based Error-Driven Parsing. In: *Proceedings Third International Workshop on Parsing Technologies*. Tilburg/Durbuy, 1993. – URL citeseer.nj.nec.com/brill93transformationbased.html

[Brill 1994] BRILL, Eric: Some Advances in Transformation-Based Part of Speech Tagging. In: *National Conference on Artificial Intelligence*, URL citeseer.nj.nec.com/brill94some.html, 1994, S. 722–727

[Brill und Satta 1996] BRILL, Eric ; SATTA, Giorgio: Efficient Transformation-based Parsing. In: *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*, 1996

[Bußmann 1990] BUSSMANN, Hadumod: *Lexikon der Sprachwissenschaft*. 2. Auflage. Alfred Kröner Verlag, 1990

[Charniak 1997] CHARNIAK, Eugene: Statistical Techniques for Natural Language Parsing. In: *AI Magazine* 18 (1997), Nr. 4, S. 33–44. – URL citeseer.nj.nec.com/article/charniak97statistical.html

[Chawathe und Garcia-Molina 1997a] CHAWATHE, S. ; GARCIA-MOLINA, H.: An expressive model for comparing tree-structured data / Stanford University Database Group. Stanford, 1997. – Forschungsbericht. – URL citeseer.nj.nec.com/chawathe97expressive.html. Available at http://www-db.stanford.edu/

[Chawathe und Garcia-Molina 1997b] CHAWATHE, Sudarshan S. ; GARCIA-MOLINA, Hector: Meaningful change detection in structured data. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Tuscon, Mai 1997, S. 26–37. – URL citeseer.nj.nec.com/chawathe97meaningful.html

[Chawathe u. a. 1996] CHAWATHE, Sudarshan S. ; RAJARAMAN, Anand ; GARCIA-MOLINA, Hector ; WIDOM, Jennifer: Change detection in hierarchically structured information. In: *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Montréal, Québec, 1996, S. 493–504. – URL citeseer.nj.nec.com/chawathe96change.html

[Clematide 2002] CLEMATIDE, Simon: Selektive Evaluation von robusten Parsern. In: BUSEMANN, Stephan (Hrsg.): *Proceedings der 6. Konferenz zur Verarbeitung natürlicher Sprache*. Saarbrücken, September 2002 (KONVENS 2002 6), S. 23–29

[Collins 1997] COLLINS, Michael: Three Generative, Lexicalized Models for Statistical Parsing. In: COHEN, Philip R. (Hrsg.) ; WAHLSTER, Wolfgang (Hrsg.): *Proceedings of*

the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics. Somerset, New Jersey : Association for Computational Linguistics, 1997, S. 16–23. – URL citeseer.nj.nec.com/collins97three.html

[Collins 1996] COLLINS, Michael J.: A New Statistical Parser Based on Bigram Lexical Dependencies. In: JOSHI, Arivind (Hrsg.) ; PALMER, Martha (Hrsg.): *Proceedings of the Thirty-Fourth Annual Meeting of the Association for Computational Linguistics*. San Francisco : Morgan Kaufmann Publishers, 1996, S. 184–191. – URL citeseer.nj.nec.com/collins96new.html

[Elworthy 1994] ELWORTHY, D.: Does Baum-Welch re-estimation help taggers? In: *Proceedings of the fourth conference on applied natural language processing ANLP-94*. Stuttgart, 1994

[Gale und Church 1990] GALE, W. A. ; CHURCH, K. W.: Poor estimates of context are worse than none. In: *Proceedings of the speech and natural language workshop*. Hidden Valley, 1990, S. 283–287

[Good 1953] GOOD, I. J.: The population frequencies of species and the estimation of population parameters. In: *Biometrika* 40 (1953), S. 237–264

[Harrison u. a. 1991] HARRISON, P. ; ABNEY, S. ; BLACK, E. ; FLICKENGER, D. ; GDANIEC, C. ; GRISHMAN, R. ; HINDLE, D. ; INGRIA, R. ; MARCUS, M. ; SANTORINI, B. ; STRZALKOWSKI, T.: Evaluating syntax performance of parser/grammars of English. In: *Proceedings of the Workshop On Evaluating Natural Language Processing Systems*, Association For Computational Linguistics, 1991

[Isert 1999] ISERT, Carsten: *The Editing Distance Between Trees*. 1999. – URL citeseer.nj.nec.com/isert99editing.html

[Katz 1987] KATZ, Slava M.: Estimation of probabilities from sparse data for the language model component of a speech recognizer. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing ASSP* 35 (1987), Nr. 3, S. 400–401

[Krenn und Samuelsson 1997] KRENN, Brigitte ; SAMUELSSON, Christer: *The Linguist's Guide to Statistics – Don't Panic*. Dezember 1997. – URL http://www.coli.uni-sb.de/~krenn/stat\protect\T1\textunderscorenlp.ps.gz

[Lager 1997] LAGER, Torbjörn: *Spaghetti and HMMeatballs. Cooking a lowfat statistical tagger in Prolog*. 1997. – URL `http://www.ling.gu.se/~lager/Spaghetti/spaghetti.html`. – Unpubliziertes Manuskript

[Lektorat des BI-Wissenschaftsverlags unter der Leitung von Hermann Engesser 1993] LEKTORAT DES BI-WISSENSCHAFTSVERLAGS UNTER DER LEITUNG VON HERMANN ENGESSER (Hrsg.): *Duden 'Informatik' : ein Sachlexikon für Studium und Praxis*. 2. vollst. überarb. und erw. Auflage. Mannheim and Leipzig and Wien and Zürich : Dudenverlag, 1993

[Manning und Schütze 1999] MANNING, C. ; SCHÜTZE, H.: *Foundations of Statistical Natural Language Processing*. Cambridge, MA : MIT Press, 1999. – In Press. Draft available at: http://www.sultry.arts.usyd.edu.au/fsnlp

[Nierman und Jagadish 2002] NIERMAN, Andrew ; JAGADISH, H. V.: Evaluating Structural Similarity in XML Documents. In: *Proceedings of the Fifth International Workshop on the Web and Databases (WebDB 2002)*. Madison, Wisconsin, USA, Juni 2002. – URL `citeseer.nj.nec.com/nierman02evaluating.html`

[Samuelsson 1993] SAMUELSSON, C.: Morphological tagging based entirely on Bayesian inference. In: *9th nordic conference on computational linguistics NODALIDA-93*. Stockholm, 1993

[Schmid 2000] SCHMID, Helmut: LoPar: Design and Implementation. In: *Arbeitspapiere des Sonderforschungsbereiches 340*. IMS Stuttgart, Juli 2000 (149). – URL `http://www.ims.uni-stuttgart.de/projekte/gramotron/SOFTWARE/LoPar-en.ht\%ml`

[Selkow 1977] SELKOW, Stanley M.: The tree-to-tree editing problem. In: *Information Processing Letters* 6 (1977), Dezember, Nr. 6, S. 184–186

[Skut 1999] SKUT, Wojciech: *Saarbrücken Dissertations in Computational Linguistics and Language Technology*. Bd. 10: *Partial Parsing for Corpus Annotation and Text Processing*. Universität des Saarlandes. Deutsches Forschungszentrum für künstliche Intelligenz GmbH, DFKI, April 1999

[Skut und Brants 1998] SKUT, Wojciech ; BRANTS, Thorsten: Chunk Tagger – Statistical Recognition of Noun Phrases. In: *Proceedings of the ESSLLI Workshop on Automated Acquisition of Syntax and Parsing*. Saarbrücken, Germany, 1998

[Skut u. a. 1998] SKUT, Wojciech ; BRANTS, Thorsten ; KRENN, Brigitte ; USKOREIT, Hans: A Linguistically Interpreted Corpus of German Newspaper Text. In: *ESSLLI-98 Workshop on Recent Advances in Corpus Annotation*. Saarbrücken, 1998

[Skut u. a. 1997] SKUT, Wojciech ; KRENN, Brigitte ; BRANTS, Thorsten ; USKOREIT, Hans: An Annotation Scheme for Free Word Order Languages. In: *Proceedings of the Fifth Conference on Applied Natural Language Processing (ANLP)*. Washington, D.C., 1997

[Stolcke und Omohundro 1993] STOLCKE, Andreas ; OMOHUNDRO, Stephen: Hidden Markov Model Induction by Bayesian Model Merging. In: HANSON, Stephen J. (Hrsg.) ; COWAN, Jack D. (Hrsg.) ; GILES, C. L. (Hrsg.): *Advances in Neural Information Processing Systems* Bd. 5, Morgan Kaufmann, San Mateo, CA, 1993, S. 11–18. – URL `citeseer.nj.nec.com/stolcke93hidden.html`

[Tai 1979] TAI, Kuo-Chung: The Tree-to-Tree Correction Problem. In: *Journal of the ACM (JACM)* 26 (1979), Nr. 3, S. 422–433. – ISSN 0004-5411

[Thielen und Schiller 1995] THIELEN, Christine ; SCHILLER, Anne: Ein kleines und erweitertes Tagset fürs Deutsche. In: *Tagungsberichte des Arbeitstreffens Lexikon + Text 17./18. Februar 1994, Schloss Hohentübingen*. Tübingen : Niemeyer, 1995 (Lexicographica Series Maior)

[Viterbi 1967] VITERBI, Andrew J.: Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. In: *IEEE Transaction on Information Theory* (1967), S. 260–269

[Wagner und Fischer 1974] WAGNER, Robert A. ; FISCHER, Michael J.: The String-to-String Correction Problem. In: *Journal of the ACM (JACM)* 21 (1974), Nr. 1, S. 168–173. – ISSN 0004-5411

[Zhang und Shasha 1989] ZHANG, K. ; SHASHA, D.: Simple fast algorithms for the editing distance between trees and related problems. In: *SIAM Journal on Computing* 18 (1989), Dezember, Nr. 6, S. 1245–1262